CIS 101 Study Guide
Winter 2015-16

Don Colton
Brigham Young University–Hawai'i

February 10, 2016

Short Cuts

Study Guide

This is the official study guide for the CIS 101 class, Beginning Programming, as taught by Don Colton, Winter 2015-16. It is focused directly on the grading of the course.

http://byuh.doncolton.com/cis101/2161/sguide.pdf is the study guide, which is this present document. **It will be updated frequently throughout the semester,** as new assignments are made, and as due dates are established, and as clarifications are developed.

Syllabus

http://byuh.doncolton.com/cis101/2161/syl.pdf is the official syllabus for this course. It is largely reproduced in Chapter 1 (page 3) below.

Textbook

This study guide is a companion to the textbook for the class, Introduction to Programming Using Perl and CGI, Third Edition, by Don Colton.

The textbook is available here, in PDF form, free.

http://ipup.doncolton.com/

The textbook provides explanations and understanding about the content of the course.

Contents

Chapter 1

Syllabus

The original, separate syllabus is the official version. A copy of it is included here for (a) convenience, (b) as a place to correct minor errors that may be in the original, and (c) because this study guide has an index but the syllabus alone does not.

Contents

1.1 Overview

Computers are great. But they are also really stupid.

By stupid, I mean computers only understand really simple commands. Anything complex must be built up out of these simple commands.

Programming is the art of building up the fun and interesting things that you want to be done, starting from just the really simple commands that the computer can understand.

Sometimes it is frustrating. Sometimes it is very satisfying.

This class teaches skills by which you can better serve those around you, and by which you can get a good job.

There are many fine programming languages. Our programming language will be Perl.

1.1.1 General Education Breadth

CIS 101 satisfies the BYUH General Education Breadth of Knowledge requirement in Science and Technology.

The Breadth of Knowledge categories are intended to give people (majors) credit for something they are already taking in their major area, and give others (explorers) a chance to explore a broader range of fields.

My goal is to serve both majors and explorers well. If you have suggestions, please let me know.

1.1.2 Expected Proficiencies

As you begin this course, we assume you have no programming experience whatsoever. We expect you can read, type, send and receive email, and visit web sites. We will teach you everything else you need to know.

Ideally you will have your own personal computer, probably a laptop, on which you can write and test programs. (You can also use the computers in the CIS department labs.) The textbook tells how to install Perl on a Windows machine, and how to find it (pre-installed) on a Macintosh.

If you are worried that you may not learn quickly enough, you should attend the special lab held right after class each day. See section 1.3.1 (page 8) below.

1.2 Course and Faculty

1.2.1 Course Information

- **Title:** Beginning Programming
- **Course Number:** CIS 101
- **Course Description:** (from the catalog) Extensive hands-on software development and testing using variables, arrays, instruction sequences, decisions, loops, and subroutines. May also include dynamic web pages (CGI) and regular expressions.

- **Prerequisites:** none
- **Semester/Year:** Winter 2015-16
- **Semester Code:** 2161
- **Meeting Time:** MWF 13:20 to 14:20
- **Location:** GCB 111
- **First Day of Instruction:** Mon, Nov 09
- **Last Day to Drop:** Fri, Dec 11
- **Last Day to Withdraw:** Wed, Jan 20
- **Last Day for Late Work:** Fri, Feb 19
- **Last Day of Instruction:** Mon, Feb 22
- **Final Exam:** Wed, Feb 24, 13:00 to 15:50

1.2.2   Faculty Information

- **Instructor:** Don Colton
- **Office Location:** GCB 128
- **Office Hours:** (In GCB 111) MWF 14:30-15:00, TuTh 15:30-16:00.
- **Email:** doncolton2@gmail.com
- **Campus Homepage:**
  http://byuh.doncolton.com/ is my campus homepage. It has my calendar and links to the homepages for each course I teach.
- **Off-Campus Homepage:**
  http://doncolton.com/ is my off-campus homepage.

I have reserved GCB 111 on MWF from 14:30 to 15:30 (right after my CIS 101 class) and on TuTh from 15:30 to 17:00 (right after my IT 240 class) so my students (and others) can study in a lab setting and meet with me and each other. I will be there at the start of those hours, and will stay as long as students are asking me questions. I also allow the room as an Open Lab for your use either individually or in groups, for my class or for other classes.

1.2.3   Course Readings and Materials

- **Textbook:** (free!)
  http://ipup.doncolton.com/ Introduction to Programming Using Perl and CGI, by Don Colton.
- **Learning Management System:**
  https://dcquiz.byuh.edu/ is the learning management system for

my courses.

- **Course Homepage:**
  <http://byuh.doncolton.com/cis101/> is my course homepage. It has links to many things including the syllabus, study guide, and textbook.
- **Study Guide:**
  <http://byuh.doncolton.com/cis101/2161/sguide.pdf> is the study guide for this course. It includes a copy of most parts of this syllabus. The study guide is updated frequently throughout the semester as assignments are made and deadlines are established or updated.

1.3 Calendar

Mon Nov 09	o1H First Webpage
Wed Nov 11	U1: Output; g11 Hello
Fri Nov 13	o1R Random Number; PQ
Mon Nov 16	U2: Input; g21 Hi Fred
Wed Nov 18	o1S Slot Machine
Fri Nov 20	c2M Mad Lib; PQ
Mon Nov 23	U3: Math; g31 Before After
Wed Nov 25	g35 Age; **Exam v0 prac**
Fri Nov 27	Thanksgiving - No Class
Mon Nov 30	o2M Mad Lib
Wed Dec 02	U4: Decisions; g40 Gift1
Fri Dec 04	**Exam v1**
Mon Dec 07	g41 Scholarship
Wed Dec 09	o4H High Low
Fri Dec 11	g42 Birthday; LD Drop; PQ
Mon Dec 14	o4P Presents
Wed Dec 16	U5: Decisions; g51 Phone Book
Fri Dec 18	**Exam v2**
Mon Jan 04	U6: Loops; g62 For Loop
Wed Jan 06	o6D Multi Dice
Fri Jan 08	g64 Factors
Mon Jan 11	g66 Triangle
Wed Jan 13	g46 Cheese
Fri Jan 15	**Exam v3**
Mon Jan 18	Human Rights Day - No Class
Wed Jan 20	o6M Maze; LD Withdraw

Fri Jan	22	U7: Arrays; g74 Split Date
Mon Jan	25	g73 Boring
Wed Jan	27	g76 Birthday
Fri Jan	29	**Exam v4**
Mon Feb	01	g77 AM/PM Time to 24H
Wed Feb	03	g79 Perfect Shuffle
Fri Feb	05	g45 Gift2
Mon Feb	08	U8: Subroutines; o8F Farm8
Wed Feb	10	g34 Celsius
Fri Feb	12	**Exam v5**
Mon Feb	15	Presidents Day - No Class
Wed Feb	17	g7A Secret Codes
Fri Feb	19	g47 Appointment
Mon Feb	22	oJS JavaScript
Wed Feb	24	**Exam v6**, 13:00 to 15:50, GCB 111

We meet 38 times including the final exam.

Exam dates are firm and will not change unless there is a massive emergency. Exams are closed-book, closed-notes, closed-neighbor, etc. You can bring blank paper.

Other activities are likely but subject to change.

1.3.1   Special Lab

As noted above, I have reserved GCB 111 on MWF from 14:30 to 15:30 (right after my CIS 101 class) and on TuTh from 15:30 to 17:00 (right after my IT 240 class) so my students (and others) can study in a lab setting and meet with me and each other. I will be there at the start of those hours, and will stay as long as students are asking me questions. I also allow the room as an Open Lab for your use either individually or in groups, for my class or for other classes.

Many students finish their activity assignments during the regular class hour, or during the first few minutes of the special lab that is held right after class each day. This saves them from having to do as much homework outside of class.

You can actually register for the special lab hour. It is called CIS 101L, and is a zero-credit class where everyone receives a grade of "P" (pass). The main reason to register is to reserve that time on your schedule, so you know

you will be able to attend as needed.

Everyone is welcome to attend whether they registered for CIS 101L or not.

1.4 Grading

I use a 60/70/80/90 model based on 1000 points.

Based on 1000 points

930+	A	900+	A–	870+	B+
830+	B	800+	B–	770+	C+
730+	C	700+	C–	670+	D+
630+	D	600+	D–	0+	F

The 1000 points are divided up as follows.
• Attendance 38+3 points.
• Quizzes (3MQ) 60 points.
• Activities 270 points.
• Exams 630 points.

If you want to major in CS, IS, or IT, you need to earn a C or better (730 points or more) in this class.

1.4.1 CIS 101 Grade Books

In my Learning Management System (DCQuiz), I keep several online grade books so you can see how your points are coming along. This lets you compare yourself with other students in the class (without seeing their names).

2161 CIS 101 Overall Grade Book: This includes the totals from all the other grade books. This is where you can find your final grade at the end of the course.

2161 CIS 101 (whatever) Grade Book shows your points in the (whatever) category. (whatever) is Attendance, 3MQ, Activities, or Exam.

1.4.2 Attendance (38+3 points)

Each day in class starts with the "daily update" (DU). It is my way of reminding you of due dates and deadlines, sharing updates and news, and

taking roll. It is your way of saying something anonymously to each other and to me. It must be taken in class at a classroom computer during a window of time that starts a few minutes before class and ends 5 minutes into class.

Attendance: You must attend to earn the Attendance points. You must attend to earn the Three-Minute Quiz points. You must attend to earn the exam points. Besides that, there is no penalty for being late or lack of attendance. You can do the activities without attending, for instance.

2161 CIS 101 Attendance Grade Book shows your attendance points, one point per day, for 38 days. You get one point for each time you do the daily update. If you arrive too late to complete the daily update, you will not receive the attendance points for that day.

Tardiness: My tardiness policy is that you should arrive in time to complete the daily update. Generally if you are less than four minutes late, you will have time to complete the daily update before the deadline.

+3: In addition to actual attendance, there are a few other things that are recorded here for tracking purposes, including completion of the email assignment and the policy test.

1.4.3   Policy Test

To ensure that each student is aware of the important policies I have for this course, I have prepared a policy test covering things you are required to know.

Mostly it has true/false questions about acceptable behaviors and cheating. I want to know that you know where the line is.

You must pass this test with a perfect (100%) score in order to pass this class. You will be given all the questions and answers in advance. You will have several chances to pass the policy test. The deadline for passing the policy test is Fri, Dec 11, which is also the last day to drop the class.

1.4.4   3MQ (60 points)

There are assigned readings. These are listed in the course calendar and again right here. Right after prayer on many days (but not exam days), there will be a three-minute quiz. It will consist of three randomly chosen

questions from the assigned readings or earlier readings.

There are about 31 three-minute quizzes.

Readings Due Dates:
 Wed Nov 11 U1: Output
 Mon Nov 16 U2: Input
 Mon Nov 23 U3: Math
 Wed Dec 02 U4: Decisions (Numeric)
 Wed Dec 16 U5: Decisions (String)
 Mon Jan 04 U6: Loops
   Fri Jan 22 U7: Arrays
 Mon Feb 08 U8: Subroutines

You might find it helpful to complete the readings earlier. Information from the readings may be helpful in doing assignments that happen before the readings are due.

The three-minute quizzes are "closed book," by which I mean that you are not allowed to look up answers while you are taking the quiz.

Your scores from these 3MQ Quizzes will be recorded in **2161 CIS 101 3MQ Grade Book**. The total from this grade book will be rescaled (curved) so the top score is worth 60 points.

Toward the end of the semester there will be five make-up quizzes which can be used to replace quizzes you have missed, or replace your lowest score on a quiz you took.

1.4.5   Activities (270 points)

On most days we will have an in-class activity assignment. Each will normally be worth 10 points. Roughly 32 assignments x 10 points = 320 points. That means that you can skip roughly five of the activities, or you can do them for extra credit.

2161 CIS 101 Activities Grade Book tracks your performance on activities.

Assignments are correct when they work properly and comply with all other stated requirements (such as style and algorithm). Points are assigned according to the date on which the correct work is received. Details are provided in the study guide, but in general it works like this:

11: Correct by 23:59 the same class day it was introduced, if it has proper style. This includes a one point (ten percent) bonus.

10: Correct by 23:59 of the next class day. This is full credit.

9: Correct by 23:59 of the next class day (one class day late). This is partial credit.

8: Correct by 23:59 of the next class day (two class days late). This is partial credit.

7: Correct by the specified late work deadline. Each assignment has its own late work deadline, which is generally about a month after the assignment was introduced, but not later than Fri, Feb 19, the last day for late work. This is partial credit for being late.

Exam Question 1 (p.164)**:** Can most activities be submitted early for extra credit?
Required Answer: yes

Exam Question 2 (p.164)**:** Can most activities be submitted late for partial credit?
Required Answer: yes

Exam Question 3 (p.164)**:** Can most activities be submitted after the deadline?
Required Answer: no

1.4.6   Final Exams (630 points)

You get six chances to pass the final exam. There are 21 sections to the exam. You should pass each section at least once.

Final Exam Dates:

Wed Nov 25	Exam v0 (Practice)	
Fri Dec 04	Exam v1	
Fri Dec 18	Exam v2	
Fri Jan 15	Exam v3	
Fri Jan 29	Exam v4	
Fri Feb 12	Exam v5	
Wed Feb 24	Exam v6	3hr Final
Fri Feb 26	Exam v7	3hr Make-up

The make-up exam is not available unless you get special approval in ad-

vance.

There are 21 exam tasks. Each is a program for you to do during one of the final exams. Each is worth 30 points. Points for each question can be earned only once.

Each exam is a "final exam" in the sense that it covers everything we learn during the semester, and by completing it, you earn the points for it as though you had done it on the day of the actual final. One practice exam is also given, for no credit, to help you understand how to do the other tests.

(630) Exam Points (21 tasks)
- 1 30p String Basic
- 2 30p Number Basic
- 3 30p Number Story
- 4 30p Number Decision
- 5 30p Number Decision Story
- 6 30p String Decision
- 7 30p String Decision Bracket
- 8 30p Repeat While
- 9 30p Repeat For
- 10 30p Repeat Last
- 11 30p Repeat Nested
- 12 30p Lists Basic
- 13 30p Lists Loop
- 14 30p Arrays Basic
- 15 30p Arrays Loop
- 16 30p Split
- 17 30p Join
- 18 30p Subroutine Returns
- 19 30p Subroutine Positional Parameters
- 20 30p Subroutine Globals and Locals
- 21 30p Subroutine Variable Parameters

Grading for each question is all-or-nothing. There is no partial credit for exam programs.

The study guide talks more about each of these tasks.

1.4.7 Other Extra Credit

Extra credit is available for reporting an error in my formal communications (the published materials I provide), so I can fix it. Send me an email. Cut and paste what I said (enough so I can easily find it). Tell me why you think it is wrong, or what it should be.

In this class, the formal materials include the following:

- The course website, parts relating to this semester.

- The course syllabus.

- The course study guide.

- The course textbook, since I wrote it.

Each error reported can earn you extra credit. (Typos in my email messages do not count.)

Syllabus errors (unless they are major) will probably be fixed only in the study guide. Check there before reporting it.

1.5 Communication

We communicate with each other both **formally** and **informally.**

Formal communication is official, carefully worded, and normally in writing. Formal is for anything truly important, like grades.

Informal communication is casual and impromptu. It is meant to be helpful and efficient. Reminders are informal. Explanations are usually informal.

Email can go either way.

If the formal and the informal do not agree with each other, trust the formal but also let me know so I can correct any errors.

1.5.1 From Me to You, Formal

I communicate formally, in writing, through (a) the syllabus, (b) the study guide, and (c) the learning management system.

(a) Syllabus: http://byuh.doncolton.com/cis101/2161/syl.pdf is the syllabus for this course. It tells our learning objectives and how you will be graded overall. You can rely on the syllabus. After class begins, it is almost never changed except to fix major errors.

(b) Study Guide: http://byuh.doncolton.com/cis101/2161/sguide.pdf is the study guide for this course. It has an index. It includes a copy of the syllabus. The study guide may be updated during the semester, as assignments are made and deadlines are established or updated.

(b1) Calendar: The study guide tells when things will happen. It contains specific due dates.

(b2) Assignments: The study guide tells what assignments have been made and how you will be graded, item by item. It provides current details and specific helps for each assignment. It may provide guidance for taking the exams.

(c) DCQuiz: https://dcquiz.byuh.edu/ is my learning management system. I use it to give tests. I use it to show you my grade books.

1.5.2 From Me to You, Informal

My main informal channels to you are (a) word of mouth and (b) email.

(a) Word of Mouth, including Lecture: Class time is meant to be informative and helpful. But if I say anything truly crucial, I will also put it into the study guide.

(b) Email: My emails to you are meant to be helpful. But if I say anything truly crucial, I will also put it into the study guide. Normally I put CIS 101 at the front of the subject line in each email I send.

1.5.3 From You to Me, Email

Rule 1: Send emails to **doncolton2@gmail.com**

Rule 2: Put **cis101** in your email subject line.

Sometimes I get buried in email, especially at the end of the semester. I do not want to overlook your email to me, or have it end up caught in my spam filter. And for my own sanity I want to be able to find and deal with all the email related to this class at the same time. This is especially true for large

classes.

My solution is to have you put **cis101** (exactly, with no spaces in it) in your email subject line, preferably as the first word. If you do this, my email system will immediately and automatically respond to you, telling you that I got your email and it is in my queue.

Exam Question 4 (p.164)**:** When sending Prof Colton email about CIS 101, should I put cis101 in the subject line?

Required Answer: yes

Exam Question 5 (p.164)**:** When sending Prof Colton email about CIS 101, should I put CIS 101 in the subject line?

Required Answer: no

If you fail to do this or do it wrong, you will not get an immediate reply and your email will end up in some other queue in my work flow. Your email will not be noticed when I am grading for this class. In the best case I will eventually handle your email. In the worst case your email will be in my spam folder and I will never even see it.

You also should include your roll-book name in the subject line. That is normally your lastname, comma, firstname.

Exam Question 6 (p.164)**:** When sending Prof Colton email about CIS 101, should I put my roll-book name in the subject line?

Required Answer: yes

1.5.4   From You to Me, Formal

Your formal channels to me, specifically how you turn in class work, are mainly via (a) the learning management system, (b) email, and (c) specifically requested projects.

(a) DCQuiz: To use my learning management system, you must log into it. Then, you can respond to questions I have posted. Each day there will be a "daily update". I say more on that elsewhere. Tests will also be given using DCQuiz.

(b) Email: You may be asked to use email in a formal way to submit some of the work you create and to tell me certain other things. The study guide tells how to send formal emails, including where to send them, what subject line to use, and what to put in the body of the message.

1.5.5   From You to Me, Informal

Your informal channels to me, typically how you ask questions and get assistance, are mainly face to face and by email or chat.

Face to Face: If you need help with your class work, I am happy to look at it and offer assistance. Often this happens during class, during open labs, or during office hours. Often I will have you put your work on your computer screen, and then I will take a look at it while we talk face to face.

Email / Chat: You can also get assistance by sending me an email or doing a chat. I will do my best to respond to it in a reasonable and helpful way. If you want something formal, use the formal rules.

If you are writing about several different things you will usually get a faster response if you break it up into several smaller emails instead of one big email. I try to respond to a whole email at once, and not just part of it. I usually answer smaller and simpler emails faster than big ones.

1.6   Instructional Methods

Exams happen on scheduled exam days. This instructional method brings you face to face with the challenges you need to be able to solve.

Lectures happen occasionally. I review material that was assigned from the text book and do what I can to make it clear and interesting. These can be short or take up most of the class hour. Longer ones happen more often at the start of the course than they do later on.

Activity days are usually the most common. A learning activity is assigned. Typically it is a program to be written. The program will be described in the study guide. I often give an overview of the problem and the techniques that I think will be helpful to solve it. Typically this takes about 15 minutes, but the actual time varies widely. Then I sit down at the front of the room and invite students to visit with me, one on one, for assistance. Students are also encouraged to help each other.

Help in Class: As students come to visit with me, I call up their computer screen from the place they were sitting, and we look at their program code or whatever else the student is asking about. We review the situation together. The student then returns to work on their program at their seat and I work with the next student waiting in line.

I want to help as many students as I can. You can help by doing these kinds of things before coming up.

(a) If asking about an assignment, have my study guide available in a tab on your browser, turned to the relevant assignment so we can review the requirements.

(b) If asking about grades, have my grade book available.

(c) If asking about an exam question, have that exam available.

1.6.1 BYUH Learning Framework

I agree with the BYUH Framework for Learning. If we follow it, class will be better for everyone.

Prepare for CIS 101

Prepare: Before class, study the course material and develop a solid understanding of it. Try to construct an understanding of the big picture and how each of the ideas and concepts relate to each other. Where appropriate use study groups to improve your and others' understanding of the material.

In CIS 101: Make reading part of your study. There is more than we could cover in class because we all learn at different rates. Our in-class time is better spent doing activities and answering your questions than you listening to a general lecture.

Engage in CIS 101

Engage: When attending class actively participate in discussions and ask questions. Test your ideas out with others and be open to their ideas and insights as well. As you leave class ask yourself, "Was class better because I was there today?"

In CIS 101: Participate in the in-class activities. As you make progress, assist those around you that want assistance. It is wonderful what you can learn by trying to help someone else.

Improve at CIS 101

Improve: Reflect on learning experiences and allow them to shape you into a more complete person: be willing to change your position or perspective on a certain subject. Take new risks and seek further opportunities to learn.

In CIS 101: After each exam, I normally allow you to see every answer submitted, every score given, and every comment I wrote, for every question, until the next exam happens. Compare your answers to those of other students. See how your answers could be improved. If you feel lost, study the readings again or ask for help.

1.6.2   Support

The major forms of support are (a) open lab, (b) study groups, and (c) tutoring.

If you still need help, please find me, even outside my posted office hours.

Office Hour / Open Lab

Office hours are MWF 14:30-15:00, TuTh 15:30-16:00.

See section 1.3.1 (page 8) for information on the open lab.

Study Groups

You are encouraged to form a study group. If you are smart, being in a study group will give you the opportunity to assist others. By assisting others you will be exposed to ideas and approaches (and errors) that you might never have considered on your own. You will benefit.

If you are struggling, being in a study group will give you the opportunity to ask questions from someone that remembers what it is like to be totally new at this subject. They are more likely to understand your questions because they sat through the same classes you did, took the same tests as you did, and probably thought about the same questions that you did.

Most of us are smart some of the time, and struggling some of the time. Study groups are good.

Tutoring

The CIS department provides tutoring in GCB 111, Monday through Friday, typically starting around 17:00 and ending around 23:00 (but earlier on Fridays). Normally a schedule is posted on one of the doors of GCB 111.

Exam Question 7 (p.164)**:** Does the CIS department provide tutoring for this class?
Required Answer: yes

Tutors can be identified by the red vests they wear when they are on duty.

1.7   Course Policies

Subject to Change: Like all courses I teach, I will be keeping an eye out for ways this one could be improved. Changes generally take the form of opportunities for extra credit, so nobody gets hurt and some people may be helped. If I make a change to the course and it seems unfair to you, let me know and I will try to correct it. If you still think it is unfair, you can appeal to the department chair or the dean. Also, you are welcome to suggest ways you think the class could be improved.

Digital Recording by me: I may digitally record the audio of my lectures some days. This is to help me improve my teaching materials.

Digital Recording by you: Almost everyone has a smart phone these days. I assume students will freely record what goes on in class, and take pictures of what is on the white board, to aid in their studies. I simply ask that you not embarrass anyone.

1.7.1   Special Treatment

There are many good reasons why students request special treatment. These include, for example, illness, field trips, performances, athletic events, and special needs. Instead of dealing with these as they arise, based on my past experience, I have adopted general policies that are intended to accommodate all but the most difficult cases, and thereby avoid the need for special treatment.

The exams are virtually identical, and given many times. If you miss one because of travels or illness or any other reason, just make it up by doing

well on the next test. If you must miss more than two exams, you may have a special case and should see me.

1.7.2 Reasonable Accommodation

This section covers special needs, including qualified special needs, as well as all other requests for special treatment.

I have carefully designed each of my classes to provide reasonable accommodation to those with special needs. Beyond that, further accommodation is usually considered to be unreasonable and only happens in extreme cases. Please see the paragraph on "Accommodating Special Needs" below for more information.

Ample Time: Specifically, I allow ample time on tests so that a well-prepared student can typically finish each test in half of the time allowed. This gives everyone essentially double the amount of time that should normally be needed.

Exam Retakes: There are no make-up exams (except in VERY special circumstances). I give the final exam a number of times and some students are able to complete it before the last time.

Deadlines: Most assignments are due the next class day after they are introduced, but I normally allow late work up to a month late (but not later than 23:59 on Fri, Feb 19). See the study guide for specific deadlines.

Even though I truly believe that these methods provide reasonable accommodation for almost everyone in almost every case, you might have a highly unusual situation for which I can and should do even more. You are welcome to see me about your situation.

1.8 Learning Outcomes

Outcomes (sometimes called objectives) are stated at several levels: Institutional (ILO), Program (PLO), and Course (CLO). In this section we set forward these outcomes and tell how they are aligned with one another.

1.8.1 ILOs: Institutional Outcomes

ILO: Institutional Learning Outcomes (ILOs) summarize the goals and outcomes for all graduates of BYUH.

Brigham Young University Institutional Learning Objectives (ILOs) Revised 24 February 2014

Graduates of Brigham Young University–Hawai'i will:

Knowledge: Have a breadth of knowledge typically gained through general education and religious educations, and will have a depth of knowledge in their particular discipline.

Inquiry: Demonstrate information literacy and critical thinking to understand, use, and evaluate evidence and sources.

Analysis: Use critical thinking to analyze arguments, solve problems, and reason quantitatively.

Communication: Communicate effectively in both written and oral form, with integrity, good logic, and appropriate evidence.

Integrity: Integrate spiritual and secular learning and behave ethically.

Stewardship: Use knowledge, reasoning, and research to take responsibility for and make wise decisions about the use of resources.

Service: Use knowledge, reasoning, and research to solve problems and serve others.

1.8.2 PLOs: Program Outcomes

PLO: Program Learning Outcomes (PLOs) summarize the goals and outcomes for graduates in programs for which this course is a requirement or an elective. These support the ILOs, but are more specific.

At the end of this section, we include the relevant page from the CIS Program Outcomes Matrix, dated April 2011.

The following outcomes are pursued at the "Introduced" level, and apply to one or more of the majors that use this course.

(a) An ability to apply knowledge of computing and mathematics appropriate to the discipline.

(b) An ability to analyze a problem, and identify and define the computing

requirements appropriate to its solution.

(i) An ability to use current techniques, skills, and tools necessary for computing practice.

(CS j) An ability to apply mathematical foundations, algorithmic principles, and computer science theory in the modeling and design of computer-based systems in a way that demonstrates comprehension of the trade-offs involved in design choices.

(CS k) An ability to apply design and development principles in the construction of software systems of varying complexity.

CIS Department Outcomes Matrix, April 2011

Program Outcomes

(a) An ability to apply knowledge of computing and mathematics appropriate to the discipline.
(b) An ability to analyze a problem, and identify and define the computing requirements appropriate to its solution.
(c) An ability to design, implement, and evaluate a computer-based system, process, component, or program to meet desired needs.
(d) An ability to function effectively on teams to accomplish a common goal.
(e) An understanding of professional, ethical, legal, security and social issues and responsibilities.
(f) An ability to communicate effectively with a range of audiences.
(g) An ability to analyze the local and global impact of computing on individuals, organizations, and society.
(h) Recognition of the need for and an ability to engage in continuing professional development.
(i) An ability to use current techniques, skills, and tools necessary for computing practice.

CS Only

(j) An ability to apply mathematical foundations, algorithmic principles, and computer science theory in the modeling and design of computer-based systems in a way that demonstrates comprehension of the tradeoffs involved in design choices. [CS]
(k) An ability to apply design and development principles in the construction of software systems of varying complexity. [CS]

IS Only

(j) An understanding of processes that support the delivery and management of information systems within a specific application environment. [IS]

IT Only

(j) An ability to use and apply current technical concepts and practices in the core information technologies. [IT]
(k) An ability to identify and analyze user needs and take them into account in the selection, creation, evaluation and administration of computer-based systems. [IT]
(l) An ability to effectively integrate IT-based solutions into the user environment. [IT]
(m) An understanding of best practices and standards and their application. [IT]
(n) An ability to assist in the creation of an effective project plan. [IT]

R = Required in that program | **CSS** = CS B.S. | **CIS** = CIS B.S. | **IS** = IS B.S. | **IT** = IT B.S.
# = choose at least 9 cr hrs | O = optional as a substitute | L = Introduced, M = Practiced with feedback, H = Demonstrated at the Mastery level

Course	Description	CSS	CIS	IS	IT	a	b	c	d	e	f	g	h	i	CSj	CSk	ISj	ITj	ITk	ITl	ITm	ITn
CIS 100	Fundamentals of Info. Systems & Tech.			R	R	L	L	L	L	L	L	L	L	L			L	L	L			
CIS 101	Beginning Programming	R	R	R	R	L	L								L	L	L					
CIS 202	Object-Oriented Programming	R	R	R	R	M	M	M		L			L	M	L	L		M	L		L	L
CIS 205	Discrete Mathematics I	R	R	R	R	M	M	L	L				M	M	M							
CIS 206	Discrete Mathematics II	R	R	R		M	M	L	L				M	M	M							
CIS 305	Systems Engineering I	R	R	R	R	M	M	M	M	L	L	M	L	M	L	L	M	L	H	L	H	M
CIS 401	Web Application Development	R		R	R	M	L	L						M			L	M	L	L		
CIS 405	Systems Engineering II	R	R	R	R	M	M	M	M	L	M	M	M	M	M	M	M	M	H	M	H	M
CIS 470	Ethics in Computer & Info. Sciences	R	R	R	R		L	L	M	H	H	H	H									
CS 203	Object-Oriented Programming II	R				M	M	M					M	M	M							
CS 210	Computer Organization	R			R	H	M	L					M	L				M				
CS 301	Algorithms & Complexity	R				L	M	L	L		M		L	M	H							
CS 320	Computational Theory	R				H	M			L		L	M		H	M						
CS 415	Operating Systems Design	R				H	H	H		M	M	M	H	H	H	H					M	
CS 420	Programming Languages	R				H	H	H		M	M	M	H	H	H	H						
CS 490R	Adv Topics in Computer Science (6 CR)	R				H	H	H					H		H	H						
IS 330	Management Information Systems					L	L		M	L	M	L	L	L			L					
IS 350	Database Management Systems	R	R	R	R	M	L	M	M	L	L	L	L	M	M	L	L	H	L			
IS 430	ITS – Enterprise Resource Planning			R			L	M	M	M	M	M	M	H			H		L		M	
IS 435	Advanced Concepts ERP Systems					H	H		H	L	M	M	M	H			H			L	H	
IS 485	Project Management & Practice				R	M	H	M	H	M	H	M	H	M	M	H	H	M				H
IT 220	Linux Essentials				R	M							M					M				
IT 224	Computer Hardware & Systems Software			R	R	M	H	L	M	L	M	L	L	L				M	M	L	L	
IT 240	Fund. Of Web Design & Technology			R	R	L	L	L		M	H	M		M		L	L	M	M	M	L	
IT 280	Data Comm. Systems & Networks	R	R	R	R	M	M	M		M	M		L	M				M	L	L		
IT 420	Linux System Administration				R	H	H	M						H				M	M	M		
IT 426	Computer Network Services				R	H	H	M	L	L	L	L	L	M				H	M	M	M	L
IT 440	Foundations of HCI				R	M	H	H	M	H	M	H	M	M			H	M	H	H	H	M
IT 480	Computer Network Design				R	H	H	H					L, M	H				M	M	M		M
IT 481	Information Assurance & Security				R		L	L		L	L	L	L	M				M	M	L	M	L
Math 112	Calculus I	O		R	#																	
Math 113	Calculus II	O			#																	
Math 119	Applied Calculus	R	O	O	#																	
Math 214	Mulitvariable Calculus				#																	

1.8.3 CLOs: Course Outcomes

CLO: Course Learning Outcomes (CLOs, also called Student Learning Outcomes, or SLOs) summarize the goals and outcomes for students who successfully complete this course. These support the PLOs, but are more specific.

Course Goals and Student Learning Outcomes are as follows:

By the conclusion of this course, students will demonstrate the ability to write clear and correct programs that utilize the following techniques.

- sequences of simple steps

- simple variables (scalars)

- decisions (if, else, elsif)

- looping (while, for, foreach)

- array and list variables

- subroutines

Students will demonstrate these major skills by creating, in timed and supervised situations, short programs that perform specific tasks.

In teaching the major skills, I also teach the following:

- dynamic web page creation

- dynamic response to web page inputs

1.9 General University Policies

1.9.1 Academic Integrity

Brigham Young University–Hawai'i is committed to upholding a policy of academic integrity among administrators, faculty, staff, and students. Basically we are talking about cheating here. We tell you what is acceptable and what is not.

Plagiarism

We learn by watching others and then doing something similar.

Plagiarism: Sometimes it is said that plagiarism is copying from one person, and research is "copying" from lots of people.

When you are having trouble with an assignment, I encourage you to look at not just one, but many examples of work done by others. Study the examples. See what you can learn from them. Do not automatically trust that they are right. They may be wrong.

http://en.wikipedia.org/wiki/Plagiarism has a wonderful article on plagiarism. Read it if you are not familiar with the term. Essentially, plagiarism is when you present the intellectual work of other people as though it were your own. This may happen by cut-and-paste from a website, or by group work on homework. In some cases, plagiarism may also create a violation of copyright law. If you borrow wording from someone else, identify the source.

Intentional plagiarism is a form of intellectual theft that violates widely recognized principles of academic integrity as well as the Honor Code. Such plagiarism may subject the student to appropriate disciplinary action administered through the university Honor Code Office, in addition to academic sanctions that may be applied by an instructor.

Inadvertent plagiarism, whereas not in violation of the Honor Code, is nevertheless a form of intellectual carelessness that is unacceptable in the academic community. Plagiarism of any kind is completely contrary to the established practices of higher education, where all members of the university are expected to acknowledge the original intellectual work of others that is included in one's own work.

Faculty are responsible to establish and communicate to students their expectations of behavior with respect to academic honesty and student conduct in the course. Observations and reports of academic dishonesty shall be investigated by the instructor, who will determine and take appropriate action, and report to the Honor Code Office the final disposition of any incident of academic dishonesty by completing an Academic Dishonesty Student Violation Report. If the incident of academic dishonesty involves the violation of a public law, e.g., breaking and entering into an office or stealing an examination, the act should also be reported to University Police. If an affected student disagrees with the determination or action and is unable

to resolve the matter to the mutual satisfaction of the student and the instructor, the student may have the matter reviewed through the university's grievance process.

Specific Rules For CIS 101

Exams: On exams you are required to work from personal memory, using only the resources that are normally present on your computer. This means the exams are closed book and closed notes. However, you are nearly always allowed (and encouraged!) to test your programs by actually running them on the computer where you are sitting. Students caught cheating on an exam may receive a grade of F for the semester, no matter how many points they may have earned, and they will be reported to the Honor Code office.

Exam Question 8 (p.164)**:** Can I bring blank paper into an exam?
Required Answer: yes

Exam Question 9 (p.164)**:** Can I bring notes into an exam?
Required Answer: no

Exam Question 10 (p.164)**:** During exams can I make notes?
Required Answer: yes

Exam Question 11 (p.164)**:** During exams can I use email?
Required Answer: no

Exam Question 12 (p.165)**:** During exams can I use my cell phone?
Required Answer: no

Exam Question 13 (p.165)**:** During exams can I use my laptop computer?
Required Answer: no

Exam Question 14 (p.165)**:** During exams can I use Google?
Required Answer: no

Exam Question 15 (p.165)**:** During exams can I use a language dictionary?
Required Answer: no

Exam Question 16 (p.165)**:** During exams can I run my programs to test them?
Required Answer: yes

3MQ: On three-minute quizzes you are required to work from personal memory, using only the resources that are normally present on your com-

puter. This means the quizzes are closed book and closed notes.

Exam Question 17 (p.165)**:** Can I bring notes into a 3MQ?
Required Answer: no

Exam Question 18 (p.165)**:** During 3MQs can I use the textbook?
Required Answer: no

Exam Question 19 (p.165)**:** During 3MQs can I use Google?
Required Answer: no

Other Than Exams:

Help each other, but do your own work.

These two goals are in conflict with each other. To resolve this conflict, I draw the line at copying.

Except during exams, you are strongly encouraged to work with your fellow students. We want everyone to get full credit on every assignment. Please help each other learn. That is the goal.

Does Program include HTML or just Perl?

Just Perl. This is a programming class and when we talk about programming we mean Perl, not HTML.

To make it fun and interesting and useful, we also do some HTML (webpage markup), but that is all extra. You can freely copy HTML from each other.

Exam Question 20 (p.165)**:** Is it okay to freely copy HTML from others?
Required Answer: yes

Exam Question 21 (p.165)**:** Is it okay to freely copy Perl from others?
Required Answer: no

If You are Looking at Someone's Program:

You **may**: look, read, make mental notes, ask questions, point out possible errors, and try to understand. (Although you can point out errors, you cannot tell how to fix them.)

You **must not**: fix someone else's program, take written notes, take pictures, take a copy of all or part of their program.

You are free to try to understand and memorize parts of their program, and then go back to your seat and type it in from memory.

Exam Question 22 (p.165)**:** On homework, can I look at someone else's

program?

Required Answer: yes

Exam Question 23 (p.165)**:** On homework, can I ask someone else questions about their program?

Required Answer: yes

Exam Question 24 (p.165)**:** On homework, can I point out possible errors in someone else's program?

Required Answer: yes

Exam Question 25 (p.165)**:** On homework, can I explain to someone else what I think their program is doing?

Required Answer: yes

Exam Question 26 (p.165)**:** On homework, can I explain what I think my program is doing?

Required Answer: yes

Exam Question 27 (p.165)**:** On homework, can I fix someone else's program?

Required Answer: no

Exam Question 28 (p.165)**:** On homework, can I tell others what to type to fix their programs?

Required Answer: no

Exam Question 29 (p.165)**:** On homework, can I accept a copy of someone else's program?

Required Answer: no

Exam Question 30 (p.165)**:** On homework, can I make written notes while looking at someone else's program?

Required Answer: no

Exam Question 31 (p.165)**:** On homework, can I take a picture or screen shot of someone else's program?

Required Answer: no

If Someone is Looking at Your Program:

You **may**: explain your approach, ask for help.

Exam Question 32 (p.165)**:** On homework, can I explain my approach to someone else?

Required Answer: yes

You **must not**: let them fix your program, give them a copy of all or part of your program.

Exam Question 33 (p.166)**:** On homework, can I give someone a copy of my program?

Required Answer: no

Exam Question 34 (p.166)**:** On homework, can I let someone make changes to my program?

Required Answer: no

This includes working with tutors or students who took the class in previous semesters.

Applicable Actions

http://honorcode.byuh.edu/ details the university honor code. In the section entitled "Applicable Actions" the following are listed.

Examples of possible actions include but are not limited to the following, for instructors, programs, departments, and colleges:
• Reprimanding the student orally or in writing.
• Requiring work affected by the academic dishonesty to be redone.
• Administering a lower or failing grade on the affected assignment, test, or course.
• Removing the student from the course.
• Recommending probation, suspension, or dismissal.

Exam Question 35 (p.166)**:** Could cheating result in a zero on the assignment?

Required Answer: yes

Exam Question 36 (p.166)**:** Could cheating result in an F in the course?
Required Answer: yes

Depending on the specifics of the offense, any of these responses may be possible.

Cheating on exams is the most common form of dishonesty that I normally encounter. Normally this happens when students bring in notes that include answers to past exam questions. I approve the studying of past exams, and bringing in of "memories" based on study, but not the access to written notes, including notes retrieved from other exams or stored on cell phones or other devices. Any such activity can result in failure of the entire course.

Most of the cheating that is discovered is because the exact same mistakes were made by two or more people.

1.9.2 Unlawful Discrimination

Brigham Young University–Hawai'i is committed to a policy of nondiscrimination on the basis of race, color, sex (including pregnancy), religion, national origin, ancestry, age, disability, genetic information, or veteran status in admissions, employment, or in any of its educational programs or activities.

1.9.3 Title IX and Sexual Misconduct

Brigham Young University–Hawai'i is committed to promoting and maintaining a safe and respectful environment for the campus community. In the USA, Title IX (Title 9) of the Education Amendments of 1972 prohibits all sexual misconduct against any participant in an educational program or activity.

Sexual Misconduct includes:

Sexual Harassment, which is unwelcome speech or conduct of a sexual nature. It includes unwelcome sexual advances, requests for sexual advances, requests for sexual favors, and other verbal, non-verbal, or physical conduct that is not requested or invited.

Stalking, which is repeatedly following, monitoring, harassing, threatening or intimidating another by phone, mail, electronic communication or social media without legitimate purpose.

Domestic and Dating Violence, which is a pattern of abusive behavior in any relationship that is used by one partner to control another partner. This includes behaviors that intimidate, manipulate, humiliate, isolate, frighten, terrorize, coerce, threaten, blame, hurt, injure, or wound.

Sexual Violence / Assault, which is actual or attempted sexual contact with another person without that person's consent.

Consent cannot be obtained when someone is a minor, under the influence of drugs or alcohol, or has certain disabilities. In the absence of an outward demonstration, consent does not exist. If at any time it is reasonably apparent that either party is hesitant, confused, or uncertain, both parties should

stop.

The following individual has been designated to handle reports of sexual harassment and other inquiries regarding BYUH compliance with Title IX:

```
Debbie Hippolite-Wright, PhD
Title IX Coordinator
Vice President, Student Development & Life
Lorenzo Snow Administration Building
55-220 Kulanui Street
Laie, Hawaii 96762
Office Phone: 808-675-4819
E-Mail: titleix@byuh.edu
Sexual Harassment Hotline: 808-780-8875
```

BYUH's Office of Honor upholds a standard which states that parties can only engage in sexual activity freely within the legal bonds of marriage between a man and a woman. Consensual sexual activity outside the bonds of marriage is against the Honor Code and may result in probation, suspension, or dismissal from the University.

1.9.4   Services for Students with Special Needs

Brigham Young University–Hawai'i is committed to providing a working and learning atmosphere, which reasonably accommodates qualified persons with disabilities. If you have a disability and need accommodations, you may wish to self-identify by contacting:

```
Services for Students with Special Needs
McKay 181
Phone: 808-675-3518 or 808-675-3999
Email address: aunal@byuh.edu
```

The Coordinator for Students with Special Needs is Leilani A'una.

Students with disabilities who are registered with the Special Needs Services should schedule an appointment with the instructor to discuss accommodations. If the student does not initiate this meeting, it is assumed no accommodations or modifications will be necessary to meet the requirements of

this course. After registering with Services for Students with Special Needs, and with permission of the student, Letters of Accommodation will be sent to instructors.

Exam Question 37 (p.166)**:** Can students with special needs get special accommodations?

Required Answer: yes

1.10 Syllabus Summary

Brigham Young University–Hawai'i has adopted certain requirements relating to the information that must be provided in syllabi. This section lists those requirements and for each item either provides the information directly or gives a link to where it is provided above.

Course Information: See section 1.2.1 (page 5).

- **Title:** Beginning Programming
- **Number:** CIS 101
- **Semester/Year:** Winter 2015-16
- **Credits:** 3
- **Prerequisites:** none
- **Location:** GCB 111
- **Meeting Time:** MWF 13:20 to 14:20

Faculty Information: See section 1.2.2 (page 6).

- **Name:** Don Colton
- **Office Location:** GCB 128
- **Office Hours:** (In GCB 111) MWF 14:30-15:00, TuTh 15:30-16:00.
- **Telephone:** 808-675-3478
- **Email:** doncolton2@gmail.com

Course Readings/Materials: See section 1.2.3 (page 6) for a list of textbooks, supplementary readings, and supplies required.

Course Description: See section 1.2.1 (page 5).

Expected Proficiencies:
See section 1.1.2 (page 5) for the proficiencies you should have before undertaking the course.

Course Goals and Student Learning Outcomes, including Alignment to Program (PLOs) and Institutional (ILOs) Learning Outcomes, and extent of coverage.

See section 1.8 (page 21) for learning outcomes, showing the content of the course and how it fits into the broader curriculum. A listing of the departmental learning outcomes is provided together with the ratings taken from department's matrix assessment document representing the degree to which the course addresses each outcome.

Instructional Methods: See section 1.6 (page 17).

Learning Management System:
https://dcquiz.byuh.edu/ is the learning management system for my courses.

Framework for Student Learning:
See section 1.6.1 (page 18) for a discussion of the student learning framework and how I use it.

Course Calendar: See section 1.3 (page 7) for the calendar in general.

Here are some items of particular interest:
- **First Day of Instruction:** Mon, Nov 09
- **Last Day to Drop:** Fri, Dec 11
- **Last Day to Withdraw:** Wed, Jan 20
- **Last Day of Instruction:** Mon, Feb 22
- **Final Exam:** Wed, Feb 24, 13:00 to 15:50
- **Final Exam Location:** GCB 111

Course Policies: See section 1.7 (page 20).
- **Attendance:** See section 1.4.2 (page 10).
- **Tardiness:** See section 1.4.2 (page 10).
- **Class Participation:** See section 1.6.1 (page 18).
- **Make-Up Exams:** See section 1.7.2 (page 21).
- **Plagiarism:** See section 1.9.1 (page 26).
- **Academic Integrity:** See section 1.9.1 (page 25).

Evaluation (Grading): See section 1.4 (page 9).

Academic Honesty: See section 1.9.1 (page 25).

Unlawful Discrimination: See section 1.9.2 (page 31).

Title IX and Sexual Misconduct: See section 1.9.3 (page 31).

Grievances: The university grievance policy states that the policies listed on the syllabus can act as a contract and will be considered if a student has a complaint about the instructor or the course.

Services for Students with Special Needs: See section 1.9.4 (page 32).

1.11 Syllabus Additions and Corrections

None yet. To be added as necessary.

Chapter 2

Calendar and Activities

The syllabus lists a calendar of planned activities. This chapter repeats, amplifies, and corrects that list. For future class periods it shows my best guess about what will happen. For past class periods it shows my best recollection of what actually happened. It will be updated from time to time as the semester continues.

Contents

This chapter lists the activities that are planned or have been assigned. As new activities are assigned, this chapter will be updated.

Reminder: You **must not** give a copy of your program to another student. You are cheating them out of the learning experience that they should be having. Please limit your assistance to these things:

(a) You **may** let them look at your program, but not make notes. If they can memorize it, I guess that is okay, but if they are writing things down, or if they take a picture of it, they have committed an honor code violation. (It is okay to take pictures of things I write on the white board and to share them.)

(b) You **may** look at their program and give them advice. Tell them where you think their errors are. Do not tell them what to type. Say things like, "you might need to change that," or "are those supposed to be the same?" or "you might need something at the end of that line." Make them do some thinking.

2.1   Mon, Nov 09

- Exams: none
- First Day of Instruction
- Assignment: send email to `doncolton2@gmail.com`
- Review some webpages of interest to this course.
  - ○ <http://doncolton.com/> is my off-campus homepage
  - ○ <http://byuh.doncolton.com/> is my on-campus homepage
  - ○ <http://byuh.doncolton.com/cis101/> is this course homepage
  - ○ <http://byuh.doncolton.com/cis101/2161/syl.pdf> is the syllabus
  - ○ <http://byuh.doncolton.com/cis101/2161/sguide.pdf> is this study guide
- Daily Update d01
  - ○ <http://dcquiz.byuh.edu/> is where you take the daily update.
- o1H, First Webpage

2.1.1   Task em: Email

- Introduced: Mon, Nov 09
- 2pt Due Date: Wed, Nov 11, 23:59.
- 1pt Due Date: Wed, Nov 18, 23:59.
- Grading Label: **em**

We normally do this in class the first day. It is worth two attendance points.

Your task is fairly simple: send me an email. If you do it correctly, you should receive and immediate response.

To: Send it to `doncolton2@gmail.com`

Subject: Start the subject line with cis101 and continue with your rollbook-name (which is your lastname, comma, your firstname).

Good example: cis101 Colton, Don

Bad example: cis101ColtonDon

Bad example: cis101_Colton_Don

Body: The body of your email should cover these things:

Name: What name would you like me to use when I call upon you in class? If your lastname is unusual, specify that too.

Prayers: Are you willing to say the opening prayer in class if I call on you?

Most people say yes. Some people say no. It is okay either way. Let me know.

Immediate Response: After you send your email, you should receive an immediate response from my email system. The response will say something like this:

This is an instant reply to let you know that I received your email relating to CIS 101 (Beginning Programming), and it has been placed into my pending file for review and/or grading as soon as I can get to it. You can expect another reply from me when I have looked at your email. Thanks! Bro Colton

The exact wording may vary, but that's the general idea.

If you do not receive an immediate response, here are some problems you can look for:

(a) Did you send it to the correct email address?

(b) Did you do the subject line properly? Make sure the spacing is right.

(c) Are you online? Sometimes people send from a mobile device like a cell phone or a tablet which may not be constantly connected with the Internet. The device may hold your mail until it can connect, and then send it.

(d) Is the network having problems? Sometimes (rarely) the network is down. Can you visit webpages? Can you send and receive email besides this one?

2.1.2   Task o1H: First Webpage

- Introduced: Mon, Nov 09
- 11pt Due Date: Mon, Nov 09, 23:59.
- 10pt Due Date: Wed, Nov 11, 23:59.
- 9pt Due Date: Fri, Nov 13, 23:59.
- 8pt Due Date: Mon, Nov 16, 23:59.
- 7pt Due Date: Mon, Nov 30, 23:59.
- Grading Label: **o1H**

Online: This is an online task, but you do not need to send me the code that you write. It just needs to work when I test it online. The general rules in section C.1 (page 135) apply. To request a regrade, see B.4 (page 133).

We normally do this in class the first day. If you miss class that day, please

work with one of the tutors to complete the assignment.

Log into cPanel. Go into File Manager.

Go into your `public_html` directory (folder).

You **must** create a directory (folder) with `cis101.2161` as its name. The cis101 part stands for this class. The 2161 part stands for this semester.

In that directory you **must** create a file with `index.html` as its name. Be careful that you do not create `INDEX.HTML` or anything else that is not `index.html` exactly.

Construct a webpage.

Your name **must** appear (a) in the title and (b) somewhere on the page, either in the h1 at the top, or in a line at the bottom.

You **must not** have any copied content that is not properly updated.

You **may** use the following lines as an example. You **may** use cut-and-paste if you want, but you **should** type them in so you start thinking about what they mean. If you have more HTML skills than this, feel free to do a more impressive job.

```
<!DOCTYPE html><head lang=en><meta charset=utf-8>
<title>Don's CIS 101 Homepage</title>
<style></style>
</head><body>
<h1>This is Don's CIS 101 Homepage</h1>
<img src=mypic.jpg alt="my picture" width=500>
</body>
```

You **must** change the name "Don's" to match your own name, of course.

You **must** upload a picture "of" yourself. It can simply be something that represents you, or it can be an actual picture of yourself. You **may** name it whatever you want, so long as the webpage works correctly.

By upload, I mean that you **must** actually upload a picture and not simply link to a picture that already exists on another website.

Verify that you can reach your page through clicking on your **o1H** link on the CIS 101 Student Projects webpage, which is:

http://dc.is2.byuh.edu/cis101.2161/

2.2   Wed, Nov 11

- Exams: none
- Finish reading Unit 1 (Output) before class
- Daily Update d02
- Review the grade books
- Policy Test practice
- g11 Hello

2.2.1   Task g11: Hello

- Introduced: Wed, Nov 11
- 11pt Due Date: Wed, Nov 11, 23:59.
- 10pt Due Date: Fri, Nov 13, 23:59.
- 9pt Due Date: Mon, Nov 16, 23:59.
- 8pt Due Date: Wed, Nov 18, 23:59.
- 7pt Due Date: Wed, Dec 02, 23:59.
- Grading Label: **g11**

GradeBot: This is a GradeBot task. If your program runs correctly and has proper indenting, GradeBot will send it to me for you. The general rules and explanations in section C.3 (page 139) apply, including the required comment line.

`# cis101 g11` *roll-book-name* is the required comment line. There may be customizations that are different for each student based on this comment line. Your wordings may differ from study guide examples.

GradeBot is at http://gradebot.tk/ and http://gbot.dc.is2.byuh.edu/.

Task: Say "Hello"

Objective: Become familiar with GradeBot.

2.3   Fri, Nov 13

- Exams: policy test
- Have previously read textbook Unit 1.
- Be working on reading textbook Unit 2.
- Daily Update d03
- Review Grade Books
- First 3MQ (Three-Minute Quiz) is Monday
- o1R Random Number
- 13:50-13:56 6m Policy Test

2.3.1   Task o1R: Random Number

- Introduced: Fri, Nov 13
- 11pt Due Date: Fri, Nov 13, 23:59.
- 10pt Due Date: Mon, Nov 16, 23:59.
- 9pt Due Date: Wed, Nov 18, 23:59.
- 8pt Due Date: Fri, Nov 20, 23:59.
- 7pt Due Date: Fri, Dec 04, 23:59.
- Grading Label: **o1R**

We will be doing an online program. We will do it in three steps. First we will make a command-line program and get it working. Then we will modify it to run on the web.

Online: This is an online task, but you do not need to send me the code that you write. It just needs to work when I test it online. The general rules in section C.1 (page 135) apply. To request a regrade, see B.4 (page 133).

There is a chapter in the textbook, probably about chapter 9, probably about page 68, that talks about random numbers.

Summary: Make a web program that displays a random number each time you load it.

Warning: I am going to give you some of the information that you need. You need to figure out what order things must happen. You are welcome to confer with your neighbors, but do not simply trust them and copy their work.

Step 1: Simple Command Line Version

You do not turn in this step. You turn in step 3. But step 1 is important for testing your program.

Write the following program (or one like it) and test it.

```
$random = rand(30);
print "Your number is $random.";
```

That will create and display a random number between zero and 30.

Run your program at the command line. Each time you run it, you should see a new random number.

Step 2: Command Line Version

You do not turn in this step. You turn in step 3. But step 2 is important for testing your program.

When your program is executed, it must print out the HTML code for a webpage.

You can use the following webpage as an example. Cut-and-paste might work but it is better to type it in so you start thinking about what it means. If you have more HTML skills than this, feel free to do a more impressive job.

```
<!DOCTYPE html><head lang=en><meta charset=utf-8>
<title>Don's Random</title>
<style></style>
</head><body>
<h1>Don's Random Number Generator</h1>
$random
</body>
```

Remember: Unless your name is Don, it should not say Don in it.

Remember: To print a quote mark, use a backslash in front of it.

You have to pick the right time for this number to be generated.

Run your program at the command line. Each time you run it, you should see the HTML version of a web page displayed.

Step 3: OnLine Version

File name: Log into cPanel. Go to your cis101.2161 folder. Create a subfolder with `random` as its name. In it create a file with `index.cgi` as its name.

Permissions: This `index.cgi` file will be a program. Because it is a program, you must set its permissions to **0755**. Normal webpages have their permissions set to **0644**.

The **0755** permission tells the is2 machine that this will be a program, and the perl line tells it that it is a perl program (as opposed to php or ruby or something else).

Add these two lines to the front of your program.

```
#! /usr/bin/perl --
print "content-type: text/html\n\n";
```

The rest of your program can be exactly like the command line version. Copy and paste and save it.

Requirement: Your program must generate a complete webpage, with DOCTYPE, head, charset, title, etc. It cannot simply print a random number. It must generate a complete webpage.

Run your program by viewing your webpage. Each time you do a reload on your webpage, it will cause your program to run again, and a new number will appear.

Verify that you can reach your page (and run your program) through clicking on your **o1R** link on the CIS 101 Student Projects webpage, which is:

http://dc.is2.byuh.edu/cis101.2161/

If you get a 404 File Not Found error, see section A.4.2 (page 126) for guidance.

If you get a 500 Internal Server Error, see section A.4.3 (page 126) for guidance.

2.4 Mon, Nov 16

- Exams: 3mq
- Have previously read textbook Units 1 and 2.
- Finish reading Unit 2 (Input) before class
- Daily Update d04
- Three-Minute Quiz q04
- g21 Hi Fred

2.4.1 Task g21: Hi Fred

- Introduced: Mon, Nov 16
- 11pt Due Date: Mon, Nov 16, 23:59.
- 10pt Due Date: Wed, Nov 18, 23:59.
- 9pt Due Date: Fri, Nov 20, 23:59.
- 8pt Due Date: Mon, Nov 23, 23:59.
- 7pt Due Date: Mon, Dec 07, 23:59.
- Grading Label: **g21**

GradeBot: This is a GradeBot task. If your program runs correctly and has proper indenting, GradeBot will send it to me for you. The general rules and explanations in section C.3 (page 139) apply, including the required comment line.

`# cis101 g21` *roll-book-name* is the required comment line. There may be customizations that are different for each student based on this comment line. Your wordings may differ from study guide examples.

GradeBot is at http://gradebot.tk/ and http://gbot.dc.is2.byuh.edu/.

There is a chapter in the textbook, probably about chapter 4, probably about page 35, that talks about accepting inputs.

Task: Ask for a name. Respond with "Hello, (name)!"

Sample Execution:

```
GradeBot would have engaged your program in this dialog:
note  GBot "# debug output lines are permitted"
  1:  "What is your name?"
 in>  "Fred"
  2:  "Wassup, Fred!"
 eof  (end of output)
```

2.5   Wed, Nov 18

- Exams: 3mq
- Have previously read textbook Units 1 and 2.
- Be working on reading textbook Unit 3.
- Daily Update d05
- Three-Minute Quiz q05
- o1S Slot Machine

2.5.1   Task o1S: Slot Machine

- Introduced: Wed, Nov 18
- 11pt Due Date: Wed, Nov 18, 23:59.
- 10pt Due Date: Fri, Nov 20, 23:59.
- 9pt Due Date: Mon, Nov 23, 23:59.
- 8pt Due Date: Wed, Nov 25, 23:59.
- 7pt Due Date: Wed, Dec 09, 23:59.
- Grading Label: **o1S**

Online Plus Code: This is an online task. For this one you **also** need to send me the code that you write, and it also needs to work when I test it online. The general rules in section C.1 (page 135) apply, including email subject line but not program comment line.

`cis101 o1S` *roll-book-name* is the required subject line. There is no required comment line because this is not a GradeBot task.

Objective: Learn about random numbers and some simple mathematics.

Summary: Make a web program that shows three images randomly chosen each time you load it. Think of it as a slot machine.

There is a chapter in the textbook, probably about chapter 9, probably about page 68, that talks about random numbers.

You might have to look at previous assignments, specifically the o1R assignment, for some how-to information that applies to this task.

When your program is executed, it will print out a webpage. You can have it print a webpage that looks something like this:

```
content-type: text/html

<!DOCTYPE html><head lang=en><meta charset=utf-8>
```

```
<title>Don's Slot Machine</title>
<style>
body { background-color: #35ffff; text-align: center; }
</style>
</head><body>
<h1>Don's Slot Machine</h1>
<p>We will pick three images.
<p>
<img src=1.jpg alt=1 title=1 width=100 height=100>
<img src=6.jpg alt=6 title=6 width=100 height=100>
<img src=4.jpg alt=4 title=4 width=100 height=100>
</body>
```

You must show three images chosen from a pool of ten possibilities.

Instead of printing an actual 1 and 6, etc., as shown, use variables, maybe $slot1, $slot2, etc.

`$slot1 = int ( rand(10) );`

That will create a random integer between 0 and 9.

The `rand(10)` part will generate a number between zero and ten, but it will never actually be ten. One tenth of the time it will be between 3.000 and 3.999 (more or less).

The `int` part converts a number like 3.14159 into an **int**eger like 3 by cutting off the fractional part of the number. It is like chomp, but for the fractional parts of numbers.

After int happens, one tenth of the time the result will be exactly 3.

Run your program by reloading your webpage. Each time you do a reload on your webpage, it will cause your program to run again, and a new set of images will appear.

You will need ten images. One good source is http://emojipedia.org/ where you can grab interesting images. Then rename them to be 0.png, 1.png, and so forth up to 9.png. Try to get square images.

Here are some ideas for images: (luck symbols) rainbow, four-leaved clover, crown, pot of gold, horse shoe, wishbone, crossed fingers; (fruit) cherry, lemon, apple, banana; (animals) duck, chicken, dog, cat; (chinese years) ram, snake, horse, tiger; (zodiac) aquarius, pisces, aries, scorpio; (geometry) circle, square, triangle, star; flags of nations; flags or symbols of sports

teams.

2.6   Fri, Nov 20

- Exams: 3mq, policy test
- Have previously read textbook Units 1 and 2.
- Be working on reading textbook Unit 3.
- Daily Update d06
- Three-Minute Quiz q06
- c2M Mad Lib
- 14:10-14:20 10m Policy Test

2.6.1   Task c2M: Mad Lib

- Introduced: Fri, Nov 20
- 11pt Due Date: Fri, Nov 20, 23:59.
- 10pt Due Date: Mon, Nov 23, 23:59.
- 9pt Due Date: Wed, Nov 25, 23:59.
- 8pt Due Date: Fri, Nov 27, 23:59.
- 7pt Due Date: Fri, Dec 11, 23:59.
- Grading Label: **c2M**

Command Line: This is a command-line task. The general rules in section C.2 (page 138) apply, including email subject line and program comment line.

`cis101 c2M` *roll-book-name* is the required subject line.

`# cis101 c2M` *roll-book-name* is the required comment line.

There is a chapter in the textbook, probably about chapter 4, probably about page 35, that talks about accepting inputs.

A "Mad Lib" is a fill-in-the-blank story. You start by asking for a list of words and then you insert them into the blanks of a story. The result is sometimes very funny.

Task: Write a mad lib. Prompt for inputs such as "name of a boy" or "activity that is free". Then compose a story that uses those inputs. Test your program. Then email it to me.

Requirement: You must invent the story yourself. You cannot get it from any other source.

Requirement: There must be at least three inputs and at most five.

Requirement: Each input must be used at least twice in the story.

Request: The story should be about two paragraphs long.

Request: Please make the story creative and interesting.

Common Problem:

You may want to put 's (or some other letters) after one of your inputs. This is tricky because those things can be seen as part of the variable name even though they are not. To solve this problem, put the variable name in curly braces, like this:

```
${x}'s
```

Or you can escape the apostrophe with a back-slash, like this:

```
$x\'s
```

2.7   Mon, Nov 23

- Exams: 3mq
- Have previously read textbook Units 1 through 3.
- Finish reading Unit 3 (Math) before class
- Daily Update d07
- Three-Minute Quiz q07
- g31 Before After

2.7.1   Task g31: Before After

- Introduced: Mon, Nov 23
- 11pt Due Date: Mon, Nov 23, 23:59.
- 10pt Due Date: Wed, Nov 25, 23:59.
- 9pt Due Date: Fri, Nov 27, 23:59.
- 8pt Due Date: Mon, Nov 30, 23:59.
- 7pt Due Date: Mon, Dec 14, 23:59.
- Grading Label: **g31**

GradeBot: This is a GradeBot task. If your program runs correctly and has proper indenting, GradeBot will send it to me for you. The general rules and explanations in section C.3 (page 139) apply, including the required comment line.

`# cis101 g31` *roll-book-name* is the required comment line. There may be customizations that are different for each student based on this comment line. Your wordings may differ from study guide examples.

GradeBot is at http://gradebot.tk/ and http://gbot.dc.is2.byuh.edu/.

There is a chapter in the textbook, probably about chapter 11, probably about page 80, that talks about calculation.

Task: Read in a number. It will be a natural (whole) number. Tell what numbers are immediately before and after it.

Sample Execution:

```
GradeBot would have engaged your program in this dialog:
note  GBot "# debug output lines are permitted"
  1:  "Please enter a number: " (no \n)
 in>   ....................."5"
  2:  "The number before 5 is 4."
```

```
 3:   "The number after 5 is 6."
eof   (end of output)
```

The good news is that this is incredibly simple mathematics. The bad news is that this is still mathematics.

2.8 Wed, Nov 25

- Exams: 3mq
- Have previously read textbook Units 1 through 3.
- Be working on reading textbook Unit 4.
- Daily Update d08
- Three-Minute Quiz q08
- g35 Age
- **Exam v0 prac**

2.8.1 Task g35: Age

- Introduced: Wed, Nov 25
- 11pt Due Date: Wed, Nov 25, 23:59.
- 10pt Due Date: Fri, Nov 27, 23:59.
- 9pt Due Date: Mon, Nov 30, 23:59.
- 8pt Due Date: Wed, Dec 02, 23:59.
- 7pt Due Date: Wed, Dec 16, 23:59.
- Grading Label: **g35**

GradeBot: This is a GradeBot task. If your program runs correctly and has proper indenting, GradeBot will send it to me for you. The general rules and explanations in section C.3 (page 139) apply, including the required comment line.

`# cis101 g35` *roll-book-name* is the required comment line. There may be customizations that are different for each student based on this comment line. Your wordings may differ from study guide examples.

GradeBot is at http://gradebot.tk/ and http://gbot.dc.is2.byuh.edu/.

You will be asked for an age in years. You will convert it into months, days, hours, minutes, and seconds.

Besides getting the right answer, to qualify for the 11th point you must do it in a way that makes it easy to see that your program is correct. Use small and simple numbers to arrive at your results.

2.9 Fri, Nov 27

- No class today in honor of Thanksgiving Day

2.10   Mon, Nov 30

- Exams: 3mq, policy
- Have previously read textbook Units 1 through 3.
- Be working on reading textbook Unit 4.
- Daily Update d09
- Three-Minute Quiz q09
- o2M Online Mad Lib, olin
- 14:10-14:20 10m Policy Test

2.10.1   Task o2M: Mad Lib

- Introduced: Mon, Nov 30
- 11pt Due Date: Mon, Nov 30, 23:59.
- 10pt Due Date: Wed, Dec 02, 23:59.
- 9pt Due Date: Fri, Dec 04, 23:59.
- 8pt Due Date: Mon, Dec 07, 23:59.
- 7pt Due Date: Mon, Jan 04, 23:59.
- Grading Label: **o2M**

Online Plus Code: This is an online task. For this one you **also** need to send me the code that you write, and it also needs to work when I test it online. The general rules in section C.1 (page 135) apply, including email subject line but not program comment line.

`cis101 o2M` *roll-book-name* is the required subject line. There is no required comment line because this is not a GradeBot task.

There is a chapter in the textbook, probably about chapter 10, probably about page 72, that talks about accepting online input.

Task: Make an online Mad Lib story generator.

This is similar to the c2M Command-Line Mad Lib project we did earlier, but the inputs are online.

As for all online programs, your program must be properly linked to the student projects page, and the displayed webpage (not just the title) must show your name.

You can use the same story you used for the command-line MadLib task, or you can make up a new story.

Online Inputs:

Your webpage must include a form. The browser collects inputs from the user and sends them to your program. The markup for a form looks like this:

```
<form method=post action=.>
```

Then it must have an input field for each of the inputs you are using. The markup for an input field looks like this:

```
<input name=abc size=20 value="xyz" autofocus>
```

The name=abc part of the input field specifies its field name. It is like a variable name. For style reasons, you must pick a name that describes the data you hope to find in that field.

The size tells how many characters should fit in the field. It should be big enough to hold the largest data you reasonably expect to find in that field.

The value tells what characters, if any, should appear in the field initially. The quote marks are needed if the value has any spaces or certain other special characters. The user will be able to change the value.

Autofocus: The autofocus attribute is an optional part of the input specification, but we require your first input field to have autofocus. It tells the browser where to put the cursor initially. It works on fields where the user can type something in. Many of the online lab assignments require you to use autofocus.

One of your inputs will be a submit button. They look like this:

```
<input type=submit name=xyz value="Tell me a story">
```

The name part is pretty much useless when you only have one submit button.

The value part appears on the face of the submit button.

If you have more than one submit button, the name and value will tell you which one was clicked.

After all the inputs, you should end your form using `</form>`.

Recommended (not Required):

Put your program in this order: (1) /usr/bin/perl line, (2) olin subroutine, (3) all of your olin subroutine calls, (4) any decisions or calculations, (5) exactly one very big print statement that prints everything needed, from content-type through the end.

Requirements:

(a) Story: Use a web program to tell a story that involves user-provided inputs. Properly handle multi-word inputs.

(b) 3 to 5 Inputs: Your program must display and accept input from at least three and at most five input fields. Each must have a helpful and suitable description. Each value must be used at least twice in your story.

(c) Autofocus: Your first input field must have autofocus.

(d) Submit: Your webpage must have a submit button. When the submit button is pressed, your program should run and the screen should be updated.

(e) Defaults: When your program runs, if user inputs are provided they must be used. Otherwise (like the very first time your program runs), suitable default values must be used. Each input must have a reasonable (non-blank) default value.

(f) Sticky: When the screen is updated, the values that were last used shown must be shown again in the input fields. (They must be "sticky.") Make sure it works for multi-word inputs. If there were no last-used values (when you first start) then the default values should be shown.

(g) Obvious (Bold): Each time an input value is used in your story, you must make it obvious. If they simply blend in it makes it hard for me to find them. The recommended way is to surround its use with bold markup. Instead of saying `$x` say `<b>$x</b>`.

(h) olin: We provide the olin subroutine to help you retrieve your inputs.

You can place a copy of this subroutine at or near the beginning or end of every program you write that requires online inputs. You are welcome to use copy-paste to insert it into your program, but verify that it copied correctly, especially the quote marks. Look in the textbook index for "olin" to find an explanation of this subroutine.

Since you will use this on several assignments, it might be worth your time to clean up the quote marks and other errors that often come with copy and paste.

The olin Subroutine

```
sub olin { my ( $name, $res ) = @_;
  if ( $_olin eq "" ) { $_olin = "&" . <STDIN> }
```

```
  if ( @_ == 0 ) { return $_olin }
  if ( $_olin =~ /&$name=([^&]*)/ ) {
    $res = $1; $res =~ s/[+]/ /g;
    $res =~ s/%(..)/pack('c',hex($1))/ge }
  return $res }
```

Using olin:

Include the olin subroutine in your program. It can be anywhere in your program, top or bottom or in between. I recommend putting it near the top. Then include one or more calls to olin as shown here.

You normally call olin with two parameters, like this:

```
$x = olin ( "name", "default" );
```

In this case, olin searches the inputs that were sent from the form on your webpage, and returns the value of the first field whose name is matched. If there is no such field, olin returns the value you provided as "default."

Notice that `chomp` is not used with olin. It is not needed. chomp is mostly used with STDIN with keyboard inputs.

Although it is not recommended, you can call olin with one parameter, like this:

```
$x = olin ( "name" );
```

In this case, if there is no matching field, olin returns the "undefined" value. Because it is harder to work with, this is probably not a good approach.

For debugging purposes, you can call olin with no parameters, like this:

```
$x = olin;
```

In this case, olin returns the entire input string that was sent by the browser, with `&` added to the front. When you are debugging it can be very informative to see what that input string really looks like. For example:

```
print olin;
```

2.11 Wed, Dec 02

- Exams: 3mq
- Have previously read textbook Units 1 through 4.
- Finish reading Unit 4 (Decisions) before class
- Daily Update d10
- Three-Minute Quiz q10
- g40 Gift1
- On Fri, Dec 04 we have **Exam v1**

Our first major test is on Friday (Fri, Dec 04). Today would be a good day to review the test questions and ask about anything confusing.

2.11.1 Task g40: Gift1

- Introduced: Wed, Dec 02
- 11pt Due Date: Wed, Dec 02, 23:59.
- 10pt Due Date: Fri, Dec 04, 23:59.
- 9pt Due Date: Mon, Dec 07, 23:59.
- 8pt Due Date: Wed, Dec 09, 23:59.
- 7pt Due Date: Wed, Jan 06, 23:59.
- Grading Label: **g40**

GradeBot: This is a GradeBot task. If your program runs correctly and has proper indenting, GradeBot will send it to me for you. The general rules and explanations in section C.3 (page 139) apply, including the required comment line.

`# cis101 g40 `*roll-book-name* is the required comment line. There may be customizations that are different for each student based on this comment line. Your wordings may differ from study guide examples.

GradeBot is at http://gradebot.tk/ and http://gbot.dc.is2.byuh.edu/.

Style: Approved style is required on this task. The rules in appendix E (page 151) apply. Be careful with your line breaks, spacing, indenting, and choice of variable names.

There is a chapter in the textbook, probably about chapter 20, probably about page 112, that talks about numeric comparisons.

Task: See whether you can afford a certain gift or not.

Sample Execution:

```
 1:   "How much money do you have? " (no \n)
in>   ........................."1.00"
 2:   "How much does the gift cost? " (no \n)
in>   ........................."2.00"
(next line depends on the numbers)
 3:   "Sorry. You cannot afford it."
 3:   "Perfect. You can afford it."
eof   (end of output)
```

2.12 Fri, Dec 04

- Exams: major exam v1
- Have previously read textbook Units 1 through 4.
- Be working on reading textbook Unit 5.
- Daily Update d11
- no Three-Minute Quiz
- **Exam v1**

2.13   Mon, Dec 07

- Exams: 3mq
- Have previously read textbook Units 1 through 4.
- Be working on reading textbook Unit 5.
- Daily Update d12
- Three-Minute Quiz q12
- g41 Scholarship
- 14:30-14:40 10m Policy Test retake (lab time)

2.13.1   Task g41: Scholarship

- Introduced: Mon, Dec 07
- 11pt Due Date: Mon, Dec 07, 23:59.
- 10pt Due Date: Wed, Dec 09, 23:59.
- 9pt Due Date: Fri, Dec 11, 23:59.
- 8pt Due Date: Mon, Dec 14, 23:59.
- 7pt Due Date: Mon, Jan 11, 23:59.
- Grading Label: **g41**

GradeBot: This is a GradeBot task. If your program runs correctly and has proper indenting, GradeBot will send it to me for you. The general rules and explanations in section C.3 (page 139) apply, including the required comment line.

`# cis101 g41` *roll-book-name* is the required comment line. There may be customizations that are different for each student based on this comment line. Your wordings may differ from study guide examples.

GradeBot is at http://gradebot.tk/ and http://gbot.dc.is2.byuh.edu/.

The Scholarship Office has released the new Scholarship Brackets. There is a GPA range for each level of scholarship. Ask for the GPA and tell what scholarship is awarded.

2.14   Wed, Dec 09

- Exams: 3mq
- Have previously read textbook Units 1 through 4.
- Be working on reading textbook Unit 5.
- Daily Update d13
- Three-Minute Quiz q13
- o4H High Low
- 14:30-14:40 10m Policy Test retake (lab time)

2.14.1   Task o4H: High Low

- Introduced: Wed, Dec 09
- 11pt Due Date: Wed, Dec 09, 23:59.
- 10pt Due Date: Fri, Dec 11, 23:59.
- 9pt Due Date: Mon, Dec 14, 23:59.
- 8pt Due Date: Wed, Dec 16, 23:59.
- 7pt Due Date: Wed, Jan 13, 23:59.
- Grading Label: **o4H**

Online Plus Code: This is an online task. For this one you **also** need to send me the code that you write, and it also needs to work when I test it online. The general rules in section C.1 (page 135) apply, including email subject line but not program comment line.

`cis101 o4H` *roll-book-name* is the required subject line. There is no required comment line because this is not a GradeBot task.

Summary: Program the high-low guessing game to run online. This gives us practice using hidden fields.

This program does not use loops, arrays, or subroutines (except olin). It mostly uses if statements.

You will probably have the following if statements:

```
if ( just getting started ) { pick a number }
elsif ( they guessed it ) { congratulate; pick a number }
elsif ( too low ) { whine }
elsif ( too high ) { whine }
```

Just getting started means that you compare things and determine that there are no numbers yet, just default values.

Pick a number means a random integer between 1 and 100.

Congratulate means tell them they guessed the number. You might do that immediately, or you might put the message in a variable to be shown later.

Whine means tell them they were wrong. You might do that immediately, or you might put the message in a variable to be shown later.

Start: When your program first starts, it must pick a number to be guessed, which must be between 1 and 100, inclusive.

Autofocus: Your webpage must have a field into which the guess is entered. That field (and no other) must have autofocus.

Hidden: Your webpage must have a hidden field that contains the number to be guessed. While you are developing your program, you probably need that field to be visible, but when you are done you must make it hidden.

Submit: Your webpage must have a submit button. Not all browsers require a submit button, but some do.

Feedback: As your program runs, it must compare the current guess to the secret number and tell whether it is too low, too high, or correct.

Win: After a correct guess, your program must immediately pick a new number to be guessed and continue playing the game.

2.15   Fri, Dec 11

- Exams: 3mq
- Have previously read textbook Units 1 through 4.
- Be working on reading textbook Unit 5.
- LD Drop: Today is the last day to drop a class
- Daily Update d14
- Three-Minute Quiz q14
- g42 Birthday
- 14:10-14:20 10m Policy Test
- 14:30-14:40 10m Policy Test retake (lab time)

2.15.1   Task g42: Birthday

- Introduced: Fri, Dec 11
- 11pt Due Date: Fri, Dec 11, 23:59.
- 10pt Due Date: Mon, Dec 14, 23:59.
- 9pt Due Date: Wed, Dec 16, 23:59.
- 8pt Due Date: Fri, Dec 18, 23:59.
- 7pt Due Date: Fri, Jan 15, 23:59.
- Grading Label: **g42**

GradeBot: This is a GradeBot task. If your program runs correctly and has proper indenting, GradeBot will send it to me for you. The general rules and explanations in section C.3 (page 139) apply, including the required comment line.

`# cis101 g42` *roll-book-name* is the required comment line. There may be customizations that are different for each student based on this comment line. Your wordings may differ from study guide examples.

GradeBot is at http://gradebot.tk/ and http://gbot.dc.is2.byuh.edu/.

Style: Approved style is required on this task. The rules in appendix E (page 151) apply. Be careful with your line breaks, spacing, indenting, and choice of variable names.

Unit 4 of the textbook talks about numeric decisions. This program will use that knowledge.

Task: Tell how many years old a person is. This may involve several if statements.

Sample Execution:

```
GradeBot engaged your program in this dialog:
note  GBot "# debug output lines are permitted"
note  GBot "# For this program, assume today is Aug 15 2014."
  1:  "Please enter your name: " (no \n)
 in>   ......................."Michelle"
  2:  "What month (number) were you born, Michelle? " (no \n)
 in>   .........................................."10"
  3:  "What day were you born, Michelle? " (no \n)
 in>   ................................."25"
  4:  "What year were you born, Michelle? " (no \n)
 in>   ................................."1984"
  5:  "Ah. You were born on 1984-10-25."
  6:  ""
  7:  "Michelle, you are 29 years old."
 eof  (end of output)
```

2.16 Mon, Dec 14

- Exams: 3mq
- Have previously read textbook Units 1 through 4.
- Be working on reading textbook Unit 5.
- Daily Update d15
- Three-Minute Quiz q15
- o4P Presents

2.16.1 Task o4P: Presents

- Introduced: Mon, Dec 14
- 11pt Due Date: Mon, Dec 14, 23:59.
- 10pt Due Date: Wed, Dec 16, 23:59.
- 9pt Due Date: Fri, Dec 18, 23:59.
- 8pt Due Date: Mon, Jan 04, 23:59.
- 7pt Due Date: Mon, Jan 18, 23:59.
- Grading Label: **o4P**

Online Plus Code: This is an online task. For this one you **also** need to send me the code that you write, and it also needs to work when I test it online. The general rules in section C.1 (page 135) apply, including email subject line but not program comment line.

`cis101 o4P` *roll-book-name* is the required subject line. There is no required comment line because this is not a GradeBot task.

Summary: Create a competitive game.

Requirements: In this game, we start with a random number of objects, between 10 and 20. Two players take turns removing one or two objects. The player to take the last object wins. See the example posted by my name on the projects page.

For objects, I am using Christmas Presents. You can use that if you like or something else. Make up a good story line to go with your choice.

You will need several input fields: (a) a way to identify which player has the next move, (b) a way to tell how many objects are left, (c) two submit buttons to tell whether one or two objects were removed.

During development, it is recommended that (a) and (b) be regular text fields. After development, it is required that they be hidden (changed to type=hidden).

Optional: Graphics are NOT required. Fancy styling is not required. You are welcome to copy the HTML and CSS code that I have used. You are welcome to use my present graphics.

You must write your own index.cgi program code.

Recommended program outline:

(a) Gather up all the inputs.

(b) Figure out whether you are just starting, or whether you are already in the middle of a contest.

(c) If just starting, create a countdown. Otherwise, update the countdown.

(d) If the game has been won, restart with a new countdown.

(e) Repaint the screen, complete with hidden inputs.

2.17 Wed, Dec 16

- Exams: 3mq
- Have previously read textbook Units 1 through 5.
- Finish reading Unit 5 (Decisions, String) before class
- Daily Update d16
- Three-Minute Quiz q16
- g51 Phone Book
- On Fri, Dec 18 we have **Exam v2**

Our next major test is on Friday (Fri, Dec 18). Today would be a good day to review the test questions and ask about anything confusing.

2.17.1 Task g51: Phone Book

- Introduced: Wed, Dec 16
- 11pt Due Date: Wed, Dec 16, 23:59.
- 10pt Due Date: Fri, Dec 18, 23:59.
- 9pt Due Date: Mon, Jan 04, 23:59.
- 8pt Due Date: Wed, Jan 06, 23:59.
- 7pt Due Date: Wed, Jan 20, 23:59.
- Grading Label: **g51**

Task: Tell what page of the phone book has the name you seek.

GradeBot: This is a GradeBot task. If your program runs correctly and has proper indenting, GradeBot will send it to me for you. The general rules and explanations in section C.3 (page 139) apply, including the required comment line.

`# cis101 g51 `*roll-book-name* is the required comment line. There may be customizations that are different for each student based on this comment line. Your wordings may differ from study guide examples.

GradeBot is at http://gradebot.tk/ and http://gbot.dc.is2.byuh.edu/.

Style: Approved style is required on this task. The rules in appendix E (page 151) apply. Be careful with your line breaks, spacing, indenting, and choice of variable names.

There is a chapter in the textbook, probably about chapter 26, probably about page 142, that talks about string comparisons. They are basically like numeric comparisons, but use different operators.

Literal strings like Davis and Dodson should be enclosed in quote marks so they are not misunderstood. See section for details.

Sample Execution:

```
 1:   "Page 20 of the phone book starts with Davis and ends with Dodson."
 2:   "What name do you seek? " (no \n)
in>   ......................"Ditto"
 3:   "Ditto would be on page 20."
eof   (end of output)
```

2.18   Fri, Dec 18

- Exams: major exam v2
- Have previously read textbook Units 1 through 5.
- Be working on reading textbook Unit 6.
- Daily Update d17
- no Three-Minute Quiz
- **Exam v2**

2.19   Mon, Jan 04

- Exams: 3mq
- Have previously read textbook Units 1 through 6.
- Finish reading Unit 6 (Loops) before class.
- Daily Update d18
- Three-Minute Quiz q18
- g62 For Loop

2.19.1   Task g62: For Loop

- Introduced: Mon, Jan 04
- 11pt Due Date: Mon, Jan 04, 23:59.
- 10pt Due Date: Wed, Jan 06, 23:59.
- 9pt Due Date: Fri, Jan 08, 23:59.
- 8pt Due Date: Mon, Jan 11, 23:59.
- 7pt Due Date: Mon, Jan 25, 23:59.
- Grading Label: **g62**

I am being more strict about variable names on this one. To the extent reasonably possible, variable names should helpfully identify the contents of the variable.

Specifically, you should have one well-named variable for each of the inputs that you receive, and additional well-named variables as needed to operate the loop.

Variable names must indicate what the contents of each variable is, to the extent reasonably possible. Variable names must not imply something that is not true, or becomes untrue as the program runs.

This task requires a "for loop" and there are some style rules that apply specifically to for loops. You can review them in E.6 (page 154).

Specifically, the initialization, test, and step phases must be specified in the for-loop way, and not in the while-loop way.

GradeBot: This is a GradeBot task. If your program runs correctly and has proper indenting, GradeBot will send it to me for you. The general rules and explanations in section C.3 (page 139) apply, including the required comment line.

`# cis101 g62 `*`roll-book-name`* is the required comment line. There may be customizations that are different for each student based on this comment

line. Your wordings may differ from study guide examples.

GradeBot is at http://gradebot.tk/ and http://gbot.dc.is2.byuh.edu/.

Style: Approved style is required on this task. The rules in appendix E (page 151) apply. Be careful with your line breaks, spacing, indenting, and choice of variable names.

There are chapters in the textbook, probably about chapters 31 and 33, probably starting about page 166, that talks about loops and for loops.

Task: Read in three numbers: the starting point, the ending point, and the step size. Count from the starting point to the ending point, adding the step size each time, one number per line.

Use a normal "for" loop (pre-test, not interior test).

When incrementing the loop variable, you should use the more efficient `$n+=$x` approach rather than the less efficient `$n=$n+$x` approach.

Note: if you do not hit the ending point exactly, that's okay. But you should get as close as possible to the ending point without going beyond it.

Sample Execution:

```
 1:  "Where should I start? " (no \n)
in>   ....................."1"
 2:  "Where should I end? " (no \n)
in>   ...................."5"
 3:  "What should I count by? " (no \n)
in>   ......................"1"
 4:  "1"
 5:  "2"
 6:  "3"
 7:  "4"
 8:  "5"
12:  "Done!"
eof  (end of output)
```

2.20 Wed, Jan 06

- Exams: 3mq
- Have previously read textbook Units 1 through 6.
- Be working on reading textbook Unit 7.
- Daily Update d19
- Three-Minute Quiz q19
- o6D Multi Dice

2.20.1 Task o6D: Multi Dice

- Introduced: Wed, Jan 06
- 11pt Due Date: Wed, Jan 06, 23:59.
- 10pt Due Date: Fri, Jan 08, 23:59.
- 9pt Due Date: Mon, Jan 11, 23:59.
- 8pt Due Date: Wed, Jan 13, 23:59.
- 7pt Due Date: Wed, Jan 27, 23:59.
- Grading Label: **o6D**

Online Plus Code: This is an online task. For this one you **also** need to send me the code that you write, and it also needs to work when I test it online. The general rules in section C.1 (page 135) apply, including email subject line but not program comment line.

`cis101 o6D` *roll-book-name* is the required subject line. There is no required comment line because this is not a GradeBot task.

There is a chapter in the textbook, probably about chapter 31, probably about page 163, that talks about loops (repeated actions).

Summary: Make a web program that rolls the number of dice selected. Each die is rolled randomly, independent of the others.

See task oD and task oF for how-to information.

Requirements:

(a) **ID:** Include your name in the title and in an h1 at the top of the page.

(b) **Autofocus/Blank:** Your program must display and accept a (numeric) input field. Use autofocus (required!) to place the cursor in that field. Blank out the numeric field between runs. Do not carry forward the latest entry.

(c) **Max:** Near your input field, you must say what your maximum number of dice is. You can pick any maximum between 10 and 100. If the user

request is larger than your limit, your loop must use your limit instead and print a suitable warning message.

(d) **Submit:** Your program must have a submit button. When the submit button is pressed, the screen should be redrawn, complete with the blank for numeric input, followed by "n" pictures of dice, where "n" is the number that was keyed into the input field. No particular title is required.

Instead of using `<img>` images of dice, you are welcome to use the HTML codes for dice.

⚀ is the html code for die face 1.
⚁ is the html code for die face 2.
⚂ is the html code for die face 3.
⚃ is the html code for die face 4.
⚄ is the html code for die face 5.
⚅ is the html code for die face 6.

(e) **Image Size:** Each image should have a displayed size that is large enough to see clearly, but no larger than 100px wide and 100px tall. You can use width= and height= if you like, or you can resize your images.

(f) **Alternatives:** Instead of normal, six-sided dice you can use something else (maybe slot machine images or playing cards) but it must have at least four alternatives from which one is selected.

2.21   Fri, Jan 08

- Exams: 3mq
- Have previously read textbook Units 1 through 6.
- Be working on reading textbook Unit 7.
- Daily Update d20
- Three-Minute Quiz q20
- g64 Factors

2.21.1   Task g64: Factors

- Introduced: Fri, Jan 08
- 11pt Due Date: Fri, Jan 08, 23:59.
- 10pt Due Date: Mon, Jan 11, 23:59.
- 9pt Due Date: Wed, Jan 13, 23:59.
- 8pt Due Date: Fri, Jan 15, 23:59.
- 7pt Due Date: Fri, Jan 29, 23:59.
- Grading Label: **g64**

GradeBot: This is a GradeBot task. If your program runs correctly and has proper indenting, GradeBot will send it to me for you. The general rules and explanations in section C.3 (page 139) apply, including the required comment line.

`# cis101 g64 ` *roll-book-name* is the required comment line. There may be customizations that are different for each student based on this comment line. Your wordings may differ from study guide examples.

GradeBot is at http://gradebot.tk/ and http://gbot.dc.is2.byuh.edu/.

Task: Find the factors of a number.

Style: Approved style (naming, spacing, and indenting) is required for credit.

Read in a number "n". Print the numbers (integers) from 1 to n, telling whether each divides into n perfectly. It divides perfectly if there is no remainder.

Sample Executions:

```
GradeBot would have engaged your program in this dialog:
note  GBot "# debug output lines are permitted"
 in>  "8"
```

```
 1:  "1 divides into 8 perfectly."
 2:  "2 divides into 8 perfectly."
 3:  "3 divides into 8 leaving a remainder of 2."
 4:  "4 divides into 8 perfectly."
 5:  "5 divides into 8 leaving a remainder of 3."
 6:  "6 divides into 8 leaving a remainder of 2."
 7:  "7 divides into 8 leaving a remainder of 1."
 8:  "8 divides into 8 perfectly."
eof  (end of output)
```

2.22 Mon, Jan 11

- Exams: 3mq
- Have previously read textbook Units 1 through 6.
- Be working on reading textbook Unit 7.
- Daily Update d21
- Three-Minute Quiz q21
- g66 Triangle

Style is important. I am particularly checking variable names as I do my grading. For each variable name, if I can easily come up with a meaningful variable name, I expect that you can too, and you should use it. (It does not need to be the same as what I would have picked.) It should not be a misleading variable name. If it is meaningful, the meaning should match the actual use of the variable.

Also, indenting is important. Gradebot enforces that. But newlines should be used appropriately. Every time you have an opening curly brace, the next statement should start on a new line, properly indented. Similarly, after every semicolon, a newline is expected. Some exceptions are allowed, as mentioned in the study guide. Closing curly braces should normally be at the end of the previous statement. Really long lines are discouraged.

2.22.1 Task g66: Triangle

- Introduced: Mon, Jan 11
- 11pt Due Date: Mon, Jan 11, 23:59.
- 10pt Due Date: Wed, Jan 13, 23:59.
- 9pt Due Date: Fri, Jan 15, 23:59.
- 8pt Due Date: Mon, Jan 18, 23:59.
- 7pt Due Date: Mon, Feb 01, 23:59.
- Grading Label: **g66**

GradeBot: This is a GradeBot task. If your program runs correctly and has proper indenting, GradeBot will send it to me for you. The general rules and explanations in section C.3 (page 139) apply, including the required comment line.

`# cis101 g66 roll-book-name` is the required comment line. There may be customizations that are different for each student based on this comment line. Your wordings may differ from study guide examples.

GradeBot is at http://gradebot.tk/ and http://gbot.dc.is2.byuh.edu/.

Objective: Do nested loops.

Summary: Given the size of the triangle, draw a solid triangle using stars.

Notice that there are spaces between the stars (but not after the last star) to make the triangle look more regular. This also makes the task slightly harder.

Hint: There are "n" rows of stars. Each row of stars starts with zero or more spaces. What is the formula for how many spaces? What is the formula for how many stars?

Suggestion: Use nested for loops.

Sample Execution:

```
GradeBot would have engaged your program in this dialog:
note  GBot "# debug output lines are permitted"
  1:  "What size triangle? " (no \n)
 in>   ..............."3"
  2:  "  *"
  3:  " * *"
  4:  "* * *"
  5:  "Done!"
 eof  (end of output)
```

2.23 Wed, Jan 13

- Exams: 3mq
- Have previously read textbook Units 1 through 6.
- Be working on reading textbook Unit 7.
- Daily Update d22
- Three-Minute Quiz q22
- g46 Cheese
- On Fri, Jan 15 we have **Exam v3**

Our next major test is on Friday (Fri, Jan 15). Today would be a good day to review the test questions and ask about anything confusing.

2.23.1 Task g46: Cheese

- Introduced: Wed, Jan 13
- 11pt Due Date: Wed, Jan 13, 23:59.
- 10pt Due Date: Fri, Jan 15, 23:59.
- 9pt Due Date: Mon, Jan 18, 23:59.
- 8pt Due Date: Wed, Jan 20, 23:59.
- 7pt Due Date: Wed, Feb 03, 23:59.
- Grading Label: **g46**

GradeBot: This is a GradeBot task. If your program runs correctly and has proper indenting, GradeBot will send it to me for you. The general rules and explanations in section C.3 (page 139) apply, including the required comment line.

`# cis101 g46 roll-book-name` is the required comment line. There may be customizations that are different for each student based on this comment line. Your wordings may differ from study guide examples.

GradeBot is at http://gradebot.tk/ and http://gbot.dc.is2.byuh.edu/.

Style: Approved style is required on this task. The rules in appendix E (page 151) apply. Be careful with your line breaks, spacing, indenting, and choice of variable names.

There is a chapter in the textbook, probably about chapter 20, probably about page 112, that talks about numeric comparisons.

Task: See how many packages of cheese should be purchased.

There are two ways to approach this task. The easy way is to check the

remainder after division. If it is non-zero, add one more package.

The other way involves a clever use of int together with adding and sub-
tracting one. No if statements are involved, but it is more on the clever side
of things.

2.24   Fri, Jan 15

- Exams: major exam v3
- Have previously read textbook Units 1 through 6.
- Be working on reading textbook Unit 7.
- Daily Update d23
- no Three-Minute Quiz
- **Exam v3**

2.25   Mon, Jan 18

• No class today in honor of Human Rights Day

2.26   Wed, Jan 20

- Exams: 3mq
- Have previously read textbook Units 1 through 6.
- Be working on reading textbook Unit 7.
- LD Withdraw: Today is the last day to withdraw from class
- Daily Update d24
- Three-Minute Quiz q24
- o6M Maze

2.26.1   Task o6M: Maze

- Introduced: Wed, Jan 20
- 11pt Due Date: Wed, Jan 20, 23:59.
- 10pt Due Date: Fri, Jan 22, 23:59.
- 9pt Due Date: Mon, Jan 25, 23:59.
- 8pt Due Date: Wed, Jan 27, 23:59.
- 7pt Due Date: Wed, Feb 10, 23:59.
- Grading Label: **o6M**

Online Plus Code: This is an online task. For this one you **also** need to send me the code that you write, and it also needs to work when I test it online. The general rules in section C.1 (page 135) apply, including email subject line but not program comment line.

$\boxed{\texttt{cis101 o6M } \textit{roll-book-name}}$ is the required subject line. There is no required comment line because this is not a GradeBot task.

There is a chapter in the textbook, probably about chapter 31, probably about page 163, that talks about loops (repeated actions).

Summary: Make a web program that builds a maze of the dimensions requested. Use a nested loop. Each maze wall is constructed randomly, independent of the others.

Mouse and Cheese: For fun, I added a drawing of a mouse and some cheese on my sample program. You do not need to do anything like that, or any other styling except one thing: you need the walls to line up properly without big gaps, as explained below.

Requirements:

(a) **Identity:** Include your name in the page title and in an h1 at the top of the page, or in a statement at the bottom of your page.

(b) **Two Fields:** Your program must display and accept two (numeric) input fields, one for width (number of columns) and one for height (number of rows). They must be clearly labeled.

(c) **Autofocus:** Use autofocus (required!) to place the cursor in the first field.

(d) **Sticky/Defaults:** Remember the numeric field values between runs. Default them to the maximum values you will accept. Draw your initial maze using those defaults.

(e) **Submit:** Your program must have a submit button.

(f) **Max Stated:** Near your input fields, you must say what your maximum number of rows and columns is. You can pick any maximum between 10 and 100, but the resulting maze must fit on a normal computer screen (1000 px wide).

(g) **Over Max:** If any user request is larger than your limit, your loop must use your limit instead and a suitable warning should be printed.

(h) **Walls:** The walls should be randomly either / or \ with a 50-50 probability. You can use the actual / and \ glyphs or you can find something else (like ╱ and ╲), or draw an image and use it. You can choose to have more possibilities, but make sure the resulting maze looks at least as good as one with just the two options.

(i) **Spacing:** Control the font size and the line height (1em is recommended). Otherwise it will not look like a maze. You can use something like this in your style section.

```
div { font-size: 200%; line-height: 1em; }
```

2.27   Fri, Jan 22

- Exams: 3mq
- Have previously read textbook Units 1 through 7.
- Finish reading Unit 7 (Arrays) before class
- Daily Update d25
- Three-Minute Quiz q25
- g74 Split Date

2.27.1   Task g74: Split Date

- Introduced: Fri, Jan 22
- 11pt Due Date: Wed, Jan 20, 23:59.
- 10pt Due Date: Fri, Jan 22, 23:59.
- 9pt Due Date: Mon, Jan 25, 23:59.
- 8pt Due Date: Wed, Jan 27, 23:59.
- 7pt Due Date: Wed, Feb 10, 23:59.
- Grading Label: **g74**

GradeBot: This is a GradeBot task. If your program runs correctly and has proper indenting, GradeBot will send it to me for you. The general rules and explanations in section C.3 (page 139) apply, including the required comment line.

`# cis101 g74` *roll-book-name* is the required comment line. There may be customizations that are different for each student based on this comment line. Your wordings may differ from study guide examples.

GradeBot is at http://gradebot.tk/ and http://gbot.dc.is2.byuh.edu/.

Use the split command to convert a string into an array. The incoming string is a date in mm/dd/yy format.

Then use an array of month names to replace the number with a word. (Using a series of if-statements will not be acceptable.)

Then print the result.

2.28   Mon, Jan 25

- Exams: 3mq
- Have previously read textbook Units 1 through 7.
- Be working on reading textbook Unit 8.
- Daily Update d26
- Three-Minute Quiz q26
- g73 Boring

2.28.1   Task g73: Boring

- Introduced: Mon, Jan 25
- 11pt Due Date: Mon, Jan 25, 23:59.
- 10pt Due Date: Wed, Jan 27, 23:59.
- 9pt Due Date: Fri, Jan 29, 23:59.
- 8pt Due Date: Mon, Feb 01, 23:59.
- 7pt Due Date: Mon, Feb 15, 23:59.
- Grading Label: **g73**

GradeBot: This is a GradeBot task. If your program runs correctly and has proper indenting, GradeBot will send it to me for you. The general rules and explanations in section C.3 (page 139) apply, including the required comment line.

`# cis101 g73` *roll-book-name* is the required comment line. There may be customizations that are different for each student based on this comment line. Your wordings may differ from study guide examples.

GradeBot is at http://gradebot.tk/ and http://gbot.dc.is2.byuh.edu/.

Your task is to listen to a list of foods and report which ones are boring. It is boring if you have had it before.

Our purpose is to give you training in working with arrays.

Each time around your outer loop, ask for a food. Then use an inner loop to check the list of foods already encountered. See if it is boring.

Your program may be returned for you to fix if it does not follow my version of good style which includes (a) helpful and unambiguous variable names (see appendix E.11, page 158), (b) whitespace around parens and curly braces, and (c) location of curly braces (not on a line by themselves, normally the last thing on their line, see appendix E.7, page 155).

2.29 Wed, Jan 27

- Exams: 3mq
- Have previously read textbook Units 1 through 7.
- Be working on reading textbook Unit 8.
- Daily Update d27
- Three-Minute Quiz q27
- g76 Birthday
- On Fri, Jan 29 we have **Exam v4**

Our next major test is on Friday (Fri, Jan 29). Today would be a good day to review the test questions and ask about anything confusing.

2.29.1 Task g76: Birthday

- Introduced: Wed, Jan 27
- 11pt Due Date: Wed, Jan 27, 23:59.
- 10pt Due Date: Fri, Jan 29, 23:59.
- 9pt Due Date: Mon, Feb 01, 23:59.
- 8pt Due Date: Wed, Feb 03, 23:59.
- 7pt Due Date: Wed, Feb 17, 23:59.
- Grading Label: **g76**

GradeBot: This is a GradeBot task. If your program runs correctly and has proper indenting, GradeBot will send it to me for you. The general rules and explanations in section C.3 (page 139) apply, including the required comment line.

`# cis101 g76 roll-book-name` is the required comment line. There may be customizations that are different for each student based on this comment line. Your wordings may differ from study guide examples.

GradeBot is at http://gradebot.tk/ and http://gbot.dc.is2.byuh.edu/.

Style: Approved style is required on this task. The rules in appendix E (page 151) apply. Be careful with your line breaks, spacing, indenting, and choice of variable names.

Task: Tell how many years old a person is.

This task is very similar to g42 that you may have done previously. In this case, the birth date will come on a single line instead of being separate questions.

2.30   Fri, Jan 29

- Exams: major exam v4
- Have previously read textbook Units 1 through 7.
- Be working on reading textbook Unit 8.
- Daily Update d28
- no Three-Minute Quiz
- **Exam v4**

2.31 Mon, Feb 01

- Exams: 3mq
- Have previously read textbook Units 1 through 7.
- Be working on reading textbook Unit 8.
- Daily Update d29
- Three-Minute Quiz q29
- g77 AM/PM Time to 24H

2.31.1 Task g77: AM/PM Time to 24H

- Introduced: Mon, Feb 01
- 11pt Due Date: Mon, Feb 01, 23:59.
- 10pt Due Date: Wed, Feb 03, 23:59.
- 9pt Due Date: Fri, Feb 05, 23:59.
- 8pt Due Date: Mon, Feb 08, 23:59.
- 7pt Due Date: Fri, Feb 19, 23:59.
- Grading Label: **g77**

GradeBot: This is a GradeBot task. If your program runs correctly and has proper indenting, GradeBot will send it to me for you. The general rules and explanations in section C.3 (page 139) apply, including the required comment line.

`# cis101 g77 `*roll-book-name* is the required comment line. There may be customizations that are different for each student based on this comment line. Your wordings may differ from study guide examples.

GradeBot is at http://gradebot.tk/ and http://gbot.dc.is2.byuh.edu/.

Task: Convert time from AM/PM format to 24-hour format.

Time is provided as HH:MM AM or HH:MM PM. You can use "split" to isolate information you want. You can use "if" to handle AM versus PM. You can use "printf" to format the output line.

There are other approaches that could also be successful.

Sample Executions:

```
GradeBot would have engaged your program in this dialog:
note  GBot "# debug output lines are permitted"
  1:  "Please enter the time as HH:MM AM or HH:MM PM: " (no \n)
 in>    ............................................"11:59 PM"
```

```
 2:  "11:59 PM is 23:59"
eof  (end of output)
```

2.32 Wed, Feb 03

- Exams: 3mq
- Have previously read textbook Units 1 through 7.
- Be working on reading textbook Unit 8.
- Daily Update d30
- Three-Minute Quiz q30
- g79 Perfect Shuffle

2.32.1 Task g79: Perfect Shuffle

- Introduced: Wed, Feb 03
- 11pt Due Date: Wed, Feb 03, 23:59.
- 10pt Due Date: Fri, Feb 05, 23:59.
- 9pt Due Date: Mon, Feb 08, 23:59.
- 8pt Due Date: Wed, Feb 10, 23:59.
- 7pt Due Date: Fri, Feb 19, 23:59.
- Grading Label: **g79**

GradeBot: This is a GradeBot task. If your program runs correctly and has proper indenting, GradeBot will send it to me for you. The general rules and explanations in section C.3 (page 139) apply, including the required comment line.

`# cis101 g79` *roll-book-name* is the required comment line. There may be customizations that are different for each student based on this comment line. Your wordings may differ from study guide examples.

GradeBot is at http://gradebot.tk/ and http://gbot.dc.is2.byuh.edu/.

You should use reasonable style, but I do not plan to be very picky.

Purpose: To gain experience in working with arrays.

Task: We will do a perfect shuffle of a deck of cards.

To do this, we perfectly divide the cards into a first half and a second half. If the number of cards is odd, the first half gets one more than the second. As we combine, we alternate by using the last card from the top, and then the last card from the bottom, until all cards are placed. Then we report the result.

Odd Example: 1 2 3 4 5. The top half is 1 2 3. The bottom half is 4 5. The last card from the top becomes the last card of the result. So, as we shuffle,

the 3 goes down, then the 5, then the 2, then the 4, then the 1. The result is: 1 4 2 5 3.

Even Example: 1 2 3 4. The top half is 1 2. The bottom half is 3 4. The last card from the top becomes the last card of the result. So, as we shuffle, the 2 goes down, then the 4, then the 1, then the 3. The result is: 3 1 4 2.

Suggestion: Use split to divide your input into an array. Divide the array into two smaller arrays (top and bottom). Use commands like push, pop, shift, and unshift, to move cards from the smaller arrays into a final result array.

2.33   Fri, Feb 05

- Exams: 3mq, 3mq makeup
- Have previously read textbook Units 1 through 7.
- Be working on reading textbook Unit 8.
- Daily Update d31
- Three-Minute Quiz q31
- Three-Minute Makeup (first one)
- g45 Gift2

2.33.1   Task g45: Gift2

- Introduced: Fri, Feb 05
- 11pt Due Date: Fri, Feb 05, 23:59.
- 10pt Due Date: Mon, Feb 08, 23:59.
- 9pt Due Date: Wed, Feb 10, 23:59.
- 8pt Due Date: Fri, Feb 12, 23:59.
- 7pt Due Date: Fri, Feb 19, 23:59.
- Grading Label: **g45**

GradeBot: This is a GradeBot task. If your program runs correctly and has proper indenting, GradeBot will send it to me for you. The general rules and explanations in section C.3 (page 139) apply, including the required comment line.

`# cis101 g45 `*roll-book-name* is the required comment line. There may be customizations that are different for each student based on this comment line. Your wordings may differ from study guide examples.

GradeBot is at http://gradebot.tk/ and http://gbot.dc.is2.byuh.edu/.

Warning: This one is a bit tricky. Read it carefully and think about it carefully. It sounds easy, but it is actually somewhat difficult.

Objective: Learn how to use if/else skillfully.

Summary: Consider two gifts and the money you have available. Tell which gifts, if any, to purchase.

Task: You are shopping for wedding gifts for a good friend. They have registered their wants on a bridal registry. There are two items not yet purchased. Ask for the price of gift 1. Ask for the price of gift 2. Ask for the amount of money you have. If you can get both, say so. If you can only get one, tell the **most expensive** thing you can afford. If you cannot afford

either, say so.

For cases that are not covered by these instructions, GradeBot will tell you
what it wants you to say in each case.

Sample Execution:

```
 1:   "What is the price of item 1? " (no \n)
in>   ............................"1"
 2:   "What is the price of item 2? " (no \n)
in>   ............................"2"
 3:   "How much money do you have? " (no \n)
in>   ............................"3"
 4:   "Buy both!"
eof  (end of output)
```

2.34   Mon, Feb 08

- Exams: 3mq, 3mq makeup
- Have previously read textbook Units 1 through 8.
- Finish reading Unit 8 (Subroutines) before class.
- Daily Update d32
- Three-Minute Quiz q32
- Three-Minute Makeup
- o8F Farm8

2.34.1   Task o8F: Farm 2

- Introduced: Mon, Feb 08
- 11pt Due Date: Mon, Feb 08, 23:59.
- 10pt Due Date: Wed, Feb 10, 23:59.
- 9pt Due Date: Fri, Feb 12, 23:59.
- 8pt Due Date: Mon, Feb 15, 23:59.
- 7pt Due Date: Fri, Feb 19, 23:59.
- Grading Label: **o8F**

This one is a bit harder but is very important.

Online Plus Code: This is an online task. For this one you **also** need to send me the code that you write, and it also needs to work when I test it online. The general rules in section C.1 (page 135) apply, including email subject line but not program comment line.

`cis101 o8F` *roll-book-name* is the required subject line. There is no required comment line because this is not a GradeBot task.

There is a unit in the textbook, probably unit 8, probably about page 217, that talks about subroutines.

Task: Similar to o6F (above) but using subroutines. And multiple crops. You are planting a farm. Ask for planting directions. Show the results.

Objective: Learn to use subroutines to avoid repetition of code.

Danger Area: Be careful to properly use local variables in your subroutines. It is easy to accidentally have a global variable.

I will read your code to verify that you used the proper structure in writing your program. In your email, include a link to your online program. I will test it online to see how well it works.

Main Program

Your main program must have the following lines, or something very similar.

```
@crops = ( "a", "b", "c", "d" ); # four or more crops
foreach $crop ( @crops ) { plant ( $crop ) } # ask how many
# print an appropriate submit button
foreach $crop ( @crops ) { harvest ( $crop ) } # show results
```

Rules About Crops

Have a crops array and initialize it very near the top of your program.

You must have **at least four crops, and at most six crops.** The crops do not have to be actual farming crops. They can be something funny or weird. Pokemon. Soldiers. Books. Whatever.

You must make an array (list) of the crops you are farming. This is the **only** place that the literal names of any of your crops may appear in the program. Everything else must be done using variables.

The text strings in @crops will need to be usable for (a) display labels, (b) input names, and (c) image file names.

Thus, if one of the crops is `"tomato"` you can display "tomato" as part of the display label, use `name="tomato"` in the input field, use `olin("tomato")` to retrieve the quantity, and use `"tomato.jpg"` to show the picture of the tomato.

Or, more specifically, if `$fruit = "tomato"` you can display `$fruit` as part of the display label, use `name="$fruit"` in the input field, use `olin($fruit)` to retrieve the quantity, and use `"$fruit.jpg"` to show the picture of the tomato.

Subroutine: max

Have a max subroutine a put it very near the top of your program.

Your max subroutine must return a number of your choice for how many of each crop can be grown. This number must be at least 10 and at most 20. You decide.

This number must not appear elsewhere in your program. It can only appear here in the max subroutine. If this number is changed in this one place, the whole program must respond properly without further changes.

Every time you need the max, you should call this subroutine to retrieve it, or get it from a variable where you have stored it.

Subroutine: plant

You must have a subroutine named plant that accepts one parameter which is the name of the crop that is available for planting.

You **must not** use any global variables. For this subroutine, it should be easy.

For the designated crop, display a blank into which a number can be entered. Use the placeholder option to show the name of the crop being requested. (The name of the crop comes from the subroutine's parameter list.) Do not make the number "sticky." Each blank must be empty each time the screen is presented.

Put \n between the input fields in the HTML (but not on the screen). This avoids having a really wide line that I would have to scroll horizontally to examine when I view your page source. If the line is longer than about 100 characters, look for a good place to split it into two lines.

This subroutine is the **only place** where you are allowed to print the input fields that ask how many crops should be grown.

Subroutine: harvest

You must have a subroutine named harvest that accepts one parameter which is the name of the crop that will be grown.

You **must not** use any global variables. For this subroutine, you will need to be careful.

You **must not** use any global variables. For this subroutine, you will need to be careful.

I intentionally repeated that line because it is a common problem I see when grading. People have global variables in the harvest subroutine.

Print a line telling what the requested crop and requested quantity are. (The

name of the crop comes from the subroutine's parameter list. The quantity comes from a call to `olin`.)

If the quantity is larger than max, complain, and use the smaller number. Example: if max is 9 and the requested quantity is more than 9, just use 9.

Print a row of that many pictures of that crop. For example:

```
<img src=\"$crop.jpg\" alt=\"$crop\" title=\"...\">\n
```

Each image should be about 100px by 100px.

Do not repeat pieces of code if you can avoid it. Specifically, you should have exactly one place where you print the image of the crop, not more than that.

This subroutine is the **only place** where you are allowed to print the images of the crops.

Subroutine: olin

Use the subroutine olin that we previously introduced to find out what inputs were provided to your program. (This subroutine does have one global variable.)

2.35 Wed, Feb 10

- Exams: 3mq, 3mq makeup
- Have previously read textbook Units 1 through 8.
- Daily Update d33
- Three-Minute Quiz q33
- Three-Minute Makeup
- g34 Celsius
- On Fri, Feb 12 we have **Exam v5**

Our next major test is on Friday (Fri, Feb 12). Today would be a good day to review the test questions and ask about anything confusing.

2.35.1 Task g34: Celsius

- Introduced: Wed, Feb 10
- 11pt Due Date: Wed, Feb 10, 23:59.
- 10pt Due Date: Fri, Feb 12, 23:59.
- 9pt Due Date: Mon, Feb 15, 23:59.
- 8pt Due Date: Wed, Feb 17, 23:59.
- 7pt Due Date: Fri, Feb 19, 23:59.
- Grading Label: **g34**

GradeBot: This is a GradeBot task. If your program runs correctly and has proper indenting, GradeBot will send it to me for you. The general rules and explanations in section C.3 (page 139) apply, including the required comment line.

`# cis101 g34 `*roll-book-name* is the required comment line. There may be customizations that are different for each student based on this comment line. Your wordings may differ from study guide examples.

GradeBot is at http://gradebot.tk/ and http://gbot.dc.is2.byuh.edu/.

There is a chapter in the textbook, probably about chapter 11, probably about page 80, that talks about calculation.

Task: Convert temperature from Fahrenheit to Celsius. The formula is: celsius = (fahrenheit - 32) * 5 / 9

Style: Approved style is required on this task. The rules in appendix E (page 151) apply. Be careful with your line breaks, spacing, indenting, and choice of variable names.

Specifically for this assignment I am looking at proper spacing and helpful variable names. See sections and for details.

Sample Execution:

```
 1:   "Enter a temperature in Fahrenheit: " (no \n)
in>   ................................."77"
 2:   "77 in Fahrenheit equals 25 in Celsius."
eof  (end of output)
```

2.36 Fri, Feb 12

- Exams: major exam v5
- Have previously read textbook Units 1 through 8.
- Daily Update d34
- no Three-Minute Quiz
- **Exam v5**

2.37   Mon, Feb 15

- No class today in honor of Presidents Day

2.38 Wed, Feb 17

- Exams: 3mq, 3mq makeup
- Have previously read textbook Units 1 through 8.
- Daily Update d35
- Three-Minute Quiz q35
- Three-Minute Makeup
- g7A Secret Codes

2.38.1 Task g7A: Secret Codes

- Introduced: Wed, Feb 17
- 11pt Due Date: Wed, Feb 17, 23:59.
- 10pt Due Date: Fri, Feb 19, 23:59.
- Grading Label: **g7A**

GradeBot: This is a GradeBot task. If your program runs correctly and has proper indenting, GradeBot will send it to me for you. The general rules and explanations in section C.3 (page 139) apply, including the required comment line.

`# cis101 g7A` *roll-book-name* is the required comment line. There may be customizations that are different for each student based on this comment line. Your wordings may differ from study guide examples.

GradeBot is at http://gradebot.tk/ and http://gbot.dc.is2.byuh.edu/.

Use reasonable style and proper indenting.

Purpose: To gain experience in working with arrays.

Task: We will encrypt and decrypt messages using a simple transposition cipher.

Basically, Julius Caesar encrypted messages by shifting the letters. A shift of 1 would turn A into B, and X into Y. A shift of 25 (or -1) would turn B into A, and Y into X. For this task, only letters are changed. Punctuation is not changed.

The recommended approach is to use split to get at the individual characters. Then convert each character into a number. Then do some math to get the new number. Then convert the new number back into a character.

Split has two parameters, the string to be split, and the boundary characters. If the boundary characters are "" then split will return an array with each

original character by itself.

Perl has a subroutine "ord" that converts a letter into a number. A becomes 65. B becomes 66. a becomes 97. b becomes 98.

Perl has a subroutine "chr" that converts a number into a letter. 65 becomes A. 66 becomes B. 97 becomes a. 98 becomes b.

The math involves adding the shift amount to the number, and then taking the remainder from dividing by 26. Thus, 27 becomes 1.

2.39   Fri, Feb 19

- Exams: 3mq, 3mq makeup
- Have previously read textbook Units 1 through 8.
- Daily Update d36
- Three-Minute Quiz q36 (last one)
- Three-Minute Makeup (last one)
- g47 Appointment

2.39.1   Task g47: Appointment

- Introduced: Fri, Feb 19
- 11pt Due Date: Fri, Feb 19, 23:59.
- 10pt Due Date: Mon, Feb 22, 23:59.
- Grading Label: **g47**

GradeBot: This is a GradeBot task. If your program runs correctly and has proper indenting, GradeBot will send it to me for you. The general rules and explanations in section C.3 (page 139) apply, including the required comment line.

`# cis101 g47` *roll-book-name* is the required comment line. There may be customizations that are different for each student based on this comment line. Your wordings may differ from study guide examples.

GradeBot is at http://gradebot.tk/ and http://gbot.dc.is2.byuh.edu/.

Style: Approved style is required on this task. The rules in appendix E (page 151) apply. Be careful with your line breaks, spacing, indenting, and choice of variable names.

There is a chapter in the textbook, probably about chapter 20, probably about page 112, that talks about numeric comparisons.

Task: See whether the person will be late for their appointment.

The difficulty in this task is that times are expressed in 12-hour format rather than the more convenient 24-hour format. A time of 1 comes later in the day than a time of 11, for instance.

2.40 Mon, Feb 22

- Have previously read textbook Units 1 through 8.
- Daily Update d37
- no Three-Minute Quiz
- oJS JavaScript
- On Wed, Feb 24 we have **Exam v6**

Our final exam is on Wednesday (Wed, Feb 24). Today would be a good day to review the test questions and ask about anything confusing.

2.40.1 Task oJS: JavaScript

- Introduced: Mon, Feb 22
- 11pt Due Date: Mon, Feb 22, 23:59.
- 10pt Due Date: Wed, Feb 24, 23:59.
- Grading Label: **oJS**

Online: This is an online task, but you do not need to send me the code that you write. It just needs to work when I test it online. The general rules in section C.1 (page 135) apply. To request a regrade, see B.4 (page 133).

Task: Create an html webpage (a) properly linked to the student projects page. It must include (b) your name, (c) a JavaScript calculator similar to the one on my demo page, (d) autofocus into the first data field.

HTML: Notice that you will be creating a webpage only, not a CGI program.

Maintain: The demo calculator has two fields (A and B), and buttons for add, subtract, multiply, and divide. **You must keep these fields and buttons.**

Divide: On the demo calculator, the divide button fails when you divide by zero. **You must fix it so it displays a special error message,** not just "Infinity" like JavaScript would normally say.

Beyond: Go beyond the demo example. Add another button, like maybe square root, or $a^2 + b^2$, hopefully something useful.

2.41 Wed, Feb 24

- Exams: major exam v6
- Have previously read textbook Units 1 through 8.
- Daily Update d38
- no Three-Minute Quiz
- **Exam v6**, 13:00 to 15:50, GCB 111

A three-hour make-up exam will be offered on Fri, Feb 26 but it is not easy to qualify. Typically this is only offered to students who would otherwise have four exams on Wed, Feb 24 or who are traveling on university business (such as a sports team). To take this exam you must make arrangements in advance.

Chapter 3

Exam Questions

There are 21 exam questions. This chapter talks about each one and helps you avoid common mistakes.

Proper style is important, and required in most cases. The exceptions are: (a) the first three questions I mostly ignore style, and (b) on the very last final exam I mostly ignore style.

Testing is important. If I receive a program that was clearly not tested, it will be marked wrong and not given a second chance.

Contents

In this chapter, we consider each exam question. We identify the key things you need to demonstrate. We mention the common mistakes that people make. Before attempting a problem (either for the first time or a subsequent time), you might benefit from reviewing the section that talks about that problem.

GradeBot has some exercises that are similar to the exam questions. They are listed as "GradeBot Examples". They might provide useful practice as you prepare to take an exam.

In the case of GradeBot, only behavior and results are measured. GradeBot does not examine your code to see how you achieved your result.

In the case of the exam itself, the human grader does review your code to make sure you achieved your result in the required manner.

3.1   q1: String Basic

GradeBot Examples: g21.

These points are earned during a final exam (or early final) by writing a working and correct program that does what is required.

The key things to demonstrate here are:

(a) How to get string input into your program. This is done by reading from `<STDIN>` and storing the result in a variable.

Example: `$flavor = <STDIN>;`

(b) How to remove the newline from the end of the string. This is done by using the `chomp` command.

Example: `chomp ( $flavor );`

(a) and (b) are often combined into a single statement.

Example: `chomp ( $flavor = <STDIN> );`

(c) How to compose a printed statement that includes information from your variables. This is done by using the variable name within another string.

Example: `print "I love $flavor ice cream."`

(d) Do exactly what was requested. If I request specific wording, you must follow it exactly. If I do not specify something exactly, you are free to do anything that works.

Example: `print "I love $flavor ice cream. "`

In this example, there is a space after ice cream. If my specification says there should be no space, then by putting a space you will lose credit for your work.

3.2 q2: Number Basic

GradeBot Examples: g30 series.

These points are earned during a final exam (or early final) by writing a working and correct program that does what is required.

You should already have the skills involved with String Basic.

The key thing to demonstrate here is:

(a) How to use simple arithmetic to calculate an answer.

Example: `$x = 2 * $y - 5;`

You will be told specifically what to do. For example, read in two numbers, multiply them together, and then add 5.

Parentheses may be useful in getting formulas to do the right thing.

Note: it is usually not necessary to `chomp` inputs that are numbers. Perl will still understand the number fine.

3.3   q3: Number Story

GradeBot Examples: g30 series.

These points are earned during a final exam (or early final) by writing a working and correct program that does what is required.

You should already have the skills involved with Number Basic.

Story problems are problems where the precise steps are not given to you. Instead, you must understand the problem and develop your own formula. Sometimes this is easy. Sometimes this is difficult.

The main thing we are measuring is whether you can invent your own formula based on the description of the problem.

Remember to test your program. Make sure your formula gives correct answers.

3.4   q4: Number Decision

GradeBot Examples: g40 series.

These points are earned during a final exam (or early final) by writing a working and correct program that does what is required.

The emphasis here is on decision. How do you decide what to do? How do you express your desires?

The key things to demonstrate here are:

(a) How to write an `if` statement.

(b) How to compare two numbers. This includes:

(b1) Example: (`$x < $y`) means less than.

(b2) Example: (`$x <= $y`) means less than or equal to.

(b3) Example: (`$x == $y`) means equal to.

(b4) Example: (`$x > $y`) means greater than.

(b5) Example: (`$x >= $y`) means greater than or equal to.

(b6) Example: (`$x != $y`) means not equal to.

(c) Near Misses. Things that look right but are wrong.

(c1) Example: (`$x = $y`) is a frequent typo for equal to, but actually means "gets a copy of".

(c2) Example: (`$x => $y`) is a frequent typo for greater than or equal to, but means the same thing as comma does when defining an array.

3.5   q5: Number Decision Story

GradeBot Examples: g40 series.

These points are earned during a final exam (or early final) by writing a working and correct program that does what is required.

As with number story, we have a story problem. And a decision will be involved. You will need to analyze the question and decide how to solve it.

3.6   q6: String Decision

GradeBot Examples: g50 series.

These points are earned during a final exam (or early final) by writing a working and correct program that does what is required.

The emphasis here is on strings and how their decisions differ from numbers.

The key things to demonstrate here are:

(a) How to compare two strings. This includes:

(a1) Example: (`$x lt $y`) means less than.

(a2) Example: (`$x le $y`) means less than or equal to.

(a3) Example: (`$x eq $y`) means equal to.

(a4) Example: (`$x gt $y`) means greater than.

(a5) Example: (`$x ge $y`) means greater than or equal to.

(a6) Example: (`$x ne $y`) means not equal to.

(b) Near Misses. Things that look right but are wrong.

(b1) Example: (`$x eg $y`) is a frequent typo for eq.

(c) Properly quote your literal strings. (See barewords in the textbook.)

(c1) (`$x eq "hello"`) is the right way to quote a string.

(c2) (`$x eq hello`) is the wrong way to quote a string.

(c3) (`$x eq $y`) is right because it is a variable, not a literal.

3.7   q7: String Bracket

GradeBot Examples: g50 series.

These points are earned during a final exam (or early final) by writing a working and correct program that does what is required.

The emphasis here is on more complicated decisions, where there are more than two options.

The key things to demonstrate here are:

(a) How to handle "clarinet through costly".

(b) How to handle "a-j, k-o, p-z".

(c) How (and when) to handle all possible capitalizations. What does "dictionary order" mean?

(d) Properly quote your literal strings. (See barewords in the textbook.)

(d1) (`$x eq "hello"`) is the right way to quote a string.

(d2) (`$x eq hello`) is the wrong way to quote a string.

(d3) (`$x eq $y`) is right because it is a variable, not a literal.

3.8   q8: Repeat While

GradeBot Examples: g60 series.

These points are earned during a final exam (or early final) by writing a working and correct program that does what is required.

Loops are an important tool. Repeat While names a specific instance of that.

The syntax is `while ( condition ) { block }`

In these loops the condition is just like `if` statements have. Often it is a comparison like (`$x < 100`) .

The block is the collection of commands that will be done repeatedly, so long as the condition is still true.

Common error: make sure the condition will eventually become false. If your condition checks for `$x` less than 100, make sure that `$x` is changing and will eventually reach 100.

Common error: if the ending condition gets skipped, the loop could run forever. (`$x < 100`) is much safer than (`$x != 100`) .

Common error: confusing the `while` syntax with the `for` syntax.

3.9   q9: Repeat For

GradeBot Examples: g60 series.

These points are earned during a final exam (or early final) by writing a working and correct program that does what is required.

Loops are an important tool. Repeat For names a specific instance of that.

The syntax is `for ( init; condition; step ) { block }`

The `init` part initializes the variable that controls the loop.

The `condition` part is just like an `if` statement or `while` statement.

The `step` part is usually something like `$x++` that increments the control variable.

Common error: confusing the `while` syntax with the `for` syntax.

3.10   q10: Repeat Last

GradeBot Examples: g60 series.

These points are earned during a final exam (or early final) by writing a working and correct program that does what is required.

Loops are an important tool. Repeat Last names a specific instance of that.

The syntax is `while ( 1 ) { block }` where the block includes something like this to break out of the loop:

`if ( condition ) { last }`

Common error: due to style requirements, `last` should be on a new line, properly indented.

3.11   q11: Repeat Nested

GradeBot Examples: g60 series.

These points are earned during a final exam (or early final) by writing a working and correct program that does what is required.

Loops are an important tool. Repeat Nested names a specific instance of that.

What we are looking for here is the ability to run one loop (the inner loop) inside another loop (the outer loop).

Example: print all possible combinations for a child's bike lock, where there are four wheels each ranging from 1 to 6.

3.12   q12: List Basic

GradeBot Examples: g70 series.

These points are earned during a final exam (or early final) by writing a working and correct program that does what is required.

Lists and arrays are the same thing. When we talk about lists, we are not using indexing. When we talk about arrays, we are using indexing.

You will have to know that shift and unshift affect the front (start) of the list, and push and pop affect the back (end) of the list.

The key things to demonstrate here are:

(a) An array can be initialized by listing elements in parentheses.

Example: `@x = ( "cat", "dog", "bird" );`

(b) An array can be modified.

(b1) using `push` to add something to the **back (end)** of a list.

Example: `push @x, "hello";`

(b2) using `pop` to remove something from the **back (end)** of a list.

Example: `$x = pop @x;`

(b3) using `shift` to remove something from the **front (start)** of a list.

Example: `$x = shift @x;`

(b4) using `unshift` to add something to the **front (start)** of a list.

Example: `unshift @x, "hello";`

3.13 q13: List Loop

GradeBot Examples: g70 series.

These points are earned during a final exam (or early final) by writing a working and correct program that does what is required.

The key thing to demonstrate here is how to use a foreach loop.

Example: `foreach $book ( @books ) { print $book }`

Example: `foreach ( @books ) { print $_ }`

Wrong: `foreach @books { print $_ }`

3.14 q14: Array Basic

GradeBot Examples: g70 series.

These points are earned during a final exam (or early final) by writing a working and correct program that does what is required.

Lists and arrays are the same thing. When we talk about lists, we are not using indexing. When we talk about arrays, we are using indexing.

(a) The whole array is named with `@` at the front.

(b) Individual slots in the array are named with `$` at the front, and `[number]` at the back.

(c) The first item in an array is at location zero.

Example: `$x = $array[0];`

Example: `$array[0] = $x;`

(d) The second item in an array is at location one.

Example: `$x = $array[1];`

(e) The last item in an array is at location -1.

Example: `$x = $array[-1];`

(f) The second to last item in an array is at location -2.

Example: `$x = $array[-2];`

Ambiguous: `@x[1]` - Perl accepts it for `$x[1]` but I do not.

3.15   q15: Array Loop

GradeBot Examples: g70 series.

These points are earned during a final exam (or early final) by writing a working and correct program that does what is required.

The key thing to demonstrate here is how to use a for loop.

(a) The size of an array can be found out.

Example: `$size = @array;`

(b) A `for` loop can be used to "index" your way through an array.

Okay: `for ( $i = 0; $i < $size; $i++ ) { print $array[$i] }`

Wrong: `for ( $i = 0; $i <= $size; $i++ ) { print $array[$i] }`

Okay: `for ( $i = 0; $i < @array; $i++ ) { print $array[$i] }`

3.16   q16: Array Split

GradeBot Examples: g70 series, specifically g74, g75, g76, g77, g78.

These points are earned during a final exam (or early final) by writing a working and correct program that does what is required.

The `split` command can be used to convert a string into an array.

Example: `@x = split ":", "11:53:28";`

Common mistake: `$x = split` ... (because dollar-x should be at-x)

3.17   q17: Array Join

GradeBot Examples: g70 series.

These points are earned during a final exam (or early final) by writing a working and correct program that does what is required.

The `join` command can be used to convert an array into a string.

Example: `$x = join ":", ( "11", "53", "28" );`

Common mistake: `@x = join ...` (because at-x should be dollar-x)

3.18   Subroutine Basics

All subroutine points require you to do the basic elements of each subroutine correctly.

Subroutines are defined using the following syntax:

`sub name { block }`

The word `sub` must be given first. It is not `Sub` or `subroutine` or forgotten.

Never use global variables unless they are necessary. That means each variable in a subroutine should be introduced with the word `my` the first time it appears, unless you are sure it is supposed to be global.

Exception: `@_` in a subroutine is naturally local. You don't have to `my` it.

3.19   Subroutine Verification

You may be asked to write a main program that correctly calls the subroutine(s) with several sets of inputs and verifies correct operation.

What does this mean?

In the real world, it is common for teams of programmers to work together on large programs. Each piece is written as a subroutine and then the subroutines are combined to make the finished program.

Verification is the process of running a subroutine that may have been written by somebody else and making sure it gives good answers back. It is generally impossible to test every set of inputs but it is a good idea to test

several sets of inputs, including boundary conditions if there are any, to make sure the subroutine does the right thing.

For example, let's say we are testing a subroutine named max that is supposed to return the largest number from a list. We need to verify that it is working. We could use lines like the following:

```
if ( max(1,2,3,4) != 4 ) { print "error 1\n" }
if ( max(2,4,3) != 4 ) { print "error 2\n" }
if ( max(1,2,-4,-3) != 2 ) { print "error 3\n" }
if ( max(-2,-4,-3) != -2 ) { print "error 4\n" }
```

If the subroutine is supposed to update a global variable, check to see if it was updated. If a subroutine is supposed to return a value, check to see that the correct value is returned.

Several means at least two. There should be several tests, if possible, not just one, and they should test a variety of situations. Do we know that it can find the max if it is first? What if it is last? What if it is in the middle? What if the max is negative? What if there are no inputs? What if there are lots of inputs?

3.20 q18: Subroutine Return

These points are earned during a final exam (or early final) by writing a working and correct program that does what is required.

(a) To return a single number from a subroutine, you can do it like this.

Example: `return 5;`

Example: `return $x;`

Wrong: `return ( 5 );` - this is an ambiguity error.

(b) To return a string from a subroutine, you can do it like this.

Example: `return "this is a string";`

Wrong: `return ( "this is a string" );` - ambiguity.

(c) To return an array from a subroutine, you can do it like this.

Example: `return ( 1, 2, 4, 8 );`

Wrong: `return "( 1, 2, 4, 8 )";` - a string is not an array

Example: `return ( "this", "is", "a", "list" );`

Wrong: `return ( this, is, a, list );` - each string should be quoted

Example: `return @x;`

Wrong: `return "@x";` - a string is not an array

(d) `return` and `print` do different things. Return gives something back to the caller. Print sends something to the end user.

Also, if the requirement is to return `"aloha"`, make sure that is what you return, and not `"aloha\n"`. The adding of `\n` to things we are printing is something that I do not grade more or less for, but printing is not the same as returning.

3.21   q19: Positional Parameter

These points are earned during a final exam (or early final) by writing a working and correct program that does what is required.

We are testing your ability to retrieve parameters that were passed into a subroutine.

The arguments to a subroutine arrive in the local variable `@_` and can be retrieved from it.

Positional parameters are always in the same slot of the array. You can get the third positional parameter by using `$_[2]` for example.

3.22   q20: Globals and Locals

These points are earned during a final exam (or early final) by writing a working and correct program that does what is required.

We are testing your ability to maintain privacy on the variables you use in your subroutine.

In Perl, variables are naturally global. This is now widely recognized to be a bad thing, but it is too late to change now.

To force variables to be local (which is the opposite of global), you have to

specially mention the word `my` before the variable the first time it is used.

Example: `my $abc;` - creates a local variable with `$abc` as its name.

Example: `my ( $abc );` - creates a local variable with `$abc` as its name.

Example: `my ( $abc, $def, $ghi );` - creates three local variables named `$abc`, `$def`, `$ghi`, respectively.

Common Error: `my $abc, $def, $ghi;` - creates ONE local variable named `$abc`, and mentions two global variables named `$def` and `$ghi`.

3.23   q21: Variable Number of Parameters

These points are earned during a final exam (or early final) by writing a working and correct program that does what is required.

We are testing your ability to retrieve parameters that were passed into a subroutine.

The arguments to a subroutine arrive in the local variable `@_` and can be retrieved from it.

A `foreach` loop is usually used to walk through the list of parameters that were sent to the subroutine.

Appendix A

Help! (Problem Solving)

Sometimes things will not work. But all is not lost. This chapter has ideas for understanding and solving the kinds of problems we often encounter in this course.

Contents

A.1 Technical Support

The CIS department has a systems administrator to help students and faculty. GCB 119 is the best place to go for technical assistance.

Your teacher is the next best place to go for technical assistance.

A.2   cPanel Login Problems

If you are trying to use cPanel and have trouble logging in, you need technical support.

Our tech support people operate the IS2 machine where our cPanel accounts are stored.

Some of the faculty have the ability to reset passwords and make other fixes.

A.3   Programming Problems

When you run your program from the command line, it will either work or not. If it does not work, it could be because of a syntax error, or it could be because of a logic error.

Syntax errors are mistakes in the wording of how you said something. The most common syntax error is the missing semi-colon. Each statement should end with a semi-colon. If it is missing, then the program cannot be understood by the computer, and it will fail.

The computer will generally give you an error message that tells what line it got up to before it knew it had an error. Often this is the line right after the actual error. If there is an error on line 10, the computer may not notice it until it is working on line 11. Then it will report a problem on line 11, even though the actual problem is on line 10.

Logic errors (also called runtime errors or semantic errors) are mistakes in what you were asking the computer to do. Maybe you forgot to initialize a variable. Maybe you did two steps in the wrong order. Maybe you left something out.

One of the best solutions for finding and fixing logic errors is the print statement. Print out information as your program runs, things like "I got to line 12" or "the value of $i is 15". This is called debug information. It can be helpful for seeing what is going on. Compare that with what you expected. Usually that helps narrow down the mistake.

A.4   Webpage Problems

Working with an online program adds one more layer of possible confusion. Instead of going directly between your program and your user, you have a browser in the middle.

A.4.1   Webpage Does Not Load

If you are trying to load a webpage that I mention in this study guide, and the webpage does not load, it could be a DNS (domain name system) problem. In any case, the people to see are our technical support people. They run our department DNS system.

In the past this has sometimes been a problem for students living on campus, because of the special way that the CIS department is "sandboxed" to protect the rest of the university from the weird things we occasionally do. But Tech Support probably has solutions.

A.4.2   404 Page Not Found Error

404 is the error number used by the world wide web to indicate a missing webpage. If you get a **404 error** when trying to see your own webpage, it means the browser asked for the page and the server said it does not exist.

The most common cause of this problem is spelling parts of the path differently than expected.

If your webpage should be named "index.html" and you actually name it "Index.html" it will not be found. Pay attention to capitalization.

If your folder should be named "myproject" and you name it "my project" (with a space) it will not be found.

I have even seen problems when the student accidentally put a space at the end of a file name, like "index.html ".

A.4.3   500 Internal Server Error

The **500 error** means that the server found your program, and your program tried to run, but that it did not return a usable webpage.

(1) The first thing to try is this. Copy your whole web program and try running it from the command line. If there are any error messages, you can see what line is causing the first problem. Fix the first problem and then run your program again. (If there is a second problem, often it was caused by the first problem, so don't worry about it unless it is still there after fixing the first problem.)

(2) If that all looks good, then check the first few lines of program output. The first line should be "content-type: text/html". It must be spelled exactly right, no extra spaces, no blank lines in front. It must be followed by a blank line.

(3) One other common problem is file permissions. Make sure your index.cgi program has permissions of **0755** so it can be executed by the server.

A.4.4   Download Request

If you spell "content-type: text/html" incorrectly, the browser will not know that you are sending a webpage, and may ask if you would like to download the content. If you get a request like that, say "no" and then check your content-type line carefully.

A.4.5   Blank or Incomplete Webpage

Your program may run but create a webpage that is incomplete. For example, maybe it has a heading but nothing after that.

Use the "show page source" command in your browser. Sometimes that will uncover problems such as missing tag endings, like if you said `</h1` when you meant to say `</h1>`.

Appendix B

DCQuiz: My Learning Management System

Contents

I developed my own learning management system (LMS) which we will use for this course. Other LMS examples include BlackBoard, Canvas, and Moodle. (I did not write them.) I currently do not use them.

https://dcquiz.byuh.edu/ is the DCQuiz URL.

Since I wrote it myself, I am also responsible for any bugs that may be in its programming. If you notice any bugs, I hope you will let me know so I can get them fixed.

I can also make improvements when I think of them. I like that.

B.1 Grade Book

The most important place you will see DCQuiz is the grade book.

I use DCQuiz to manage my grade book for this class. You will be able to see the categories in which points are earned, and how many points are credited to you.

You will also be able to see how many points are credited to other students, but you will not be able to see which students they are.

This gives you the ability to see where you stand in the class, on a category-by-category basis, and in terms of overall points. Are you the top student? Are you the bottom? Are you comfortable with your standing?

B.2 Daily Update

Another place you are likely to see DCQuiz is the daily update.

Typically in class I start with a quiz called the Daily Update. It usually runs the first five minutes of class, and is followed by the opening prayer.

By having you log in and take the daily update quiz, I also get to see who is in class, in case I need a roll sheet and I did not take roll in some other way.

I also give you an opportunity to make an anonymous comment. This can be anything you want to say. It might include announcements, such as birthdays or concerts. It might include questions. It can be a simple greeting.

Comments provide a chance for each student to say something without the embarrassment of everyone else knowing who said it. You can say how unfair you think I am for something. You can ask about something you find confusing.

I introduce it something like this:

If you wish, you can type in a comment, question, announcement, or other statement at the start of class for us to consider. Or you can leave this blank.

This is a good opportunity to ask about something you find confusing.

The identity of the questioner (you) will not be disclosed to the class, and normally I will not check (although I could). My goal is for this to be

anonymous.

B.3   Exams

DCQuiz was originally developed for giving tests. My problem was hand-writing, actually. Students would take tests on paper and sometimes I could not read what they had written.

So I cobbled together an early version of DCQuiz to present the questions and collect the answers.

I got a couple of additional wonderful benefits, almost immediately.

First, I got the ability to grade students anonymously. All I was seeing was their answer. Not their handwriting. Not the color of their ink. Not their name at the top of the paper. It was wonderful. I could grade things without so much worry about whether every student was being treated fairly.

Second, I got the ability to share my grading results with every student in the class. Each student can see, not only the scores earned by other students, but the actual answers that other students put to each question. This gives students the ability to learn from each other.

Third, it gave students a way to verify that they were being graded fairly compared to their fellow students. If you can see your own answer, and see that everyone with higher points gave a better answer, that is a good thing. If you think your answer is better, it gives you a reason to come and see the teacher so you can argue for more points, or you can be taught the reasons for their answers getting more points.

Fourth, it gave me the convenience of grading anywhere without carrying a stack of papers. I could grade on vacation. (Wait. Doesn't that make it a not-vacation?) I could grade in class, or in my office, or at home.

Fifth, although I never did this, it theoretically has the ability for me to let other people be graders. But I never did this.

B.3.1   Taking Exams

As it currently operates, DCQuiz lets you, the student, log in and see a list of quizzes. (The grade book is actually just another quiz, but it is one where I enter grades that you earned some other way.)

Quizzes typically have starting and ending times. Before the quiz starts, there is a note telling when it will start. As the quiz gets closer, like within an hour or two, an actual count-down clock will appear telling you how long until the quiz is available.

Once you start the quiz, if it has an ending time, you will be able to see a count-down timer telling you how much time you have left.

As you take a quiz, you can see the main menu, the question menu, and the question page.

Main Menu: The main menu was already mentioned. That's where you see what quizzes are available.

Question Menu: The question menu shows you what questions are on this quiz. It lets you select a question to work on. It shows you which questions you have answered already. It shows you which answers have already been scored. It lets you say that you are done. It lets you cancel the quiz (if that is allowed).

Question Page: The question page shows a single question, and lets you type in your answer. Some questions only allow a single-line answer. When you press ENTER it takes you automatically to the next question. Other questions let you type in several lines.

Early Grading: The question page may allow an option for early grading. If you think you have given your final answer, you can submit it for early grading. If I have time, I will grade it while the test is still under way. That could give you confidence to answer related questions, knowing that you got something right.

Throwbacks: Along with early grading, I sometimes give you a second chance by doing something that I call a "throwback." That is when I look at your answer, and I think it is very close, but maybe you missed something. If so, I may unsubmit it for you. Then it will show up on your list of questions again. You can look at it and read the question again, and maybe realize what it was that you had not noticed before.

Good-Faith Testing: To get a throwback, you must have made a good-faith effort to test your program. If I see a syntax error, then I know you did not test your program, and I will not give a throwback. If I see an obvious math error, I will know that your testing was insufficient, and I will not give a throwback.

If I see an error that I think is due to overlooking some detail of the problem,

then I will give a throwback. I hope you will read the question again and notice the problem.

B.3.2 Reviewing Exams

When an exam is finished, DCQuiz lets me, the author of the exam, share it with you, the student who took the exam.

You can see reviewing opportunities on the main menu.

After selecting an exam to review, you will see a question menu similar to the one that was used for taking the exam. But instead of seeing your answer, you will see all the scores that were earned, with your score highlighted. If yours is the top score, it will appear first. If it is the bottom score, it will appear last.

You can select a question to drill down and see more details. Specifically, you can see each of the answers provided by each student that wrote an answer. And you can see the score it received. And you can see any notes the grader (me) may have made while grading.

This is intended to (a) let you teach yourself by seeing examples of work by other students, and (b) let you verify that you were graded fairly. (Every once in a while, maybe a few times per semester, a student will see that I entered their grade wrong, or I overlooked something. This is your chance to get errors fixed.)

Sometimes an exam is not open for review. The teacher gets to decide. But even if the exam is closed, you can still see the question menu (with the questions blanked out), and you can see your score and everyone else's score. Questions and answers are not available, but scores are available, even long after you took the test.

Sometimes an exam is deleted or revised and reused. The teacher gets to decide. When an exam is deleted, all questions and answers and scores are also deleted. After that, there is no way to see anything about that exam.

I generally revise and reuse the daily update exams. This causes all answers and scores to be deleted, but I keep the questions and just modify them for the next class meeting.

B.4 Requesting a Regrade

If dcquiz is involved with the grading of an item, either as part of a test or as part of a grade book, it also provides you with the best way to ask for a regrade.

Dcquiz provides a Regrade button on the review screen for each problem.

When should you ask for a regrade?

Maybe I did the initial grading wrong because I was in a hurry and I just did not carefully understand what you said in your answer. If you have checked your work and you are pretty sure it is right, then press the button for Regrade and tell me why you think your answer is right. I will review your work and respond to you.

Another situation is when you wrote an online program and it was originally wrong but you have fixed it. Press the Regrade button and tell me what you have fixed. I will review your work and respond to you.

If dcquiz is not involved, so there is no Regrade button to press, you can still request a regrade by sending me an email. The required subject line is: `cis101 roll-book-name` The body of your email should explain your thinking.

B.5 Other Features

DCQuiz has other features, such as the ability to limit where a test is taken, or to require a special code to access a test. Those features will be explained in class if they are ever needed.

Appendix C

Activities General Information

Contents

We assume you are studying outside of class time, and that the textbook that I provide contains enough background information to avoid lots of lecturing in class.

My intention is that we will do in-class activities many times through the semester.

Chapter 2 (page 36), the Assignments chapter, will be updated as new activities are assigned. Check there for activity details.

This current chapter explains the general rules that apply to each of the types of activities that will be assigned.

Grading Label means a short label I use to track this activity for grading purposes. Online activities have labels that start with o. GradeBot activities have labels that start with g. Command-line activities have labels that start with c.

Grades will be posted to the "2161 CIS 101 Activities" grade book in the column specified by the grading label.

C.1   oXX: Online General Rules

Online tasks generally follow these rules. Exceptions and clarifications are provided as needed for each task.

Label: Each online task has a grading label consisting of the letter "o" (for online) followed by (normally) one or two other characters that specify which online task it is.

Task: Create a webpage (index.html) or CGI program (index.cgi) that is properly linked to the CIS 101 student projects page. It must clearly display your name right on the webpage (not just in the title). Other requirements vary by task.

http://dc.is2.byuh.edu/cis101.2161/ is the student projects page URL.

C.1.1   How To Submit

Create a webpage properly linked to the student projects page. I normally grade everyone's submissions at once, both at the 11pt (extra credit) deadline and at the 10pt (full credit) deadline.

Late Work: If you complete your work late, tell me via email so I will know to grade it. Follow the email rules in section C.4 (page 141) in the construction of your subject line.

C.1.2   Online Requirements and Suggestions

Web design classes get deeply into style, also called CSS. They also cover HTML, which is the markup of the material on the webpage.

It turns out that HTML is actually very easy and can be covered relatively quickly. In this section I provide a bit of training to help you write valid HTML, but this is not a web design course, so it is not required of you.

Webpages consist of four main parts: content, markup, styling, and scripts. In this class we will worry only about the content and markup.

Content is what the user will see. It is typed directly into the webpage.

Markup is always specified like `<tag ...>` where tag is one of the special words that make up the HTML language. The opening and closing angle-brackets show that this is an HTML tag. Here are some tags you will use: !DOCTYPE, head, meta, title, style, body, h1, p, and img.

End tags may be required, optional, or forbidden.

Required: `<h1>` in particular requires a matching `</h1>`. An ending tag is also required for `<b>` (bold), `<i>` (italic), `<u>` (underline), `<div>` (division), `<span>` (span), and `<form>` (form).

Optional: `<head>`, `<body>`, `<p>` (paragraph), and `<li>` (list item) fall into this category.

Forbidden: `<img>` and `<br>` and `<input>` do not have separate end tags.

1. Suggested: **DOCTYPE:** You should have a proper DOCTYPE tag. It will be the first tag in your webpage. For html5 it would look like this example:

   Example: `<!DOCTYPE html>`

2. Suggested: **head:** You should have a proper head tag. It will be the second tag in your webpage.

   Example: `<head lang=en>`

3. Suggested: **charset:** You should have a proper meta charset tag. It tells the browser what character set you are using. Unless you have a special need to do something else, you should use this meta charset tag. Normally this will be your third tag.

   Example: `<meta charset=utf-8>`

4. Suggested: **title:** You should have a proper title that includes your name and describes the task. Titles are up to about 50 characters long.

   Example: `<title>Don's Page About Whatever</title>`

5. Suggested: **description:** You can also have a meta description tag. Descriptions are about two lines long, typically. They are used by search engines to describe your webpage. They are not the same as your title.

   If you have one, it must accurately describe your webpage.

   Example: `<meta name=description content="Don's Page About Whatever">`

6. Suggested: **style:** You can have a proper style section, even if it is empty. Between these lines you would put any styling you will be doing. This would appear in your head section, before body.

   Example starting line: `<style>`

   Example ending line: `</style>`

   Style is often used to provide a background color or background image. It also lets you do centering and make borders. In this course you are not ever required to use style, but it is encouraged and I would be happy to tell you how to do various things.

   If you do not plan to use the style section, you can have a single line like this (or just leave it out):

   `<style></style>`

7. Suggested: **/head:** It is recommended to explicitly end the head section before starting the body (content) section.

   Example ending line: `</head>`

8. Suggested: **body:** You should have an explicit body tag (and an explicit /body tag). The body tag will come immediately after your `</head>` tag, if you have one.

   Example: `<body>`

9. Suggested: **h1:** You should have a proper h1 tag that identifies yourself and your project. Typically this is similar to or identical to your title. It must be ended with a `</h1>` tag.

   Example: `<h1>Don's Project About Whatever</h1>`

10. Required: **Acknowledge Professional Content:** It is best to simply avoid using professional-looking content unless it is your own. If you choose to include any professional-looking content, including for example images or text that you did not personally create, you must

also include a brief statement giving credit to the source. **This is true even if you are the source.** If I think it looks professional, then it needs attribution and credit. (Take it as a compliment.) You can put your acknowledgement statement at the bottom of your page.

11. Suggested: **/body:** You should explicitly end the body section.

    Example ending line: `</body>`

C.1.3   Online Template

Here is a template you can copy and modify. It includes all the recommended elements of your webpage.

```
<!DOCTYPE html><head lang=en><meta charset=utf-8>
<title>required title goes here</title>
<meta name=description content="optional description">
<style></style>
</head><body>
<h1>required heading goes here</h1>
put some content here
</body>
```

C.2   cXX: Command-Line General Rules

Command-line tasks generally follow these rules. Exceptions and clarifications are provided for each task.

Label: Each command-line task has a grading label consisting of the letter "c" (for command-line) followed by (normally) one or two other characters that specify which online task it is.

Task: Write a program. Requirements vary by task.

Follow the email rules in section C.4 (page 141) as you construct your subject line and the body of your message.

If your submission was not acceptable, I will reply to it giving at least one reason that it was not acceptable. (There may be other problems that I did not notice or mention.) Normally you should fix the problem and resubmit your program.

If your submission was acceptable, I will reply to it with the word "Success" and the number of points earned. Sometimes I include additional comments. You may want to save my reply until you see your grade reflected in my grade book.

C.3   gXX: GradeBot Task General Rules

GradeBot is an automatic program grader. You write the program. Grade-Bot runs your program and gives it sample problems to solve. Basically, GradeBot tests to see whether your program behaves properly and gets the right answers.

GradeBot itself is explained in appendix D (page 144).

This section explains how and what to submit for GradeBot assignments.

0. Test Drive: Anyone can try any of the GradeBot activities at any time. You even have a choice of programming languages. This can be a fun way to develop some skills even when an activity has not been assigned to you.

1. Customization: To get credit in this class for your GradeBot work, you must begin your program with the required comment. It identifies the course, the assignment, and the student. It is used by GradeBot to customize your task in small ways. If you change the comment, it may change the details of your task.

`# cis101 gxx roll-book-name` is the required comment line.

Your roll-book-name is your lastname, then a comma, then your firstname. Lastname means your own last name (family name), and firstname means your own first name (given or personal name), as shown in my grade book.

If customizations are in effect, the background color of the GradeBot screen will turn **light green.** This will be true for all assigned tasks.

If customizations are not in effect, the background color of the GradeBot screen will be **light yellow.**

2. Understand the Task: Sometimes there are specific rules to follow. These are given in the task assignment and may include things like what algorithm to use or what styling is required.

3. Test Your Program: With the required comment in place, write and test your program. Make sure you get this message:

```
Success!  No errors found.
```

Success means that GradeBot thinks your program was successful (it produced the right outputs). There may still be specific things that are too difficult for GradeBot to check, including your choice of algorithm or some aspects of style.

4. Getting Credit: If the customization line is recognized, GradeBot will also invite you to provide your email address. This is required for credit and it lets me reply to you.

```
Email address not found, so success report was not sent.
```

If this line appears at the top of your screen, it means that (a) your program does what it should, but (b) you have not provided an email address yet. Fill in the blank provided for your email address and test your program again.

```
Success report sent ...
```

If this line appears at the top of your screen, it means that GradeBot has notified me that it thinks your program was successful.

GradeBot will also send a copy of your program to you at the same time. IF YOU DO NOT RECEIVE AN EMAIL, then I probably did not receive one either, and I will not be able to give you credit for your work.

5. My Reply: When I receive the success report, it goes into a queue until I can check it personally.

When I agree that your program meets the requirements, I will update the grade book. I will reply to you so you know how many points you received.

As mentioned above, there may be specific things that would be too difficult for GradeBot to check, including your choice of algorithm or some aspects of style.

If I do not agree that your program meets the requirements, I will reply to you and tell you what was wrong. Then you can fix your program and submit it again.

6. Bonus Point: Full credit for most GradeBot activities is 10 points. I also publish an 11pt deadline. If you submit a correct program by the 11pt deadline, and if I consider your program to be good enough, you will receive 11 points.

C.4 Email Submission Rules

In some cases, I require you to submit your work by email. When email is involved, there are a few rules I need you to follow.

If your program violates the rules enough that grading becomes difficult, I will probably reply to your submission telling you what rules you violated and asking you to fix and resubmit.

C.4.1 To: Line

Please send email to `<doncolton2@gmail.com>`. That is my preferred email address.

If you really cannot use that, you are welcome to email to `<don.colton@byuh.edu>`. They both ultimately go the same place, so you do not need to send to both. Either one is fine.

C.4.2 Subject Line

The subject line of the email must be as follows:

`cis101 label roll-book-name`

The reason for this rule is to facilitate the recording of grades. When I receive your email, it may be in the midst of many other emails from other students. I need to keep things straight so that I can record your grade properly.

The `cis101` part must be written exactly as shown. It is a magic word that lets the email system identify your message as something that belongs to this class. It keeps it out of my spam folder. Warning: Do not add a space in the middle of it, for example, or your email will not go to the right place.

The `label` part is replaced by the grading label for that assignment.

Your roll-book-name is your lastname, comma, your firstname. Lastname means your own last name (family name), and firstname means your own first name (given or personal name), as shown in my grade book. This is the name that you asked me to use for you. I use that name in my grade book.

When I go to record your grade, I scan down my grade book, which is sorted

by lastname and firstname. If I do not see the lastname and firstname that you provided, it requires extra steps for me to verify which person should receive credit. I would prefer to have you do those extra steps instead of me.

So, for example, if I were submitting task p1 and my lastname were Colton and my firstname were Don, I would use this subject line:

`cis101 p1 Colton, Don`

C.4.3 Instant Response

When you have properly submitted an email message to me (correct email address, and correct magic word "`cis101`" in the subject line), you will receive an automatic reply almost immediately. This proves that I received your message. If you do not receive such a reply, it may mean that I did not receive your message.

Here is a typical reply:

> This is an instant reply to let you know that I received your email relating to CIS 101 (Beginning Programming), and it has been placed into my pending file for review and/or grading as soon as I can get to it. You can expect another reply from me when I have looked at your email. Thanks! Bro Colton

If you want your email treated as a formal communication to me, look for the instant response.

C.4.4 Body When Submitting a Program

If you are submitting a program, the remainder of the email should be that program, and nothing else unless the assignment specifically requires it.

Do not send your program as an attachment. Send it directly in-line as plain text so I will see it immediately when I open your email.

The first line of your program must be a comment line that repeats the required subject line. This is to facilitate the recording of grades.

In Perl, `#` is used to start a comment line. So, following the previous example from above, I would have this comment line as the first line of my program.

```
# cis101 p1 Colton, Don
```

The email must be in plain-text form. It should not be in html form or **rich text** form or in the form of an attachment. In plain text, there is no coloring to the letters. There is no bold or italics.

The reason for this rule is to facilitate testing of your program. When I receive your email, I may need to test it by running it. I do this by doing a copy-paste from your email into GradeBot (for instance) or into an empty program file. Then I run your program.

I should be able to use copy-paste to make a copy of your program for testing.

Also, please make it easy for me to tell where your program begins and ends. If you have anything else in your email, make sure it is clearly separated from your program.

I do not accept programs sent as attachments because of the extra work it requires on my end.

You must avoid having each line of your program start with `>` or `>>` as is common when you are replying. Having those characters makes it impossible to copy-paste and run your program.

If your program includes any long comments, make sure each line of the long comment starts with the `#` character. Otherwise, the program will not run. I mention this because your email client may automatically break up long lines, thus converting your correct and working program into an incorrect and broken program. Be alert to this possibility and protect against it by not using long lines.

Appendix D

GradeBot

GradeBot is my automated program grader.

Contents

I normally grade programming activities based on the following three criteria.

Behavior: How does the program respond to various situations? This is always important.

Style: How clearly is the program written? My standards vary from course to course and from assignment to assignment.

Algorithm: What algorithms are used? Were they efficient? My standards vary from course to course and from assignment to assignment.

When I grade on algorithm, I rely on visual inspection of the program source code.

When I grade on style, I rely on a combination of GradeBot and visual inspection of the program source code.

When I grade on behavior, I usually rely on GradeBot. GradeBot verifies that your program seems to be running correctly by giving your program test cases to solve, and then seeing whether your program returns the correct answer each time.

(I have used GradeBot for many years. This particular version of GradeBot is sometimes called GradeBot Lite because it is descended from what used to be a huge system that was also called GradeBot.)

D.1 Where Can I Find GradeBot?

http://gradebot.tk/ is the web interface for GradeBot.

http://gbot.is2.byuh.edu/ is an alternate URL that you can use in case the short URL stops working or does not work for you.

If you want to explore, press the [List All Labs] button. Then pick a lab from the list and press its button.

GradeBot will give you a "SAMPLE EXECUTION" to show the behavior it is expecting from your program.

D.2 Source Code File

To keep things simple, GradeBot requires you to submit a single file of source code. You are not permitted to write modules as separate files, and compile them separately and then link the results. Such abilities are present in Integrated Development Environments, but not in GradeBot. Everything must fit into a single file. In some languages, this can be an uncomfortable or even unacceptable restriction. Sorry.

GradeBot also allows you to type your program directly into the web interface. For short programs, that is probably what happens most often. GradeBot also allows you to upload your file by selecting it on your local computer.

D.3 Choice of Language

GradeBot is able to accept programs in any of several different languages. The list includes C, C++, Java, Perl, Python, Ruby, and Tcl. Other languages may be added. Feel free to make a request.

You must tell GradeBot what language you are using. Currently Grade-Bot defaults to Perl. If you are using another language, just click on the appropriate radio button.

When grading your program, GradeBot will compile or interpret your program and then check its behavior. If your program has syntax errors, Grade-Bot will let you know. If there are no syntax errors, GradeBot will run your program. If you request Indent Checking, GradeBot will also look at your indenting to see whether it seems consistent. I often require consistent indenting.

Java: This popular language is perhaps the most disadvantaged with Grade-Bot for several reasons: (a) Java likes to have separate files; (b) Java takes a long time (half a second) to start, so 20 tests will take 10 seconds if you are lucky. The bottom line is that Java can be used, but it is a bit harder to make things work.

D.4 Customizations

GradeBot can customize its requirements for each student. This is done based on the first line of the program, which may be a required comment line.

If that first line includes a particular string, the whole line is used to seed a random number generator that creates the differing requirements for each student.

For example, the instructor may require each program to include a comment line that identifies the task and the student, something like this:

```
# cis101 g05 Colton, Don
```

GradeBot may be looking for the string `cis101 g05` as its trigger for customizing.

Because these customizations are intended to be different for every possible first line, the result is that every student may have to create a slightly

different program.

It is an honor code violation to submit a comment line that makes it look like you are somebody else, especially somebody else in the class. Other than that, it is okay to mess around with the comment line.

Changing the first line can change the customizations, so after you start working on a program, you should be careful about changing the first line. The other lines of the program do not affect the customizations in any way.

An example of a customization is the string the student must use to prompt for the first input. GradeBot might select from among the following options.

```
Please enter a number.
Kindly enter a number.
Please type in your number.
Type in a number please.
```

The purpose of this customization feature is to cut down on simple copying that might go on among students. Without customization, the same program could be submitted by every student. With customization every student would need to adjust the behavior of their program a bit to match the expectations of GradeBot.

D.5 GradeBot Log File

GradeBot keeps a log of programs submitted and their results. Currently it keeps the first line of the program (the comment line) together with the name of the lab being attempted and the results of testing (success or not). This information is available to me and can be used to give students credit for completing work even if they have not emailed it to me. For this reason, it is an honor code violation to submit a comment line that has someone else's name on it.

For programs that GradeBot thinks are successful, it also sends me an email that includes the full program. This information can be used to give students credit for completing work even if they have not emailed it to me.

Together these features reduce the need to email programs to me, but there may still be cases where emailing is required.

D.6   Standard In, Standard Out

Rigorous testing of computer programs is actually very difficult. To make it possible, I have made some decisions to keep things as simple as possible.

Tasks are generally limited to programs that read input from STDIN and write output to STDOUT. Beyond that, it can provide input through command line arguments, and it can inspect the return value from the programs it is testing.

That means that GradeBot does not get involved with mouse-based input, or network-based input or output, or graphical outputs.

Generally GradeBot does not get involved with reading and writing files, or accessing databases.

GradeBot also limits the amount of time each program is allowed to run.

D.7   The Grading Script

GradeBot has a version of each program you are asked to write. It generates sample inputs (somewhat randomly), runs its own version of the program, and collects the outputs. That is used to make a script: say this to the program, wait for the program to say this, repeat.

Your program is tested by seeing whether it can follow the script. Your program must behave exactly the same as GradeBot's program did.

This puts some serious constraints on your program. You must get all the strings right. If GradeBot wants "Please enter a number: " then that's exactly what your program must print. You may find yourself squinting at the output where GradeBot says you missed something. Usually it is a minor typographical problem.

Once you get the first test right, GradeBot typically invents another test and has you run it. And another. And another. Eventually, you either make a mistake, or you get them all correct.

If you make a mistake, GradeBot will tell you what it was expecting, and what it got instead.

If you get everything correct, GradeBot will announce your success. That means your program's observable behavior is correct.

D.8   Understanding GradeBot's Requirements

GradeBot is very picky. That is because it is really hard to tell when something is almost right, or close enough. GradeBot's only real choice is to require absolute perfection. That means spelling and spacing must be exact.

GradeBot shows you exactly what it wants, and exactly what you provided. Sometimes it can tell you what is wrong, but often you have to compare things and figure it out yourself. Just do a careful side-by-side comparison and adjust your spelling and spacing to make GradeBot happy.

(GradeBot may actually do things wrong in some cases. Maybe GradeBot spells something wrong or uses the wrong grammar. If you run into a case like that, the simple solution is to play along and do it the way GradeBot wants. You can also let me know where GradeBot is wrong and I may be able to fix it. I may even give extra credit for the mistakes you report.)

Standard Input

"`in>`" is shown to designate input that your program will be given (through the standard input channel).

Standard Output

Numbered lines are shown to designate output that your program must create.

Quotes are shown in the examples to delimit the contents of the input and output lines. The quotes themselves are not present in the input, nor should they be placed in the output.

Each line of output ends with a newline character unless specified otherwise.

"eof" stands for end of file and means that your program must terminate cleanly.

Command Line Inputs

GradeBot will start your program by saying this:

`GradeBot started your program with this command line:`

It will then present the command line that was used to start your program. Often there is no special command line input, and the command line simply starts your program. Sometimes there are command line arguments. Here is an example:

`"./gcd 35 28"`

In this example, `./gcd` is the name of your program that is being run. `35` is the first command line argument. `28` is the second command line argument.

In most languages, command line arguments are available as an array named "argv" or "args" or something similar.

Return Codes

When your program terminates, it must send back a return code of zero unless something else was specified in the requirements. Many languages do this automatically.

For C programs, remember to start your program with "int main" and end it with "return 0;"

For other languages, do something similar if necessary.

Appendix E

Programming Style

As your programs become more complex, style becomes important.

Contents

You should use good style. I consider programming style to be very important. I have different rules for CIS 101 and for the other classes I teach. Here we explain those rules.

In real life programming situations, it is common for work groups to adopt style rules. By using the same style, programs tend to be easier to read and understand. For most of the problems on each test, specific style is required.

Because style is a huge aid to making your program easier to read, I have developed the following style rules.

E.1 May, Should, and Must

In the requirements mentioned here and elsewhere, I use the words "may," "should," and "must" (and sometimes "should not" or "must not") to describe the importance of requirements.

I try to be careful in my use of these words so you can trust them.

When I say "may" or "can" it means that you are permitted but not encouraged nor required to do that particular thing.

When I say "should" it means that the thing is optional but strongly encouraged.

When I say "must" it means that the thing is fully required, and that without compliance, you will not receive credit.

E.2 In Every Class

Style is really all about making your program easy to read and update. That is what I truly care about. I have often seen programs that were buggy because they did not follow good style rules, and as a result the programmer had become confused.

Indenting: Indenting **should** correctly reveal the internal structure of your program. It **must** be consistent (a similar amount of indenting throughout your program) and reasonable (one or more spaces or tabs).

Naming: Variable names **must** helpfully describe the contents they hold. Subroutine names **must** helpfully describe what they do.

Comments: You **should** use comments to explain your code if things are not immediately obvious. Use comments to explain what you are trying to accomplish with the key paragraphs of your program. Use comments to explain anything that might be hard to understand in the future.

Your code **should** be readable to a programmer even if they are unfamiliar with your particular programming language. Most programming languages are similar enough that any programmer can read them. (Writing, on the

other hand, can often require memorization of syntax rules.)

If I complain about your style, what it probably means is I had trouble understanding your program. It may run fine in the computer, but it was difficult for me.

E.3   In CIS 101

In CIS 101 I have additional rules to help students learn good habits.

E.4   Spacing

The first style rule I require is spacing. You **must** put one space between tokens. There are a few exceptions.

Example: `(15+$x)` is bad because it does not have enough spaces.

Example: `( 15 + $x )` is good because the spacing is perfect.

This requires that you know what a token is. I cover this in the textbook.

Mistake: adding spaces inside a quoted string changes its meaning. A quoted string is by itself a single token. I require spaces between tokens, not within tokens.

Exception: You **should** omit the space before a semi-colon.

Example: `$x = 23 + $y;` is okay.

Example: `$x = 2 3 + $y;` is bad.

Exception: You **should** omit the space between a variable and a unary operator.

Example: `$x++;` is okay.

Example: `$x = -$y;` is okay.

Exception: You **should not** put a space between the dollar sign (or at sign) and the rest of the variable name.

Example: `$x` is okay.

Example: `$ x` is bad.

E.5   One Statement per Line

Each statement **should** be on its own line.

My Rule: Start a new line after each opening { or semi-colon.

Exception: A relevant comment **may** be placed after a semi-colon or opening curly brace.

Exception: Short statements are generally okay after the opening curly brace. I will accept things like these:

```
if ( $x == 5 ) { last }
if ( $x == 5 ) { $y++ }
else { print "goodbye" }
```

Exception: The `for` loop uses two semi-colons to organize its control structure (initialize; condition; step). You **must not** start a new line after those semi-colons unless it seriously improves the clarity of the code.

E.6   For Loops

The "for" loop uses semi-colons, so I am talking about it here, right after we talked about semi-colons.

There are a few special style rules that apply to for loops. The for loop has a specific control structure that is a bit different from other parts of the program. All the controls are specified at the top of the loop. In contrast, with a while loop, the init happens before the loop and the step happens at the end of the body.

The layout of a for loop is like this:

```
for ( init; cond; step ) { body }
```

The init (initialization) is an actual statement. You **may** have more than one statement, and separate them by commas.

The cond (condition) is not a statement, but is a condition like we would find with an if statement or a while loop.

The step is an actual statement. You **may** have more than one statement, and separate them by commas.

Taken together, init, cond, and step are the "controls" of the for loop, and style-wise they **must** be placed at the start of the loop to clearly indicate how the loop starts, stops, and moves forward. Normally they are all given on the same line, along with the word "for" itself. They **must not** be broken up on separate lines unless there is a special reason.

Because the controls should be kept together, it is bad style to do the init activity before the for loop, and bad style to do the condition inside the loop unless it is necessary, and bad style to do the step activity inside the body. Any one of these would violate the style requirements of the for loop.

Order of the condition: Most for loops are controlling a single variable, something like this.

Good: `for ( $x = 1; $x < 10; $x++ ) { body }`

We could also write it like this, but this is much harder to read because we expect the variable to be the first thing in each slot.

Bad: `for ( $x = 1; 10 > $x; $x++ ) { body }`

E.7 Indenting and Curly Braces

The "indent" is the number of blanks at the start of each line.

Summary: Your program **must** start out with no indenting. Each time you have an opening curly brace or opening parenthesis, you increase your indent. Each time you have a closing curly brace or closing parenthesis, you decrease your indent.

I use and recommend two spaces for the indent. You **must** use one or more spaces, or a tab, for your indent. You **must** be consistent. Decide on your indent and use it everywhere an indent is needed.

Crazy indenting makes programs substantially harder to read, so I am picky about this and GradeBot enforces it.

The main program **must not** be indented. There should be no spaces in front of the actual code.

Blocks are created by putting { before and } after some lines of code. These are called **curly braces**. This happens with decisions, loops, and subroutines.

Within the block, the indent **must** be increased by one step.

The opening { **must** be at the end of a line, and not on a line by itself. Normally it will be part of an if statement or a loop.

The closing } **must** be at the end of the last line in the block, and not on a line by itself. If there are two }s they **should** be on the same line.

The following is a good example.

```
if ( $x < 5 ) {
  print "$x is less than 5" }
```

The following is a **bad** example because the closing brace is on the next line.

```
if ( $x < 5 ) {
  print "$x is less than 5"
  }
```

The following is a **bad** example because the opening brace is on the next line.

```
if ( $x < 5 )
  {
  print "$x is less than 5" }
```

E.8   Use the Stated Values Exactly

Often a problem will specify certain numbers or strings that define how the program should run. If possible, you **must** use those exact same values in writing your program. If not, you **must** include a nearby comment that states the exact value.

Example: Print "Hello, World!"

Good: `print "Hello, World!"`

Okay: `print "Hello, World!\n"`

Bad: `print "hello, world!"` (wrong capitalization)

Bad: `print " Hello, World! "` (extra spaces)

Example: Print the numbers from 1 to 100.

Good: `for ( $i = 1; $i <= 100; $i++ ) { print $i }`

Bad: `for ( $i = 1; $i < 101; $i++ ) { print $i }`

If you cannot use the exact value specified in your program itself, then you **must** use the exact value in a comment nearby.

Example: If the last name is in the A-G range, do something.

Good: `if ( uc $ln lt "H" ) { # A-G`

E.9 Put Quotation Marks On Literal Strings

You **should** always put quote marks on literal strings. A literal string is a sequence of letters, possibly including interpolated variables.

It is wrong (but possible) to use strings without quotation marks.

Example: `$x = random; # $x will be "random"`

Example: `$x = currenttime; # $x will be "currenttime"`

Sometimes it fails to give the expected answer.

Example: `$x = rand; # $x will not be "rand"`

Example: `$x = localtime; # $x will not be "localtime"`

That is because unquoted strings, also known as barewords, can be interpreted to be function names or subroutine names. If nothing matches, then they are normally understood to be strings, but this cannot be relied upon.

Because the meanings cannot be relied upon, I strongly frown on this casual use of unquoted strings. I expect them to be properly quoted. Partly this is a style issue. Partly it goes deeper, as it avoids ambiguity.

E.10 Avoid Useless Statements

Every statement **must** make a meaningful contribution to the success of the program. Meaningless statements add confusion to the program, as future programmers are wondering why those statements exist.

One example is the useless chomp.

```
chomp ( $x = $y + 5 );
```

In this example, `$x` will not have a trailing newline because it is the result of a mathematical operation. The chomp is to remove trailing newlines, but one could never exist in this setting. Therefore the chomp is useless.

Another example is the useless initialization in a for loop:

```
for ( $x = $x; $x < 100; $x++ ) {
```

In this case, `$x` was apparently initialized before the loop (which is a style error itself), and the `$x = $x` is useless. You are copying x into x, which changes nothing but makes the program run slower (ever so slightly). In this case, the loop should be written like this:

```
for ( ; $x < 100; $x++ ) {
```

If you include a statement that is not necessary for the correct operation of the program, you **should** also include a comment that explains why that statement is there.

E.11 Use Helpful Names

Look in the textbook index for "variable naming."

Variables and subroutines are named. The computer does not care how meaningful the names are that you use, but programmers will care. I will care. The names **must** be helpful. They **must** bear some obvious relationship to the thing they represent.

Variable names often imply something about the values that are stored in them. Those implications should not be violated. For example, if you have a variable named `$start` it should contain a starting value. If you later change that value so that it is no longer a starting value, you have violated the meaning of the name. This will confuse people. Pick a better name.

Long descriptive names **may** be abbreviated and explained when used.

Example: `$eoy = 1; # eoy means end of year, 1 means true.`

Names like $x and $y **may** be used in short-range contexts where their meaning is clear by the immediately surrounding code. Like "he", "she", and "it" in English, they become confusing in wider contexts. Use something more meaningful.

Ambiguous names are not good. Avoid ambiguity.

Example: If you are converting temperature from Fahrenheit to Celsius, and you call one of your variables `$temp` or `$temperature` it is bad because it is not clear whether the information in that variable is the temperature in Celsius or in Fahrenheit. Variable names **should** give a clear indication of the type of information held inside.

Example: If you are asking for the quantity of engines and the cost of engines, then `$engine` is **not** a good variable name because one cannot tell whether you are talking about the quantity of engines or the cost per engine or maybe the total cost of that quantity of engines. Be clear.

E.12   Parentheses: Math vs Array

You **should not** use outer parentheses when calculating are assigning values.

You **should not** say `$x = ( something );` because the parentheses make it ambiguous. It has two possible meanings. Perl usually handles it okay, but I frown on it.

First Meaning: Mathematical Grouping

Parentheses **may** be used in a **mathematical expression** to force a certain order of operations.

Example: `$x = ( 3 + 2 ) * 5; # this is okay`

In the next example, we have added some outer parentheses. These outer parentheses do not help anything. They do not clarify anything. They do not change the meaning of the equation.

Example: `$x = ( ( 3 + 2 ) * 5 ); # this is discouraged`

Second Meaning: Defining an Array or List

Parentheses are also used in **defining arrays.**

Example: `@x = ( 3 ); # this is okay`

Here is the ambiguity that we wish to avoid:

Example: `$x = ( 3 ); # $x will be 3`

Example: `$x = @x = ( 3 ); # $x will be 1`

Bottom line? You **should not** put "outer" parentheses around a whole expression or statement.

E.13 Limited Use of Blank Lines

Blank lines **should not** be used in the program unless they improve the clarity of the code.

Sometimes it is helpful to add a blank line here or there to divide longer sections of code into natural paragraphs. It can show that some lines fit together as a natural block of related programming.

Appendix F

Credit for Catching Spelling Errors

I offer extra credit for reports of spelling and grammar errors in my formal communications, by which I mean written materials like syllabi, study guides, and text books as well as current portions of webpages. This is very helpful to me in correcting spelling mistakes. And it sometimes gets my students to read my materials more carefully.

Using Tools: This has gotten to be sort of a game at times, which makes it fun. We can get into Grammar Nazi mode and be picky, picky, picky. Students will cut and paste my words into a document and then run a spelling checker or grammar checker. Or they will directly open the PDF in a spelling or grammar checker.

Tools Break: You are welcome to do this, but you should be aware that spelling and grammar checkers work by a simplified set of rules compared to real life. If there are two spellings for a word, the spelling checker will commonly only accept one and will reject the other. This does not make the other wrong.

How To Tell Me: To get this credit, send me an email with details of the error (or supposed error) that you have found.

What To Tell Me: Provide enough context that I can easily find the error. Maybe copy and paste the whole paragraph where the error is. I give less credit if you just send me the page number and the wrong word. It is too much like looking for a needle in a haystack. Give me context.

Warning: Language Changes: The truth about English, and probably all languages, is that language changes over time. New words are created. New spellings are accepted. New grammar happens. And old grammar is resurrected.

Style Guides: I generally follow the accepted practices as shown in style guides such as the Chicago Manual of Style. But I take exception to certain things like those that are noted below. For things that I have considered and listed below, even though they may show up with a checker, I do not consider them to be incorrect.

My Rules: My rules are (a) is it commonly done? (b) is it ambiguous? (c) is it pretty? These are the same rules used by grammarians, but our decisions in any given case may be different.

Here is a list of things that have been previously considered.

webpage: Modern usage is split between making this two words, "web page," and one word, "webpage." I have decided to go with the one word version.

website: Modern usage is split between making this two words, "web site," and one word, "website." I have decided to go with the one word version.

themself: Modern usage has tended away from gender-specific words like himself in favor of gender-neutral words. I have migrated from him and her to "singular" **them** as my solution of choice to the gender-neutral dictates of modern political correctness. Some dictionaries do not recognize themself as a word, and instead suggest themselves. For plural them, this would be correct, but for singular them, themself is correct and is documented to have been used as far back in time as the 1400s.

vs: - Should it have a dot? The usage argument is that in British writing, abbreviations are dotted when the final letters have been dropped, but not when the intermediate letters have been dropped. Versus removes intermediate letters. American usage may differ. I do not put a dot after it. I don't like how it looks with a dot. It is a conscious decision, not an error.

zeros: versus zeroes: Both are considered correct. Google says that zeros is more commonly used.

Ambiguous Plurals: The plural of 15 is 15s, not 15's. Using an apostrophe generally indicates possession, but people do commonly (and incorrectly) use an apostrophe for plurals when without it the meaning seems less clear. My choice when making a plural that would look ambiguous is to quote the

string being pluralized. So, for me, the plural of (a) is ("a"s) rather than
(a's) or (as).

Ambiguous Quoted Punctuation: When should punctuation that is not
part of a quote be moved inside the quote marks? Typesetters tradition-
ally float a period (full stop) inside a trailing quote mark because it looks
better that way. In computing, quote marks typically delimit strings that
have special meaning, and putting punctuation inside the marks changes the
meaning of the string. I usually float punctuation if it does not change the
meaning of the thing quoted. Otherwise not.

Series Comma: Some people write a list of three things as (a, b and c),
but others write it as (a, b, and c). I write it the second way. This is not an
error. Both usages are correct, but I find the first usage to be ambiguous,
so I almost always use the second form.

Appendix G

Test Bank

Test Bank

1: (p.12) Can most activities be submitted early for extra credit?

2: (p.12) Can most activities be submitted late for partial credit?

3: (p.12) Can most activities be submitted after the deadline?

4: (p.16) When sending Prof Colton email about CIS 101, should I put cis101 in the subject line?

5: (p.16) When sending Prof Colton email about CIS 101, should I put CIS 101 in the subject line?

6: (p.16) When sending Prof Colton email about CIS 101, should I put my roll-book name in the subject line?

7: (p.20) Does the CIS department provide tutoring for this class?

8: (p.27) Can I bring blank paper into an exam?

9: (p.27) Can I bring notes into an exam?

10: (p.27) During exams can I make notes?

11: (p.27) During exams can I use email?

164

12: (p.27) During exams can I use my cell phone?

13: (p.27) During exams can I use my laptop computer?

14: (p.27) During exams can I use Google?

15: (p.27) During exams can I use a language dictionary?

16: (p.27) During exams can I run my programs to test them?

17: (p.28) Can I bring notes into a 3MQ?

18: (p.28) During 3MQs can I use the textbook?

19: (p.28) During 3MQs can I use Google?

20: (p.28) Is it okay to freely copy HTML from others?

21: (p.28) Is it okay to freely copy Perl from others?

22: (p.28) On homework, can I look at someone else's program?

23: (p.29) On homework, can I ask someone else questions about their program?

24: (p.29) On homework, can I point out possible errors in someone else's program?

25: (p.29) On homework, can I explain to someone else what I think their program is doing?

26: (p.29) On homework, can I explain what I think my program is doing?

27: (p.29) On homework, can I fix someone else's program?

28: (p.29) On homework, can I tell others what to type to fix their programs?

29: (p.29) On homework, can I accept a copy of someone else's program?

30: (p.29) On homework, can I make written notes while looking at someone else's program?

31: (p.29) On homework, can I take a picture or screen shot of someone else's program?

32: (p.29) On homework, can I explain my approach to someone else?

33: (p.30) On homework, can I give someone a copy of my program?

34: (p.30) On homework, can I let someone make changes to my program?

35: (p.30) Could cheating result in a zero on the assignment?

36: (p.30) Could cheating result in an F in the course?

37: (p.33) Can students with special needs get special accommodations?

Index