

CIS 101 Study Guide
Winter 2014

Don Colton
Brigham Young University-Hawai'i

April 9, 2014

Study Guide

This is the official study guide for the CIS 101 class, Beginning Programming, as taught by Don Colton, Winter 2014. It is focused directly on the grading of the course.

<http://byuh.doncolton.com/cis101/2141/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/cis101/2141/syl.pdf> is the official syllabus for this course. It is largely reproduced in Chapter 1 (page 3) below.

Text Book

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

The text book is available here, in PDF form, free.

<http://ipup.doncolton.com/>

The text book provides explanations and understanding about the content of the course.

Contents

1	Syllabus	3
2	Calendar	22
3	Problem Solving	24
4	DCQuiz: My Learning Management System	27
5	Activities General Information	33
6	GradeBot	40
7	Programming Style	45
8	Activities Assigned	51
9	Exam Questions	92
10	Final Projects	105
A	Spelling	109
	Index	111

Chapter 1

Syllabus

The original, separate syllabus is the official version. This is a copy of that syllabus, and is provided for your convenience and as a place for me to correct minor errors such as spelling mistakes.

Contents

1.1 Overview	4
1.1.1 Preparation	5
1.1.2 There May Be Changes	5
1.2 Course Details	5
1.2.1 About the Course	5
1.2.2 About the Instructor	5
1.2.3 My Websites	6
1.3 Learning Objectives	6
1.4 Communication	7
1.4.1 Me to You, Formal	7
1.4.2 Me to You, Informal	8
1.4.3 You to Me, Formal	8
1.4.4 You to Me, Informal	8
1.5 Grading	9
1.5.1 Tracking Your Grade	10
1.5.2 Effort: (50 points) Daily Update	10
1.5.3 Effort: (250 points) Study Time	10
1.5.4 Effort Points are Optional	11
1.5.5 Activities: Daily (135 points)	11

1.5.6	Activities: Project (40 points)	12
1.5.7	Skill: Exams (525 points)	13
1.5.8	Other Extra Credit	14
1.6	Calendar	14
1.6.1	Special Dates	15
1.6.2	Excused Absences	15
1.6.3	Reasonable Accommodation	15
1.7	Support	16
1.7.1	Office Hour / Open Lab	16
1.7.2	Study Groups	16
1.7.3	Tutoring	17
1.8	BYUH Learning Framework	17
1.8.1	Prepare for CIS 101	18
1.8.2	Engage in CIS 101	18
1.8.3	Improve at CIS 101	18
1.9	Standard Statements	18
1.9.1	Academic Integrity	19
1.9.2	Dress and Grooming Standards	20
1.9.3	Accommodating Special Needs	21
1.9.4	Sexual Harassment	21

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 powerful knowledge. It teaches skills by which you can better serve those around you. It teaches skills you can “take to the bank.”

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

1.1.1 Preparation

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.

1.1.2 There May Be Changes

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 are brave enough, you are welcome to suggest ways the class could be improved.

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

1.2 Course Details

1.2.1 About the Course

- **Course Number:** CIS 101
- **Title:** Beginning Programming
- **Course Description:** 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.
- **Textbook:** Introduction to Programming Using Perl and CGI, by Don Colton.
- **Classroom:** GCB 111
- **Start/End:** Mon, Jan 6 to Mon, Apr 7
- **Class Time:** MWF 11:00 to 12:00
- **Final Exam:** Wed, Apr 9, 10:00–12:50

1.2.2 About the Instructor

- **Instructor (me):** Don Colton
- **My email:** doncolton2@gmail.com

- **My Office:** GCB 128
- **Office Hour:** MWF 12:10 to 13:10.

I have reserved GCB 111 on MWF 12:10 to 13:10 so my students (and others) can study in a lab setting and meet with me and each other. I allow the room as an Open Lab for your use either individually or in groups, for my class or for other classes. MWF 12:10 to 13:10 I will be present in GCB 111 or in my office to assist students that come.

1.2.3 My Websites

Here is a list of my other websites that you may encounter this semester.

- <http://byuh.doncolton.com/cis101/> is my course homepage. It has links to everything else, including the study guide and the textbook.
- <http://ipup.doncolton.com/> has the textbook I wrote for this class.
- <https://dcquiz.byuh.edu/> is the learning management system for my courses.
- <http://byuh.doncolton.com/> is my campus homepage. It has my calendar and links to the homepages for each of my classes.
- <http://doncolton.com/> is my off-campus homepage.

1.3 Learning Objectives

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

1.4 Communication

We communicate with each other both formally and informally.

Formal communication tends to be written and precise. Formal is for anything truly important, like grades. Formal is authoritative.

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

1.4.1 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/cis101/2141/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/cis101/2141/sguide.pdf> is the study guide for this course. It includes a copy of the syllabus. The study guide is updated frequently throughout 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 provides 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.

1.4.2 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.

1.4.3 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 below. Exams will also be given using DCQuiz.

(b) **Email:** You will use formal email messages to submit some of the programs you write 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.

(c) **Student Projects:** The study guide may tell you to submit certain work in the form of a webpage or web-based program. If so, it will say specifically where to put it. I will go to that spot to grade it.

1.4.4 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 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.

1.5 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 points are divided up as follows.

Effort 300

Daily Update 50

Study Time 250

Achievement 700

Activities 135

Final Project 40

Exams 525

You need to earn a C or better (730 points or more) in the class if you plan to major in CS, IS, or IT. If you earn less, you must retake the class or change majors.

1.5.1 Tracking Your Grade

I keep an online grade book so you can see how your points are coming along. It also lets you compare them with other students in the class (without seeing their names).

<https://dcquiz.byuh.edu/> is my personal Learning Management System. That is where I maintain my online grade book.

Your points are organized into three grade books: Overall, Effort, and Activities.

2141 CIS 101 Overall Grade Book: The Overall includes the totals from Activity and Effort and adds your exam performance. It also shows your final grade.

2141 CIS 101 Effort Grade Book: The Effort tracks the daily updates and study time.

2141 CIS 101 Activities Grade Book: The Activities tracks your performance on in-class activities.

1.5.2 Effort: (50 points) Daily Update

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 during the 10-minute window of time that starts 5 minutes before class and ends 5 minutes into class.

The DU is worth two points per class period, with 50 points expected (for 25 out of 37 class periods), and about 75 points possible. Anything beyond 50 is extra credit. It is also a reward for coming on time, or close enough that you can do the update.

As part of the Daily Update, once a week I will ask you how much time you spent studying the previous week. I will use your report to update your study time points.

1.5.3 Effort: (250 points) Study Time

We award points for study time (ST), which is time spent engaging with materials directly related to this course.

Each week you are invited to report, on your honor, how many hours you studied during the previous week, Sunday morning through Saturday night. We award two “effort” points per hour of “study,” for a goal of 18 points (9 hours, including class time) and a maximum of 20 points (10 hours) per week, whether there is a holiday or not.

There are 14 weeks. $14 \times 18 = 252$. $14 \times 20 = 280$ (max). Anything beyond 250 points is extra credit.

Most students max out the study time points each week. This provides them with extra credit that helps ensure they get a good grade in the class.

Carry Forward: If you study more than the maximum time for which I will give credit, you are invited to report them, and also carry forward the extra hours and report them in the next week. For example, since 10 hours is the maximum that counts, if you studied 15 hours, you would report 15 hours of study, and I would count the first 10 hours. You would then take the remaining 5 hours and count it toward the following week.

There is no Carry Backward.

1.5.4 Effort Points are Optional

The effort points (daily update and study time) are there as a safety net. They are meant to be easy to earn. They help to make sure you will pass the class.

But when I calculate your final grade, I do it several ways:

- (a) Counting every point, based on 1000 total points.
- (b) Counting all but daily update and study time, based on 700 total points.

I grade several ways because some students have previous experience (or natural genius) and do not need to study as much.

I use whichever method gives you the best grade.

1.5.5 Activities: Daily (135 points)

On most days we will have an in-class activity assignment. Each will normally be worth 5 points.

Roughly 27 assignments \times 5 points = 135 points. The total will be 135.

Anything beyond that is extra credit.

The number of in-class activities is not perfectly predictable. The overall points will be adjusted so the full-credit values add up to 135 or more.

Points are assigned on a 0-to-5 basis as follows:

0: nothing found, or way too little.

1: It's a start. Runs without crashing.

3: Nice but missing something important.

4: Missing something minor.

5: Perfect. Totally meets the standards for achievement.

Bonus points may be given based on peer voting.

Some assignments may take two days and count double.

On activity work, you are encouraged to work with (but not just copy) your fellow students. We want everyone to get full credit on every assignment.

Every assignment will have ample opportunities for individual creativity. Duplicate work will break my heart.

1.5.6 Activities: Project (40 points)

(40) Project Points

10 Project CGI: write a dynamic web page

10 Project Pictures: use img tags

10 Project Multi Input: process multiple inputs

10 Project Hidden Fields: pass state (counter, etc)

The final project is due by 23:59 on Tuesday, the day after the last day of class. I plan to grade it early on Wednesday unless you have asked me to grade yours earlier.

Project points are earned for performance on out-of-class work. The project must be your own work. It should be fun. A game would be ideal. You are allowed to consult with others including websites but you are not allowed to cut and paste code written by others. Each online screen must clearly identify you as the author. It must accept user input. It should utilize hidden fields (state) that are needed for its operation.

Your final project cannot just be something we did in class. The in-class activities are good examples, and teach good principles, but they do not demonstrate understanding or creativity. If your project is based on something we did in class, it must go beyond it in some substantial and significant way.

<http://dc.is2.byuh.edu/cis101.2141/> is the place to link your project. It is the Student Projects page for this class. Link it to the “proj” slot.

See the study guide for a more detailed presentation of the official project details.

1.5.7 Skill: Exams (525 points)

There are 21 exam tasks. Each is a program for you to do during one of the final exams. Each is worth 25 points. Points for each question can be earned only once.

There are several exams given during the semester. Each one 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. Except for the last exam, they are called “early final” exams. Each early final lasts for about one hour. The last final lasts for about three hours. One practice exam is also given, for no credit, to help you understand how to do the other tests.

(525) Exam Points (21 tasks)

- 1 25p String Basic (1B)
- 2 25p Number Basic (2B)
- 3 25p Number Story (2S)
- 4 25p Number Decision (4D)
- 5 25p Number Decision Story (4S)
- 6 25p String Decision (5D)
- 7 25p String Decision Bracket (5B)
- 8 25p Repeat While (6W)
- 9 25p Repeat For (6F)
- 10 25p Repeat Last (6L)

- 11 25p Repeat Nested Loops (6N)
- 12 25p Lists Basic (7B)
- 13 25p Lists Loop (7L)
- 14 25p Arrays Basic (8B)
- 15 25p Arrays Loop (8L)
- 16 25p Split (8S)
- 17 25p Join (8J)
- 18 25p Subroutine Returns (9R)
- 19 25p Subroutine Positional Parameters (9P)
- 20 25p Subroutine Globals and Locals (9G)
- 21 25p Subroutine Variable Parameters (9V)

The study guide talks more about each of these tasks.

1.5.8 Other Extra Credit

Report an error in my formal communications (the published materials I provide), so I can fix it. In this class, the 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 are common and do not count.)

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

1.6 Calendar

We meet about 37 times plus the final.

The due date and deadline for things will be published in the study guide and mentioned in class. The study guide will be updated regularly throughout the semester.

1.6.1 Special Dates

Mo Jan 06 First Day of Instruction
Fr Jan 17 **exam 0** practice test
Mo Jan 20 No Class: Human Rights Day
Fr Jan 24 **exam 1**
Fr Feb 07 **exam 2**
Mo Feb 17 No Class: Presidents Day
Fr Feb 21 **exam 3**
Fr Mar 07 **exam 4**
Fr Mar 21 **exam 5**
We Mar 26 No Class: Kuhio Day
Fr Apr 04 **exam 6**
Mo Apr 07 Last Day of Instruction
Tu Apr 08 All Late-Work Deadline
We Apr 09 **exam 7** Final Exam, 10:00–12:50

The exam dates will not change unless there is a fire or a flood or something. Exams happen about twice a month. They are closed-book, closed-notes, closed-neighbor, etc. You can bring blank paper. **Some memorization is required.**

1.6.2 Excused Absences

There are many good reasons why students request special treatment. Instead of dealing with these as they arise, based on my years of experience, I have adopted general policies that are intended to accommodate all but the most difficult cases.

1.6.3 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: I give the final exam seven times.

Deadlines: Most assignments are due soon after they are discussed, but I normally allow late work at full credit for several more weeks (except at the end of semester).

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.

1.7 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.

1.7.1 Office Hour / Open Lab

I have reserved GCB 111 on MWF 12:10 to 13:10 so my students (and others) can study in a lab setting and meet with me and each other. I allow the room as an Open Lab for your use either individually or in groups, for my class or for other classes. MWF 12:10 to 13:10 I will be present in GCB 111 or in my office to assist students that come.

1.7.2 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.

1.7.3 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.

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

Not all of the tutors know about everything. But all of the tutors should know which tutors do know about whatever you are asking about, so they can direct you toward the best time to get your questions answered.

The best way to work with a tutor is to show them something that you have written and ask them why it does not work the way you want. This can open the door to a helpful conversation.

Another good way to work with a tutor is to show them something in the textbook and ask about it.

The worst way to work with a tutor is to plunk down next to them and say, "I don't understand. Can you teach me?" If you did not try hard to read carefully, you are wasting everybody's time.

1.8 BYUH Learning Framework

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

1.8.1 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 listening to a general lecture.

1.8.2 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. Those that finish first are encouraged to help those that want assistance. It is amazing what you can learn by trying to help someone else.

1.8.3 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. Review your answers and those of other students. See how your answers could be improved. If you feel lost, study the readings again or ask for help.

1.9 Standard Statements

All syllabi are encouraged or required to address certain topics. These are generally considered to be common sense, but we find that it is useful to mention them explicitly anyway.

1.9.1 Academic Integrity

We learn by watching others and then doing something similar.

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.

Do not simply copy. Do your own work. When I review computer code, sometimes I see quirky ways of doing things. They appear to work even though they may be wrong. And then I see someone else that has done it exactly the same wrong way. This does not feel like “doing your own work.” Cut and paste is pretty much an honor code violation. Read and learn is totally okay. Copying other ideas is okay. I don’t want to see any cut and paste.

<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.

CIS 101: In this course group work is permitted and encouraged but you are not allowed to turn in work that is beyond your understanding, whether you give proper attribution or not. Make

sure you understand what you are submitting and why each line is there.

You must write your own programs. You can look at what other people have done, and you can show other people what you have done, but you are forbidden to copy it. Look at it, yes. Understand it, yes. Ask about it, yes. Explain it, yes. Copy it, no.

CIS 101: 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.

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.

1.9.2 Dress and Grooming Standards

The dress and grooming of both men and women should always be modest, neat and clean, consistent with the dignity adherent to representing The Church of Jesus Christ of Latter-day Saints and any of its institutions of higher learning. Modesty and cleanliness are important values that reflect personal dignity and integrity, through which students, staff, and faculty represent the principles and standards of the Church. Members of the BYUH community commit themselves to observe these standards, which reflect the

direction given by the Board of Trustees and the Church publication, “For the Strength of Youth.” The Dress and Grooming Standards are as follows:

Men. A clean and neat appearance should be maintained. Shorts must cover the knee. Hair should be clean and neat, avoiding extreme styles or colors, and trimmed above the collar leaving the ear uncovered. Sideburns should not extend below the earlobe. If worn, moustaches should be neatly trimmed and may not extend beyond or below the corners of mouth. Men are expected to be clean shaven and beards are not acceptable. (If you have an exception, notify the instructor.) Earrings and other body piercing are not acceptable. For safety, footwear must be worn in all public places.

Women. A modest, clean and neat appearance should be maintained. Clothing is inappropriate when it is sleeveless, strapless, backless, or revealing, has slits above the knee, or is form fitting. Dresses, skirts, and shorts must cover the knee. Hairstyles should be clean and neat, avoiding extremes in styles and color. Excessive ear piercing and all other body piercing are not appropriate. For safety, footwear must be worn in all public places.

1.9.3 Accommodating 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 any disability that may impair your ability to complete this course successfully, please contact the students with Special Need Coordinator, Leilani A‘una at 808-293-3518. Reasonable academic accommodations are reviewed for all students who have qualified documented disabilities. If you need assistance or if you feel you have been unlawfully discriminated against on the basis of disability, you may seek resolution through established grievance policy and procedures. You should contact the Human Resource Services at 808-780-8875.

1.9.4 Sexual Harassment

Title IX of the education amendments of 1972 prohibits sex discrimination against any participant in an educational program or activity that receives federal funds, including federal loans and grants. Title IX also covers student-to-student sexual harassment. If you encounter unlawful sexual harassment or gender-based discrimination, please contact the Human Resource Services at 808-780-8875 (24 hours).

Chapter 2

Calendar

Following is the projected calendar. Items marked with ** are firmly scheduled. Other topics and due dates may change.

Mo Jan 06 37: o1 First Webpage
We Jan 08 36: Unit 2: Input, g21 Hi Fred
Fr Jan 10 35: Unit 1: Output, oR Random Number
Mo Jan 13 34: oD Dice, images.
We Jan 15 33: cM Mad Lib
Fr Jan 17 32: ** **exam 0** practice test
Mo Jan 20 ** No Class: Human Rights Day
We Jan 22 31: Unit 3: Math, g31 Before After
Fr Jan 24 30: ** **exam 1**
Mo Jan 27 29: Unit 4: Decisions, g41 Numeric Decision
We Jan 29 28: g42 Birthday
Fr Jan 31 27: Unit 5: Decisions, g51 Phone Book
Mo Feb 03 26: oM Mad Lib
We Feb 05 25: Unit 6: Loops
Fr Feb 07 24: ** **exam 2**
Mo Feb 10 23: oF Farm1
We Feb 12 22: Unit 7: Arrays, g71 Array 1
Fr Feb 14 21: g72 Roll
Mo Feb 17 ** No Class: Presidents Day
We Feb 19 20: oT LocalTime
Fr Feb 21 19: ** **exam 3**
Mo Feb 24 18: g34 Celsius
We Feb 26 17: g45 Afford

Fr Feb 28 16: oHL High Low
Mo Mar 03 15: oHL High Low
We Mar 05 14: oD2 Multi Dice
Fr Mar 07 13: ** **exam 4**
Mo Mar 10 12: Discuss Subroutines
We Mar 12 11: oF2 Farm 2
Fr Mar 14 10: Unit 8: Subroutines, oF2 Farm 2
Mo Mar 17 9: oF2 Farm 2
We Mar 19 8: g43 Leap Year
Fr Mar 21 7: ** **exam 5**
Mo Mar 24 6: g64 Factors
We Mar 26 ** No Class: Kuhio Day
Fr Mar 28 5: g75 Tally
Mo Mar 31 4: oB Boring
We Apr 02 3: oJS JavaScript
Fr Apr 04 2: ** **exam 6**
Mo Apr 07 1: late work makeup
Tu Apr 08 ** All Late-Work Deadline, 23:59
We Apr 09 0: ** **exam 7** Final Exam, 10:00–12:50

Chapter 3

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.

Contents

3.1	cPanel Login Problems	24
3.2	Programming Problems	24
3.3	Webpage Problems	25
3.3.1	Webpage Does Not Load	25
3.3.2	404 Error	25
3.3.3	Internal Server Error	26
3.3.4	Blank or Incomplete Webpage	26

3.1 cPanel Login Problems

If you are trying to use cPanel and have trouble logging in, the person to see is Micah Uyehara, in GCB 106. His email address is <micah.uyehara@byuh.edu>. He runs the IS2 machine where our cPanel accounts are stored.

3.2 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 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.

3.3 Webpage Problems

3.3.1 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 person to see is Micah Uyehara, in GCB 106. He runs 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 Micah has solutions.

3.3.2 404 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 things 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 ”.

3.3.3 Internal Server Error

This means that your program tried to run, but that it did not return a usable webpage.

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.)

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.

3.3.4 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>`.

Chapter 4

DCQuiz: My Learning Management System

Contents

4.1	Grade Book	28
4.2	Daily Update	28
4.2.1	Study Time	28
4.2.2	Comment	28
4.2.3	Genuine Questions	29
4.3	Exams	29
4.3.1	Taking Exams	30
4.3.2	Reviewing Exams	31
4.4	Other Features	32

I have developed my own learning management system (LMS) that will be used 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.

4.1 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?

4.2 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.

4.2.1 Study Time

Generally I give you the opportunity to tell me how much study time you have accumulated since the last reporting. Normally this is reported on the first class of the week, and covers the prior week (Sunday through Saturday).

4.2.2 Comment

Generally 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.

4.2.3 Genuine Questions

I may include genuine questions in the daily update, and these can be graded. It's kind of unpredictable.

4.3 Exams

DCQuiz was originally developed for giving tests. My problem was handwriting, 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.

4.3.1 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 do 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.

4.3.2 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.

4.4 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.

Chapter 5

Activities General Information

Contents

5.1	oXX: Online General Rules	34
5.2	cXX: Command-Line General Rules	35
5.3	gXX: GradeBot Task General Rules	35
5.4	Email Submission Rules	37
5.4.1	To: Line	37
5.4.2	Subject Line	37
5.4.3	Body When Submitting a Program	38

We assume you are studying outside of class time, and that the text book 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 8 (page 51), the Activities 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.

Discussed means the date we talked about it in class.

5pt Due Date means the date by which you must complete the assignment for it to earn 5 points.

4pt Due Date means the date by which you must complete the assignment for it to earn 4 points, if you miss the 5pt Due Date.

3pt Due Date means the date by which you must complete the assignment for it to earn 3 points, if you miss the 4pt Due Date.

The 3pt Due Date is usually **Tue, Apr 8, 23:59**.

23:59 means 11:59 PM.

If you are submitting something after the 5pt Due Date but before the 3pt Due Date, you must either submit it by email or else notify me by email so I know to grade it.

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 “CIS 101 Activities” grade book in the column specified by the grading label.

5.1 oXX: Online General Rules

Online tasks generally follow these rules. Exceptions and clarifications are provided for each task.

Label: Each online task has a grading label consisting of the letter “o” (for online) followed by (normally) 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 should clearly display your name. Other requirements vary by task.

How to Submit: Create a webpage properly linked to the student projects page. I normally grade everyone’s submissions at once.

Late Work: If you complete or improve your work so that regrading may be justified, tell me so via email. Follow the email rules in section 5.4 (page 37) in the construction of your subject line.

5.2 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) two other characters that specify which online task it is.

Task: Write a program. Requirements vary by task.

Follow the email rules in section 5.4 (page 37) 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 possibly additional comments.

5.3 gXX: GradeBot Task General Rules

GradeBot itself is explained in chapter 6 (page 40).

GradeBot tasks generally follow these rules. Exceptions and clarifications are provided for each task.

Label: Each GradeBot task has a grading label consisting of the letter “g” (for gradebot) followed by (normally) two other characters that specify which online task it is.

Currently those characters are numeric digits.

Lab ID: Within GradeBot, each task has a lab ID (name). Normally this ID consists of “cis101.(label).(word)” where (label) is the label above (with or without the g prefix), and (word) helps identify and describe the lab.

Task: Write a program. GradeBot gives further directions for each program.

Have GradeBot test your program. When your program is running correctly, you will get this message:

```
Success! No errors found. Nice job. Assignment complete!
```

After receiving this message, you can submit your program to me. I may require additional things, such as the use of specific program elements, or specific style.

Follow the email rules in section 5.4 (page 37) as you construct your subject line and the body of your message.

Specifically, your subject line should look like this:

```
cis101 gxx lastname, firstname
```

 is the required subject line.

where `gxx` is replaced by the grading ID number, `lastname` is replaced by your lastname as shown on my roll sheet, and `firstname` is replaced by your firstname as shown on my roll sheet. (You can put a comma between lastname and firstname if you want.)

The body of your email should be your program and nothing else, in plain text. The first line of your program must be a comment line (in Perl) that repeats the subject line but with a hash in front, like this:

```
# cis101 gxx lastname, firstname
```

 is the required comment line.

Is it difficult for me to tell what part of your email is your program? Submit only your program, and not anything else.

Is your program not copy-paste ready? For example, does it have `>` at the front of any of the lines? It should not.

Is your program in plain text or rich (html) text? It should be in plain text. Sometimes the use of **rich text** changes your program by inserting invisible characters that make your program fail. For that reason, I may reject any program sent as rich text.

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 I tell you to fix the problem and resubmit your program.

If your submission was acceptable, I will reply to it with the word “Success” and possibly additional comments. You may want to save my reply until you see your grade reflected in my gradebook.

5.4 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.

5.4.1 To: Line

Please send email to <doncolton2@gmail.com>. That is my preferred email address.

If you 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.

5.4.2 Subject Line

The subject line of the email must be as follows:

```
cis101 label lastname, firstname
```

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 `label` part is replaced by the grading label for that assignment.

The `lastname` part is to be replaced by your own last name.

The `firstname` part is to be replaced by your own first name. 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

5.4.3 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.

The problem is that sometimes **rich text** inserts invisible characters into your program in a way that makes it impossible for a cut-and-paste of the program to run. Then I am faced with the choice of rejecting the program, or finding and deleting the bogus characters.

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.

Chapter 6

GradeBot

GradeBot is my automated program grader.

Contents

6.1	Where Can I Find GradeBot?	41
6.2	Source Code File	41
6.3	Choice of Language	41
6.4	Standard In, Standard Out	42
6.5	The Grading Script	42
6.6	Interpreting GradeBot's Requirements	43

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 style and algorithm, I usually rely on visual inspection of the program source code.

When I grade on behavior, I try to use 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.

This particular version of GradeBot is called GradeBot Lite because it is descended from what used to be a huge system that was also called GradeBot.

6.1 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 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.

6.2 Source Code File

To keep things simple, GradeBot requires you to submit a single file of source code. You are not able 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 restriction.

GradeBot does allow you to type your program directly into the web interface. It also allows you to upload your file by selecting it on your local computer.

If you use the web interface, GradeBot will (incorrectly) tell you “ERROR: uploaded file type: empty”. You should ignore this message. Someday I will get it fixed.

6.3 Choice of Language

GradeBot is able to accept programs in several different languages. The list includes C, C++, Java, Perl, Python, Ruby, and Tcl.

You must tell GradeBot what language you are using. Simply 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, GradeBot will let you know. If there are no syntax errors, GradeBot will run your program.

Java: This popular language is perhaps the most disadvantaged with GradeBot 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.

6.4 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.

6.5 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 error.

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.

6.6 Interpreting 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.)

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.

Chapter 7

Programming Style

As your programs become more complex, style becomes important.

Contents

7.1	Other Than CIS 101	46
7.2	In CIS 101	46
7.3	Spacing	46
7.4	Use the Values Specified	47
7.5	Parentheses: Math vs Array	48
7.6	One Statement per Line	48
7.7	Indenting	49
7.8	Helpful Blank Lines	49
7.9	Helpful Names	49

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. This chapter explains 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.

7.1 Other Than CIS 101

In CIS 101 I am partly on a mission to teach you to use good style. As part of that I have very specific rules that I enforce. In other classes, my style rules are much more simple, and are as follows.

Style is really all about making your program easy to read and update. That is all I really care about.

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.

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.)

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.

Indenting should correctly reveal the internal structure of your program. It should be consistent and reasonable. I personally like indenting by two spaces for each additional level of nesting.

Variable names should helpfully describe the contents they hold. Subroutine names should helpfully describe what they do.

7.2 In CIS 101

The remainder of this chapter gives the very specific style rules that I enforce.

7.3 Spacing

The first style rule I require is spacing. I am very picky. You must put one space between tokens. There are a few exceptions.

Example: `(3+5)` is bad because not enough spaces.

Example: `(3 + 5)` is good because spacing is perfect.

This requires that you know what a token is. I cover this in the text book.

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 may omit the space before a semi-colon.

Example: `$x = (3 + 5);` is okay.

Exception: You may omit the space between a variable and a unary operator.

Example: `$x++;` is okay.

Example: `$x = -$y;` is okay.

7.4 Use the Values Specified

Often a problem will specify certain numbers or strings that define how the program should run. If possible, use those exact same values in writing your program. If not, include a comment that has 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 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`

7.5 Parentheses: Math vs Array

In the form `$x = (something);` there is a confusing ambiguity. I do not allow it because it is confusingly ambiguous.

The problem is ambiguity. It has two possible meanings. Perl probably handles it okay, but I still do not accept it.

Parentheses can be used in a **mathematical expression** to force a certain order of operations.

Example: `$x = (3 + 2) * 5; # this is okay`

Example: `$x = ((3 + 2) * 5); # this is not okay`

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? Do not put parentheses around a whole expression or statement. If you do, I will probably mark it wrong.

7.6 One Statement per Line

Each statement should be on its own line.

In real life, statements are often combined onto one line if they are closely related. This is not real life. For exams, it is easier if I have a simple rule and stick with it.

Start a new line after each opening `{` or semi-colon.

Exception: The `for` loop uses two semi-colons to separate its control structure (`init; condition; step`). You should not normally start a new line after those semi-colons.

Exception: A relevant comment can be placed after a semi-colon.

7.7 Indenting

Indent is the number of blanks at the start of each line.

The main program should 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. This happens with decisions, loops, and subroutines.

Within the block, I require indenting to be increased by two.

Warning: Because crazy indenting makes programs substantially harder to read, I have become very picky about this.

Warning: If you write your program using an editor like notepad++, and then cut-and-paste it to save as your exam answer, the indenting may be messed up. You should go back through your program and fix any indenting problems that may have occurred.

Common Error: using TAB instead of two spaces. I will mark it wrong.

Common Error: using one space instead of two spaces. I will mark it wrong.

7.8 Helpful Blank Lines

Blank lines are used to divide a program into natural “paragraphs.” The lines within each paragraph are closely related to each other, at least as seen by the programmer.

Rule: Keep things fairly compact. Use blank lines and comments to help visually identify groups of related lines. Do not use an excessive number of blank lines.

7.9 Helpful Names

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 should be helpful. They should bear some obvious relationship to the thing they represent.

Long descriptive names can 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.

Chapter 8

Activities Assigned

Activities give you a way to develop and test your programming skills.

Contents

8.1	o1: First Webpage	53
8.1.1	Online Expectations	54
8.2	g21: Hi Fred	56
8.3	oR: Random Number	57
8.4	oD: Dice	59
8.5	cM: Mad Lib	61
8.6	g31: Before After	62
8.7	g41: Numeric Decision	63
8.8	g42: Birthday	64
8.9	g51: Phone Book	66
8.10	oM: Mad Lib	67
8.11	oF: Farm 1	70
8.12	g71: Array 1	71
8.13	g72: Roll	73
8.14	oT: LocalTime	75
8.15	g34: Celsius	76
8.16	g45: Afford	77
8.17	oHL: High Low	79
8.18	oD2: Multi Dice	81
8.19	oF2: Farm 2	83
8.19.1	Main Program: Crops	83

8.19.2	Main Program: Call “plant”	84
8.19.3	Subroutine “Plant”	84
8.19.4	Main Program: Harvesting Call	84
8.19.5	Subroutine “Harvest”	85
8.20	g43: Leap Year	86
8.21	g64: Factors	88
8.22	g75: Tally	89
8.23	oB: Boring	90
8.24	oJS: JavaScript	91

This chapter lists the activities that have been assigned. As new activities are assigned, they will be added to this chapter.

8.1 o1: First Webpage

- Status: Officially Assigned.
- Discussed: Mon, Jan 6
- 5pt Due Date: Wed, Jan 8, 12:00
- 4pt Due Date: Fri, Jan 10, 12:00
- 3pt Due Date: Tue, Apr 8, 23:59
- Grading Label: **o1**

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 5.1 (page 34) apply.

If you are requesting a regrade, this is the required subject line:

```
cis101 regrade o1 lastname, firstname
```

We will try to 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).

Create a directory (folder) with `cis101.2141` as its name. The `cis101` part stands for this class. The `2141` part stands for this semester.

In that directory 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. Make sure you at least have this minimum content: head, title, body, h1, image.

Avoid these problems: any HTML syntax error, copied content that is not properly updated.

You can use the following lines as an example. You can use cut-and-paste if you want, but it is better to 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>  
<meta name=description content="Don's CIS 101 Homepage">
```

```
</head><body>
<h1>This is Don's CIS 101 Homepage</h1>
<img src=mypic.jpg alt="my picture" width=500>
</body>
```

Change the name “Don’s” to match your own name, of course.

Upload a picture of yourself. It can simply be something that represents you, or it can be an actual picture of yourself. Name it whatever you want, so long as the webpage works correctly.

Verify that you can reach your page through clicking on your o1 link on the CIS 101 Student Projects webpage, which is:

<http://dc.is2.byuh.edu/cis101.2141/>

8.1.1 Online Expectations

This is not a web design class, but there are a few simple things that I will expect anyway. They are good habits to get into. I will enforce them from time to time.

For anything you put online, including webpages created by hand and those created by running a program, there are a few expectations that I have. The following things should be present, in this order.

1. Have a proper DOCTYPE statement.

Example: `<!DOCTYPE html>`

2. Have a proper head statement.

Example: `<head lang=en>`

3. Have a proper meta charset statement.

Example: `<meta charset=utf-8>`

4. Have a proper title. Titles are up to about 50 characters long.

Example: `<title>Don's Page About Whatever</title>`

5. Have a proper meta description statement. 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.

Example: `<meta name=description content="Don's Page About Whatever">`

6. Have a proper style section, even if it is empty. Between these lines you would put any styling you will be doing.

Example starting line: `<style>`

Example ending line: `</style>`

7. Have a proper body statement.

Example: `<body>`

8. Have a proper h1 statement that identifies yourself and your project. Typically this is similar to your title.

Example: `<h1>Don's Project About Whatever</h1>`

In addition to all of that, I should not see any visible errors when I do a "show page source" in Firefox.

8.2 g21: Hi Fred

- Status: Officially Assigned.
- Discussed: Wed, Jan 8.
- 5pt Due Date: Fri, Jan 10, 12:00
- 4pt Due Date: Mon, Jan 13, 12:00
- 3pt Due Date: Tue, Apr 8, 23:59
- Grading Label: **g21**

This is a GradeBot task. The general rules and explanations in section 5.3 (page 35) apply, including email subject line and program comment line.

`cis101 g21 lastname, firstname` is the required subject line.

`# cis101 g21 lastname, firstname` is the required comment line.

Gradebot can be found at <http://gradebot.tk/>
and at <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's your name?"
in> "Fred"
  2: "Hello, Fred!"
eof (end of output)
```

8.3 oR: Random Number

- Status: Officially Assigned.
- Discussed: Fri, Jan 10.
- 5pt Due Date: Mon, Jan 13, 12:00
- 4pt Due Date: Wed, Jan 15, 12:00
- 3pt Due Date: Tue, Apr 8, 23:59
- Grading Label: **oR**

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 5.1 (page 34) apply.

If you are requesting a regrade, this is the required subject line:

```
cis101 regrade oR lastname, firstname
```

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.

Log into cPanel. Go to your cis101.2141 folder. Create a sub-folder with `random` as its name. In it create a file with `index.cgi` as its name.

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 first line of your program must look like this, including spaces:

```
#!/usr/bin/perl --
```

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).

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

```
content-type: text/html

<!DOCTYPE html><head lang=en><meta charset=utf-8>
<title>Don's Random Number Generator</title>
<meta name=description content="Get a random number.">
</head><body>
<h1>Don's Random Number Generator</h1>
<h2>${random}</h2>
</body>
```

(Unless your name is Don, it should not say Don in it.)

Remember to **print** that webpage. The content-type part is important. We will talk about it in class, and you can look it up in the text book.

`random` is supposed to be a variable that contains a random number. You can create one like this:

```
random = rand(30);
```

That will create a random number between zero and 30.

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 oR link on the CIS 101 Student Projects webpage, which is:

<http://dc.is2.byuh.edu/cis101.2135/>

8.4 oD: Dice

- Status: Officially Assigned.
- Discussed: Mon, Jan 13.
- 5pt Due Date: Wed, Jan 15
- 4pt Due Date: Fri, Jan 17
- 3pt Due Date: Tue, Apr 8, 23:59
- Grading Label: **oD**

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 5.1 (page 34) apply.

If you are requesting a regrade, this is the required subject line:

```
cis101 regrade oD lastname, firstname
```

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 rolls two (or more) multi-sided dice randomly each time you load it.

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

In the proper location, create a sub-folder with `dice` as its name. In it create a file with `index.cgi` as its name.

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

```
content-type: text/html

<!DOCTYPE html><head lang=en><meta charset=utf-8>
<title>Don's Dice Roller</title>
<meta name=description content="roll some dice">
<style>
body { background-color: #35ffff; text-align: center; }
</style>
</head><body>
<h1>Don's Dice Roller</h1>

<p>We will roll some dice.</p>
```

```
<p>We rolled a 1 and a 6.</p>
<p><img src=1.jpg alt=1 title='we rolled 1'>
<img src=6.jpg alt=6 title='we rolled 6'></p>
</body>
```

Instead of printing an actual 1 and 6 as shown, use variables, maybe \$d1 and \$d2 for dice 1 and dice 2.

```
$d1 = 1 + int ( rand(6) );
```

That will create a random integer between 1 and 6.

The `rand(6)` part will generate a number between zero and six, but it will never actually be six. One sixth 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 integer 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 sixth 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 dice will appear.

You will also need images of dice. You are welcome to copy my dice from my webpage and upload them to your own webpage.

Feeling creative? You can use another kind of dice (like a d8 or a d20). You can use other images, maybe star, bell, cherry, rainbow, etc, like a slot machine. But please do have at least six image choices, and at least two displayed at a time.

8.5 cM: Mad Lib

- Status: Officially Assigned.
- Discussed: Wed, Jan 15.
- 5pt Due Date: Fri, Jan 17, 13:00
- 4pt Due Date: Wed, Jan 22, 13:00
- 3pt Due Date: Tue, Apr 8, 23:59
- Grading Label: cM

This is a command-line task. The general rules in section 5.2 (page 35) apply, including email subject line and program comment line.

`cis101 cM lastname, firstname` is the required subject line.

`# cis101 cM lastname, firstname` 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 with a list of words and then you insert them into the blanks of a story. The result is sometimes very funny.

Task: Write a program. Prompt for at least three 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.

Requested: Please make the story creative and interesting.

8.6 g31: Before After

- Status: Officially Assigned.
- Discussed: Wed, Jan 22.
- 5pt Due Date: Fri, Jan 24, 13:00
- 4pt Due Date: Mon, Jan 27, 13:00
- 3pt Due Date: Tue, Apr 8, 23:59
- Grading Label: **g31**

This is a GradeBot task. The general rules and explanations in section 5.3 (page 35) apply, including email subject line and program comment line.

`cis101 g31 lastname, firstname` is the required subject line.

`# cis101 g31 lastname, firstname` is the required comment line.

Gradebot can be found at <http://gradebot.tk/>
and at <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.

8.7 g41: Numeric Decision

- Status: Officially Assigned.
- Discussed: Mon, Jan 27.
- 5pt Due Date: Wed, Jan 29, 13:00
- 4pt Due Date: Fri, Jan 31, 13:00
- 3pt Due Date: Tue, Apr 8, 23:59
- Grading Label: **g41**

This is a GradeBot task. The general rules and explanations in section 5.3 (page 35) apply, including email subject line and program comment line.

`cis101 g41 lastname, firstname` is the required subject line.

`# cis101 g41 lastname, firstname` is the required comment line.

Gradebot can be found at <http://gradebot.tk/>
and at <http://gbot.dc.is2.byuh.edu/>.

Approved style is required on this task. The rules in chapter 7 (page 45) apply. Be careful with your 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)

```


8.8 g42: Birthday

- Status: Officially Assigned.
- Discussed: Wed, Jan 29.
- 5pt Due Date: Fri, Jan 31, 13:00
- 4pt Due Date: Mon, Feb 03, 13:00
- 3pt Due Date: Tue, Apr 8, 23:59
- Grading Label: **g42**

This is a GradeBot task. The general rules and explanations in section 5.3 (page 35) apply, including email subject line and program comment line.

`cis101 g42 lastname, firstname` is the required subject line.

`# cis101 g42 lastname, firstname` is the required comment line.

Gradebot can be found at <http://gradebot.tk/> and at <http://gbot.dc.is2.byuh.edu/>.

Approved style is required on this task. The rules in chapter 7 (page 45) apply. Be careful with your spacing, indenting, and choice of variable names.

There is a chapter in the textbook, probably about chapter 25, probably about page 129, that talks about and, or, not, and related operations that might be helpful with this task.

Task: Tell how many years old a person is. This will probably 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 Feb 26 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)
```

8.9 g51: Phone Book

- Status: Officially Assigned.
- Discussed: Fri, Jan 31.
- 5pt Due Date: Mon, Feb 03, 13:00
- 4pt Due Date: Wed, Feb 05, 13:00
- 3pt Due Date: Tue, Apr 8, 23:59
- Grading Label: **g51**

This is a GradeBot task. The general rules and explanations in section 5.3 (page 35) apply, including email subject line and program comment line.

`cis101 g51 lastname, firstname` is the required subject line.

`# cis101 g51 lastname, firstname` is the required comment line.

Gradebot can be found at <http://gradebot.tk/>
and at <http://gbot.dc.is2.byuh.edu/>.

Approved style is required on this task. The rules in chapter 7 (page 45) apply. Be careful with your spacing, indenting, and choice of variable names.

There is a chapter in the textbook, probably about chapter 26, probably about page 133, that talks about string comparisons. They are basically like numeric comparisons, but use different operators.

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

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: "It would be on page 20."
eof (end of output)

```

8.10 oM: Mad Lib

- Status: Officially Assigned.
- Discussed: Mon, Feb 3
- 5pt Due Date: Wed, Feb 5, 13:00
- 4pt Due Date: Fri, Feb 7, 13:00
- 3pt Due Date: Tue, Apr 8, 23:59
- Grading Label: oM

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 5.1 (page 34) apply.

If you are requesting a regrade, this is the required subject line:

```
cis101 regrade oM lastname, firstname
```

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.

As always, your program must be properly linked to the student projects page, and the displayed webpage must show your name.

Strongly Suggested but 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 mondo print statement that prints everything needed, from content-type through the end.

Requirements:

(a) Your program must display and accept three or more input fields. Each must have a suitable description and a reasonable (non-blank) default value.

The first time your program runs, it must use your default values to construct the story.

(b) Your program must have a submit button. When the submit button is pressed, the screen should be redrawn. The input values should still be as entered. A story must be presented that uses the contents of the input fields.

(c) We provide the following subroutine to make this easier.

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 under “olin” in the index of the textbook for an explanation of this subroutine.

Since you will use this on several assignments, it might be worth your time to clean up the indenting errors that often come with cut 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. Then include one or more calls to olin as shown here.

You normally call olin with two parameters, like this:

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

Notice that `chomp` is not needed and is not used with olin. `Chomp` is used with `STDIN` from the keyboard.

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

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. This is probably not a good idea.

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. It can be very interesting to see what that input string really looks like.

8.11 oF: Farm 1

- Status: Officially Assigned.
- Discussed: Mon, Feb 10.
- 5pt Due Date: Wed, Feb 12, 13:00
- 4pt Due Date: Fri, Feb 14, 13:00
- 3pt Due Date: Tue, Apr 8, 23:59
- Grading Label: **oF**

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 5.1 (page 34) apply.

If you are requesting a regrade, this is the required subject line:

```
cis101 regrade oF lastname, firstname
```

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

Task: Print pictures of plants. Use a loop.

Requirements:

- (a) Include your name in an h1 at the top of the page.
- (b) Your program must display and accept a (numeric) input field. Use autofocus (required!) to place the cursor in that field. Optionally you can have more fields for more crops. Clear the numeric field between runs. Do not carry forward the latest entry.
- (c) Your program must have a submit button. When the submit button is pressed, the screen should be redrawn, followed by “n” pictures of your crop, where “n” is the number that was keyed into the input field.
- (d) The display size of your images cannot be bigger than 100 px wide and 100 px tall. You can use width=100 if you want.
- (e) Your loop must not exceed some specified limit of iterations. You can pick your limit between 5 and 10. If the user request is larger than your limit, you must complain and your loop must use your limit instead.

Suggestions:

- (a) Use the olin subroutine provided previously.
- (b) Pick something amusing for your crop and for the name of your garden.

8.12 g71: Array 1

- Status: Officially Assigned.
- Discussed: Wed, Feb 12.
- 5pt Due Date: Fri, Feb 14, 13:00
- 4pt Due Date: Wed, Feb 19, 13:00
- 3pt Due Date: Tue, Apr 8, 23:59
- Grading Label: **g71**

This is a GradeBot task. The general rules and explanations in section 5.3 (page 35) apply, including email subject line and program comment line.

`cis101 g71 lastname, firstname` is the required subject line.

`# cis101 g71 lastname, firstname` is the required comment line.

Gradebot can be found at <http://gradebot.tk/>
and at <http://gbot.dc.is2.byuh.edu/>.

There is a chapter in the textbook, probably about chapter 37, probably about page 195, that talks about arrays.

Summary: Use a loop and an array to remember and count names.

Start with an empty array. Prompt for and read in a name. Add the name to the array. Repeat until your input is blank. See how many names are in the array. Report that number.

I am grading on: Init, STDIN, Blank, Tally, Indexing, Style.

Init: You must initialize your array to be empty. I will check for that when you turn it in.

STDIN: You are allowed to have exactly one `<STDIN>` statement. It will be inside your main loop.

Blank: Repeat until your input is blank, but do not add that blank line to the array. When you are done, the size of the array will be equal to the number of names in the array.

Tally: You are **NOT** allowed to tally the names as you read them in. You must use the size of the array to see how many names are in the array. Tally means something like `x=x+1` or `x++` as you loop.

Indexing: Do not use indexing. Instead, use push/pop kinds of array manipulation.

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

Name Style: Your variable names must represent the thing that is stored in them. Do not call your variable “tally” if you are using it as a flag. Do not call your variable “x” if you can think of something more meaningful to call it, like “name”.

Sample Execution:

```
1: "Name? " (no \n)
in> ..... "Renaë"
2: "Name? " (no \n)
in> ..... "Lamar"
3: "Name? " (no \n)
in> ..... ""
4: "There were 2 names."
eof (end of output)
```

8.13 g72: Roll

- Status: Officially Assigned.
- Discussed: Fri, Feb 14.
- 5pt Due Date: Wed, Feb 19, 13:00
- 4pt Due Date: Fri, Feb 21, 13:00
- 3pt Due Date: Tue, Apr 8, 23:59
- Grading Label: **g72**

This is a GradeBot task. The general rules and explanations in section 5.3 (page 35) apply, including email subject line and program comment line.

`cis101 g72 lastname, firstname` is the required subject line.

`# cis101 g72 lastname, firstname` is the required comment line.

Gradebot can be found at <http://gradebot.tk/>
and at <http://gbot.dc.is2.byuh.edu/>.

There is a chapter in the textbook, probably about chapter 37, probably about page 195, that talks about arrays.

Summary: Add names to an array. Report how many names were added. Tell whether a specific name is in the list.

Requirements: (you must do it this way). Use push and foreach. Do not use indexing (like `$x[1]`). Approved style (spacing and indenting) is required for full credit.

Suggestions: (you do not have to do it this way). Use a flag or counter to tell whether you found the student.

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: "Who is attending? " (no \n)
in> ..... "Enoch"
2: "Who is attending? " (no \n)
in> ..... "Delano"
3: "Who is attending? " (no \n)
in> ..... ""
4: "There are 2 students present."
```

```
5: "Whom do you seek? " (no \n)
in> ..... "Delano"
6: "Delano is present."
eof (end of output)
```

8.14 oT: LocalTime

- Status: Officially Assigned.
- Discussed: Wed, Feb 19
- 5pt Due Date: Mon, Feb 24, 13:00
- 4pt Due Date: Wed, Feb 26, 13:00
- 3pt Due Date: Tue, Apr 8, 23:59
- Grading Label: oT

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 5.1 (page 34) apply.

If you are requesting a regrade, this is the required subject line:

```
cis101 regrade oT lastname, firstname
```

There is a chapter in the textbook, probably about chapter 41, probably about page 206, that talks about arrays and localtime.

Task: Create a dynamic webpage that, at a minimum, includes (a) your name, (b) the current hours:minutes:seconds, (c) the current day, month, year, with month as a word, not as a number, (d) the current day of the week as a word (not just a number).

(Note: There is another way to get the date and time. It is pre-formatted string like Fri Apr 16 12:34:56 2014. This will not be accepted. Use the array version of localtime.)

As an example, you can look at my oT webpage. You are welcome and encouraged to decorate your page, but you are not required to do so.

For assistance, you can look up “localtime” in the index of the text book. Or look up “gmtime” (Greenwich Mean Time). GMT is also called UTC.

You can do a Google search for “perl localtime” on the web.

The following suggestions might be helpful.

```
( $s, $m, $h, $d, $mo, $y, ... ) = localtime(time);
$y += 1900; # correct the year
@dow = ( "Sun", "Mon", ..., "Sat" );
```

8.15 g34: Celsius

- Status: Officially Assigned.
- Discussed: Mon, Feb 24.
- 5pt Due Date: Wed, Feb 26, 13:00
- 4pt Due Date: Fri, Feb 28, 13:00
- 3pt Due Date: Tue, Apr 8, 23:59
- Grading Label: **g34**

This is a GradeBot task. The general rules and explanations in section 5.3 (page 35) apply, including email subject line and program comment line.

`cis101 g34 lastname, firstname` is the required subject line.

`# cis101 g34 lastname, firstname` is the required comment line.

Gradebot can be found at <http://gradebot.tk/>
and at <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:
 $\text{celsius} = (\text{fahrenheit} - 32) * 5 / 9$

Final Exam Style is required. Follow the rules that I require for final exam programs. Specifically for this assignment I am looking at spacing and variable names.

Sample Execution:

```

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

```

8.16 g45: Afford

- Status: Officially Assigned.
- Discussed: Wed, Feb 26.
- 5pt Due Date: Fri, Feb 28, 13:00
- 4pt Due Date: Mon, Mar 3, 13:00
- 3pt Due Date: Tue, Apr 8, 23:59
- Grading Label: **g45**

This is a GradeBot task. The general rules and explanations in section 5.3 (page 35) apply, including email subject line and program comment line.

`cis101 g45 lastname, firstname` is the required subject line.

`# cis101 g45 lastname, firstname` is the required comment line.

Gradebot can be found at <http://gradebot.tk/> and at <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)
```

8.17 oHL: High Low

- Status: Officially Assigned.
- Discussed: Fri, Feb 28.
- 12pt Due Date: Mon, Mar 3, 13:00
- 10pt Due Date: Wed, Mar 5, 13:00
- 8pt Due Date: Fri, Mar 7, 13:00
- 6pt Due Date: Tue, Apr 8, 23:59
- Grading Label: **oHL**

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 5.1 (page 34) apply.

If you are requesting a regrade, this is the required subject line:

```
cis101 regrade oHL lastname, firstname
```

Summary: Program the high-low guessing game to run online. This requires a hidden field to hold the number being guessed.

This is a two-class activity, worth double points (10 points) plus bonus points (two more points) for being early.

This is the kind of thing that would actually be suitable as a final project for the class, except for the fact that we are doing it as a regular assignment.

I will automatically grade all programs at the 12pt and 10pt due dates. If you need your program graded after that, you should request it by email.

Requirement: Create a webpage (a) properly linked to the student projects page.

Requirement: It must include (b) your name, prominently displayed, (c) an autofocus field which is blank into which a guess is entered, (d) a hidden field with the number to be guessed, and (e) a submit button. (Not all browsers require a submit button but some do.)

Requirement: (f) When the program first starts, it must pick a number to be guessed, which must be between 1 and 100, inclusive. (g) If the guess is too low, it should say so. (h) If the guess is too high, it should say so. (i) If the guess is correct, it should say so and immediately (j) pick a new number to be guessed.

There will be NO loops in this program, and your only STDIN will be part of olin. (Look in the text book index for “memo to self”.)

Advice: While developing and testing your program, it is convenient to make the secret number into a regular (text) input field so you can see what is going on. After you get your program working, change it from a regular field into a hidden field.

You are welcome to steal the graphics from my own high-low program, or make your own, or do your program without graphics.

Optional Idea: Count how many guesses were made and make a clever comment based on the skill or luck of the player.

Reminder: When the answer is guessed, you must immediately pick a new answer.

8.18 oD2: Multi Dice

- Status: Officially Assigned.
- Discussed: Wed, Mar 5
- 5pt Due Date: Fri, Mar 7, 13:00
- 4pt Due Date: Mon, Mar 10, 13:00
- 3pt Due Date: Tue, Apr 8, 23:59
- Grading Label: **oD2**

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 5.1 (page 34) apply.

If you are requesting a regrade, this is the required subject line:

```
cis101 regrade oD2 lastname, firstname
```

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.

Link to the oD2 link on the student projects page.

Requirements:

- (a) Include your name in an h1 at the top of the page.
- (b) Your program must display a field and accept a numeric input. Each time the page is drawn, the blank must be empty and have the cursor (autofocus) in it. Do not carry forward the latest entry.
- (c) Near your input field, you must say what your maximum number of dice is.
- (d) 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.
- (e) Your loop must not exceed 100 iterations. You can pick a smaller limit if you wish, but it must be at least 10. If the user request is larger than your limit, your loop must use your limit instead.

(f) Each image should have a displayed size that is no larger than 100px wide and 100px tall. You can use `width=` and `height=` if you like, or you can resize your images.

Recommended but Optional:

- (a) Have a title as part of the head of your webpage.
- (b) If too many dice are requested, print a suitable warning message.
- (c) Instead of normal, six-sided dice you can use something else, but it must have at least four alternatives from which one is selected, and the image size must be no larger than 100 by 100.

8.19 oF2: Farm 2

- Status: Officially Assigned.
- Discussed: Wed, Mar 12 and Fri, Mar 14
- 12pt Due Date: Fri, Mar 14, 13:00
- 10pt Due Date: Mon, Mar 17, 13:00
- 8pt Due Date: Wed, Mar 19, 13:00
- 6pt Due Date: Tue, Apr 8, 23:59
- Grading Label: **oF2**

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 5.1 (page 34) apply, including email subject line and program comment line.

```
cis101 oF2 lastname, firstname
```

 is the required subject line.

```
# cis101 oF2 lastname, firstname
```

 is the required comment line.

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

This is a two-class activity, worth double points (10 points) plus bonus points (two more points) for being early.

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

Your task includes writing two subroutines: `plant` is one and `harvest` is the other. These subroutines will call `olin` as needed and do all the printing required to accomplish their tasks.

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.

8.19.1 Main Program: Crops

You must have **at least four crops**, but you can have more if you wish.

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.

```
@crops = ( ... );
```

The text strings in @crops must be usable for (a) display labels, for (b) input names, and for (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.

8.19.2 Main Program: Call "plant"

You must use the following foreach loop, or one like it, to display the names and quantities of the crops.

```
foreach $crop ( @crops ) { plant ( $crop ) }
```

Then include an appropriate "submit button."

8.19.3 Subroutine "Plant"

Do not use any global variables.

Write a subroutine named `plant` that does the following:

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.

8.19.4 Main Program: Harvesting Call

You must use the following foreach loop, or one like it, to display the harvest.

```
foreach $crop ( @crops ) { harvest ( $crop ) }
```

8.19.5 Subroutine “Harvest”

Do not use any global variables.

Print a line telling what the requested crop and quantity are. (The name of the crop comes from the subroutine’s parameter list. The quantity comes from a call to `olin`.)

After printing it, if the quantity is unreasonably large, complain, and convert it to a smaller number. Example: if 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\">
```

8.20 g43: Leap Year

- Status: Officially Assigned.
- Discussed: Wed, Mar 19.
- 5pt Due Date: Fri, Mar 21, 13:00
- 4pt Due Date: Mon, Mar 24, 13:00
- 3pt Due Date: Tue, Apr 8, 23:59
- Grading Label: **g43**

This is a GradeBot task. The general rules and explanations in section 5.3 (page 35) apply, including email subject line and program comment line.

`cis101 g43 lastname, firstname` is the required subject line.

`# cis101 g43 lastname, firstname` is the required comment line.

Gradebot can be found at <http://gradebot.tk/>
and at <http://gbot.dc.is2.byuh.edu/>.

Task: Tell whether a given year is a leap year or not.

If a year has Feb 29, then it is a leap year. Tell whether a year is leap year or not. If the year is a multiple of 4, then it is. Unless if it is a multiple of 100; then it is not. Unless if it is a multiple of 400; then it is.

You can do this with a single if statement that uses ands and ors.

You can do this with an if, elsif, else approach.

You are also welcome to use some other approach.

Helpful hint: The remainder operator, also called modulus, is represented by the percent sign. `11 % 3` means the remainder when eleven is divided by three, and the answer is two. (Three goes into eleven three times, with a remainder of two.) See the text book for more information. Look up “remainder” in the index.

Sample Executions:

GradeBot would have engaged your program in this dialog:

```
1: "What is the year? " (no \n)
in> ..... "2011"
2: "2011 is not a leap year."
eof (end of output)
```

GradeBot would have engaged your program in this dialog:

```
1: "What is the year? " (no \n)
in> ..... "2012"
2: "2012 is a leap year."
eof (end of output)
```


8.21 g64: Factors

- Status: Officially Assigned.
- Discussed: Mon, Mar 24.
- 5pt Due Date: Fri, Mar 28, 13:00
- 4pt Due Date: Mon, Mar 31, 13:00
- 3pt Due Date: Tue, Apr 8, 23:59
- Grading Label: **g64**

This is a GradeBot task. The general rules and explanations in section 5.3 (page 35) apply, including email subject line and program comment line.

`cis101 g64 lastname, firstname` is the required subject line.

`# cis101 g64 lastname, firstname` is the required comment line.

Gradebot can be found at <http://gradebot.tk/>
and at <http://gbot.dc.is2.byuh.edu/>.

Task: Find the factors of a number.

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)
```

8.22 g75: Tally

- Status: Officially Assigned.
- Discussed: Fri, Mar 28.
- 5pt Due Date: Mon, Mar 31, 13:00
- 4pt Due Date: Wed, Apr 2, 13:00
- 3pt Due Date: Tue, Apr 8, 23:59
- Grading Label: **g75**

This is a GradeBot task. The general rules and explanations in section 5.3 (page 35) apply, including email subject line and program comment line.

`cis101 g75 lastname, firstname` is the required subject line.

`# cis101 g75 lastname, firstname` is the required comment line.

Gradebot can be found at <http://gradebot.tk/>
and at <http://gbot.dc.is2.byuh.edu/>.

Task: Add a list of numbers.

The numbers are provided as a space-separated string. You can use “split” to create a list. You can use “foreach” to walk down the list. You can use “+=” to add each item to a running total.

There are other approaches that could also be successful.

Sample Executions:

GradeBot would have engaged your program in this dialog:

```
1: "Numbers: " (no \n)
in> ..... "5 3 1 6"
2: "The total is 15."
eof (end of output)
```

8.23 oB: Boring

- Status: Officially Assigned.
- Discussed: Mon, Mar 31.
- 5pt Due Date: Wed, Apr 2, 13:00
- 4pt Due Date: Fri, Apr 4, 13:00
- 3pt Due Date: Tue, Apr 8, 23:59
- Grading Label: **oB**

Summary: It's Dinner Time. Ask what is for dinner. If you have had it before, then it is boring. If not, then it is yummy.

The user input field must be autofocus and big enough to type “spaghetti” (or bigger).

There must be a submit button.

Suggestions: (you do not have to do it this way). Pass a hidden field that contains the dinner history. Create this history using a join command. Parse this history using a split command. Use the history to decide whether the next meal is boring or not.

8.24 oJS: JavaScript

- Status: Officially Assigned.
- Discussed: Wed, Apr 2.
- 5pt Due Date: Fri, Apr 4, 13:00
- 4pt Due Date: Mon, Apr 7, 13:00
- 3pt Due Date: Tue, Apr 8, 23:59
- Grading Label: **oJS**

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 5.1 (page 34) apply.

If you are requesting a regrade, this is the required subject line:

```
cis101 regrade oJS lastname, firstname
```

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. For five-point credit, you must add another button, like maybe square root, or $a^2 + b^2$, hopefully something meaningful. For four-point or three-point credit, this is not required.

Chapter 9

Exam Questions

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

Contents

9.1	q1: String Basic	93
9.2	q2: Number Basic	94
9.3	q3: Number Story	94
9.4	q4: Number Decision	95
9.5	q5: Number Decision Story	95
9.6	q6: String Decision	96
9.7	q7: String Bracket	96
9.8	q8: Repeat While	97
9.9	q9: Repeat For	98
9.10	q10: Repeat Last	98
9.11	q11: Repeat Nested	98
9.12	q12: List Basic	99
9.13	q13: List Loop	100
9.14	q14: Array Basic	100
9.15	q15: Array Loop	101
9.16	q16: Array Split	101
9.17	q17: Array Join	101
9.18	Subroutine Basics	102
9.19	q18: Subroutine Return	102
9.20	q19: Positional Parameter	103

9.21 q20: Globals and Locals	103
9.22 q21: Variable Number of Parameters	104

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.

9.1 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.

9.2 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.

9.3 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.

9.4 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.

9.5 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.

9.6 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 $y`) is a frequent typo for eq.
- (c) Properly quote your literal strings. (See barewords in the text book.)
 - (c1) (`$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.

9.7 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 text book.)
 - (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.

9.8 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.

9.9 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.

9.10 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.

9.11 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.

9.12 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";`

9.13 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 $_ }`

9.14 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.

9.15 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] }`

9.16 q16: Array Split

GradeBot Examples: g70 series, specifically g74, g75, g76.

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)

9.17 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)

9.18 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.

9.19 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.

9.20 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.

9.21 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`.

9.22 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.

Chapter 10

Final Projects

- Status: Officially Assigned.
- Discussed:
- Deadline: Tue, Apr 8, 23:59

10.1 As Stated in the Syllabus

(40) Project Points

10 Project CGI: write a dynamic web page

10 Project Pictures: use img tags

10 Project Multi Input: process multiple inputs

10 Project Hidden Fields: pass state (counter, etc)

The final project is due by 23:59 on Tuesday, the day after the last day of class. I plan to grade it early on Wednesday unless you have asked me to grade yours earlier.

Project points are earned for performance on out-of-class work. The project must be your own work. It should be fun. A game would be ideal. You are allowed to consult with others including websites but you are not allowed to cut and paste code written by others. Each online screen must clearly identify you as the author. It must accept user input. It should utilize hidden fields (state) that are needed for its operation.

Your final project cannot just be something we did in class. The

in-class activities are good examples, and teach good principles, but they do not demonstrate understanding or creativity. If your project is based on something we did in class, it must go beyond it in some substantial and significant way.

<http://dc.is2.byuh.edu/cis101.2141/> is the place to link your project. It is the Student Projects page for this class. Link it to the “proj” slot.

10.2 Additional Details

Doing a project is a great way for you to become empowered. Our nominal goal is that each student be able to build something fun and useful. The real goal is to enrich the student by giving them the ability to create the programs they want or need without always relying on others.

Final projects must be different from things we did in class. They can be similar, but should have at least a few fundamental improvements or changes. Simply using a different picture or different words is not enough. It must have different logic.

How to Submit It

Final projects must be linked to my student projects page, proj column, which links to your /proj/ directory.

I will automatically check for all projects soon after the deadline, so you do not need to tell me that you are doing a project or not.

However, if you want me to review and possibly grade your final project early, you can send me an email. Use the following subject line, or something very close to it.

Subject Line: `cis101 final project, lastname, firstname`

It will speed things up for both of us if you could please include a clickable link to your project right in your email. It is likely to get you a faster response.

Size

What is the right size for a project at this point in your skills development? This unit contains a few pre-defined projects that could be appropriate for demonstrating and improving your programming skills. They are given as examples. They have served in the past as actual assignments.

Invent a Project

Doing pre-defined projects is often boring and can lead to some inappropriate sharing of code. This does not enhance learning. So instead here is a list of requirements that your project should satisfy.

Online: For any credit at all, your program must run online as a web application. Anyone in the world should be able to run your program. This part is absolutely required.

Authorship: The code comments and the program output (webpages) should clearly identify you as the author and owner of the program.

Creative: Do something creative and unique. If it looks like the project your neighbor already turned in, it might not qualify. If it is too similar to something we did in class, it would not qualify.

Fun: Your program should be fun. A game would be ideal. Fun is a subjective judgment, so we will trust you on this. If you think it is fun, we will agree that it is fun.

Images: For maximum credit, your program must appropriately use pictures, typically by way of an HTML `` statement. Ideally the pictures would change depending, for example, on the progress of the game.

Multi Input: For maximum credit, your program must accept multiple inputs, for example buttons or text fields, to allow the user to interact with it. Hidden fields count as inputs. All your submit buttons except the first count as inputs if they each do something different. Your first submit button does not count.

State: For maximum credit, your program must have some sort of meaningful state that it carries forward. Some or all of the state must be carried in hidden fields that are actually important to your program's operation. It could be a counter or a running total or anything else that is "state." And

hidden fields count towards the multiple input requirement.

Appendix A

Spelling

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 carefully.

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.

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.

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.

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 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 my list.

themselves - 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 themselves as a word, and instead suggest themselves. For plural them, this would be correct, but for singular them, themselves 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 traditionally 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.

Index

- cis101.2141, [53](#)
- Afford, [77](#)
- Array 1, [71](#)
- Before After, [62](#)
- Birthday, [64](#)
- Boring, [90](#)
- Celsius, [76](#)
- cM, [61](#)
- command-line rules, [35](#)
- daily update, [28](#)
- Dice, [59](#)
- Email rules, [37](#)
- Factors, [88](#)
- Farm 1, [70](#)
- Farm 2, [83](#)
- First Webpage, [53](#)
- g21, [56](#)
- g31, [62](#)
- g34, [76](#)
- g41, [63](#)
- g42, [64](#)
- g43, [86](#)
- g45, [77](#)
- g51, [66](#)
- g64, [88](#)
- g71, [71](#)
- g72, [73](#)
- g75, [89](#)
- GradeBot rules, [35](#)
- Hi Fred, [56](#)
- High Low, [79](#)
- JavaScript, [91](#)
- Leap Year, [86](#)
- LocalTime, [75](#)
- Mad Lib, [61](#), [67](#)
- Multi Dice, [81](#)
- Numeric Decision, [63](#)
- o1, [53](#)
- oB, [90](#)
- oD, [59](#)
- oD2, [81](#)
- oF, [70](#)
- oF2, [83](#)
- oHL, [79](#)
- oJS, [91](#)
- olin subroutine, [68](#)
- oM, [67](#)
- online rules, [34](#)
- oR, [57](#)

oT, [75](#)

Phone Book, [66](#)

project, [105](#)

Random Number, [57](#)

rich text, [36](#), [38](#)

Roll, [73](#)

style, [45](#)

Syllabus, [3](#)

Tally, [89](#)