CIS 101 Study Guide
Fall 2013

Don Colton
Brigham Young University–Hawai‘i

January 7, 2014

Study Guide

This is the official study guide for the CIS 101 class, Introduction to Programming, as taught by Don Colton, Fall 2013. It is focused directly on the grading of the course.

`http://byuh.doncolton.com/cis101/2135/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/cis101/2135/syl.pdf` is the official syllabus for this course. It is largely reproduced in Chapter 2 (page 8) 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

Chapter 1

Overview

Contents

For many this is a really fun class. For some this is a really difficult class.

It is a foundational step in developing your ability to serve those around you by giving them better ways to use their computers.

And it can be the gateway to automating some of the (mind-numbing) tasks that can be involved with things like computer systems administration.

We build web-based programs that you can share through the Internet with anyone in the world: friends, family, anybody. And we develop skills you can use in later classes and the work place.

The **textbook** is online free. You can download the PDF from `http://ipup.doncolton.com/`

We will use the Third Edition. I often make small improvements, like spelling corrections, throughout the semester. Once in a while I make bigger changes, like whole new sections or chapters. So printing it out is not really recommended. Why kill a tree? Reading a fresh copy on your computer would be ideal, if that works for you.

I try hard to not lecture much in class. (Sometimes I end up lecturing anyway.) The book contains my lectures. You read them outside of class. In class we may review parts of the reading and I always give you a chance to ask questions.

Most time in class is spent actually making things. I go over parts of the textbook to introduce activities, but there is lots more in the book that you are expected to read on your own.

Based on past experience, everyone who regularly attends will pass the class. (The F grades usually only happen to people who quit coming to class.)

Most students do a project. The study guide has official project details.

1.1 So, What is Programming?

First the bad news. Computers are pretty stupid. You have to give them simple steps to follow.

Now the good news. Computers are fast, reliable, and cheap. They don't get offended, go on strike, call in sick, or take vacation.

Many interesting tasks can be built up out of the simple steps that computers can perform. For these reasons, even though they are pretty stupid, computers are very popular.

The art of programming is the art of converting useful activities into simple steps that a computer can perform.

Our programming language will be Perl.

1.2 Course Learning Objectives

This study guide and the accompanying textbook are intended to meet the following 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
- decisions (if, else, elsif)
- looping (while, for, foreach)
- array and list variables
- subroutines

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

In covering those skills, I also teach the following:

- dynamic webpage creation
- dynamic response to webpage inputs

1.3 Sending Me Email

I have "Email Submission Rules" listed in section 4.4 (page 69). But similar rules apply to all emails related to this class.

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

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

Rule Three: Single-topic emails get answered faster than multi-topic emails do. Avoid combining several topics in a single email.

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

My solution is to have you put **cis101** in your email subject line, preferably as the first word. If you do this, my email system will immediately and automatically respond to you, telling you that I got your email and it is in my queue.

If you fail to do this, you will not get an immediate reply and your email will end up in some other queue in my work flow. Your email will not be noticed when I am grading for this class. In the best case I will read your email and ask you to send it again with the correct word in the subject line. In the worst case your email will be in my spam folder and I will never even see it.

You have been warned.

1.4 Reporting Your Study Time

Once a week I will ask for your study time as part of the daily update.

For study during the very last week, which includes the final exam, you can report on the day of the final exam.

If you do not report in some other way, you can report by sending me the details by email.

With email, use this as your subject line:

```
cis101 study time lastname, firstname
```

Inside the email, say something like: "For the week of (starting month and day) to (ending month and day), I studied (how many) hours." Be specific about which week it is.

1.5 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. He runs the IS2 machine where our cPanel accounts are stored.

1.6 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 intricate 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.

Chapter 2

Syllabus Extracts

The syllabus is intended to be stable and reliable. I publish the syllabus at the start of the semester, and then I do not change it in any way, except in case of extreme emergency.

The study guide, on the other hand, gets changed and updated throughout the semester.

This chapter of the study guide simply repeats, for your convenience, material given in the syllabus. In case it differs from the syllabus, the syllabus itself is always correct and authoritative, and this study guide is probably out of date.

2.1 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.

2.1.1 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.

2.1.2 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.

2.2 Course Details

2.2.1 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, Sep 9 to Mon, Dec 8
- **Class Time:** MWF 12:10 to 13:10
- **Final Exam:** Fri, Dec 13, 13:00–15:50

2.2.2 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.

2.2.3 About the Instructor

- **Instructor (me):** Don Colton
- **My email:** doncolton2@gmail.com
- **My Office:** GCB 128
- **Office Hour:** MWF 13:10 to 13:40.

I have reserved GCB 111 on MWF 13:20 to 14:20 as "CIS 101L" so my students and others can study in a lab setting and meet with each other and with me. The room is available as an Open Lab for your use either individually or in groups, for my class or for other classes. MWF 13:10 to 13:40 I will be present in GCB 111 or in my office to assist students that come.

2.2.4 About the Study Guide

I provide a study guide for this course. The study guide provides current details and specific helps for each assignment. It provides guidance for taking the exams.

http://byuh.doncolton.com/cis101/2135/sguide.pdf has the study guide for this course.

The study guide will be updated regularly throughout the semester, as new assignments are given.

The study guide provides details about (what I consider to be) my very liberal policy on regrading and late work. I have a liberal policy because I want every student to complete every assignment and learn everything that was involved. And I know that things come up to prevent you from doing everything on time.

2.3 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

2.4 Grading

Here is the actual grade distribution from Fall 2012: (35 students): grade average 2.69, 4.0x13 3.7 3.0x4 2.7x3 2.4x2 2.0x2 1.7x4 1.4 0.7x2 0.0x3 (Wx2).

Here is the actual grade distribution from Winter 2013: (24 students): grade average 2.22, 4.0x6 3.7 3.0x5 2.7x3 2.4 0.0x8.

Grading is based on Effort (30.0%), Activities (13.5%), Final Project (4.0%), and Exams (52.5%). Plus there is about 10% extra credit available.

2.4.1 Grading Scale

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

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.

2.4.2 Tracking Your Grade

I keep an online gradebook 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.

2135 CIS 101 Overall Grade Book: This includes the totals from Activity and Effort and adds your exam performance. It also shows your final grade.

2135 CIS 101 Effort Grade Book: This tracks the daily updates and study time.

2135 CIS 101 Activities Grade Book: This tracks your performance on in-class activities.

2.4.3 Effort: Daily Update (50 points)

Each day in class starts with the "daily update" (DU). It is my way of reminding you of due dates and deadlines, sharing updates and news, and taking roll. It is your way of saying something anonymously to each other and to me. It must be taken in class 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 about 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.

2.4.4 Effort: Study Time (250 points)

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 x 18 = 252. 14 x 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.

2.4.5 Effort Points are Optional

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

But when I calculate your final grade, I do it two 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 both 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.

2.4.6 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 x 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.

2.4.7 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/cis101.2135/` 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.

2.4.8 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.

2.4.9 Other Extra Credit

Report an error in the published materials I provide. In this class, they 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.)

2.5 Calendar

We meet about 37 times plus the final.

See the study guide for dates and deadlines.

2.5.1 Special Dates

Mo	Sep	09	First Day of Instruction
Fr	Sep	20	**exam 0** practice test
Fr	Sep	27	**exam 1**
Fr	Oct	11	**exam 2**
Fr	Oct	25	**exam 3**
Fr	Nov	08	ISECON, No Class
Mo	Nov	11	**exam 4**
We	Nov	20	EIL Program Review, No Class
Fr	Nov	22	**exam 5**
Fr	Nov	29	Thanksgiving Friday, No Class
Fr	Dec	06	**exam 6**
Mo	Dec	09	Last Day of Instruction
Tu	Dec	10	Project Deadline (23:59)
Fr	Dec	13	**exam 7** Final Exam, 13:00–15:50

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.**

2.5.2 Excused Absences

You can see that I have built a bit of slack into the grading so you can miss a few days (or assignments) if you need to, and still earn an A. Taking a friend to the airport? Taking your spouse or child to the doctor? Taking a field trip for another class? No problem. You are excused.

If you have to miss an exam, since there are several exams and they have identical content, my advice is to study harder for one of the other exams. If you have to miss the last final, that's a bit more difficult, but it is still possible to do all the problems without taking all the exams.

Beyond that I do not offer special treatment to anyone except in HIGHLY unusual situations.

2.6 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.

2.6.1 Office Hour / Open Lab

I have reserved GCB 111 on MWF 13:20 to 14:20 as "CIS 101L" so my students and others can study in a lab setting and meet with each other and with me. The room is available as an Open Lab for your use either individually or in groups, for my class or for other classes. MWF 13:10 to 13:40 I will be present in GCB 111 or in my office to assist students that come.

The CIS department operates an open lab with tutors in GCB 111 most afternoons and evenings.

2.6.2 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.

A good time for your study group to meet is immediately after class. Eat lunch together (carefully) and work on the class activities.

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.

2.6.3 Tutoring

The CIS department provides tutoring in GCB 111, Monday through Friday, typically starting around 5 PM and ending around 11 PM (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 use 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 use a tutor is to show them something in the textbook and ask about it.

The worst way to use 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.

If you still need help, please come and see me, even outside my posted office hours.

2.7 BYUH Learning Framework

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

2.7.1 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.

2.7.2 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.

2.7.3 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.

2.8 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.

2.8.1 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.

2.8.2 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, you are invited to contact the Students With Special Needs Coordinator at 808-675-3518. Reasonable academic accommodations are made for all students who have qualified documented disabilities.

2.8.3 Plagiarism

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 just 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.

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.

2.8.4 Sexual Harassment

BYUH's policy against sexual harassment complies with federal Title IX of the Education Amendments of 1972 to protect university students from student-to-student sexual harassment both in and out of the classroom setting. Any incidents of such student-to-student harassment should be reported to either the Director of Human Resources (808-675-3713) or the Honor Code Office (808-675-3531). Allegations of sexual harassment are taken seriously. Upon receiving a report of sexual harassment, the Director of Human Resources will take appropriate action to resolve and correct conditions resulting from individual perceptions or from inappropriate behavior.

Chapter 3

Activities Assigned

Contents

This is a list of the activities that have been assigned. They are listed in the order they were discussed in class.

As new tasks are assigned, they will be added to this chapter.

The purpose of these activities is to give you a way to develop and test your programming skills.

3.1 o1: First Webpage

- Status: Officially Assigned.
- Discussed: Mon, Sep 9.
- 5pt Due Date: Mon, Sep 9, 17:00
- 4pt Due Date: Fri, Sep 13, 23:59
- 3pt Due Date: Tue, Dec 10, 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 4.1 (page 67) apply.

If you are requesting a regrade, this is the required subject line:
`cis101 regrade o1 lastname, firstname`

Log into cPanel. Go into File Manager.

Go into your `public_html` directory (folder).

Create a directory (folder) named `cis101.2135` The cis101 part stands for this class. The 2135 part stands for this semester.

In that directory create a file named `index.html`. Be careful that you do not create `INDEX.HTML` or anything else that is not exactly `index.html`.

Construct a webpage. Make sure you at least have this minimum content: head, title, body, h1, image. Make sure you do not have any HTML errors.

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/cis101.2135/`

3.2 g21: Hi Fred

- Status: Officially Assigned.
- Discussed: Mon, Sep 9.
- 5pt Due Date: Wed, Sep 11, 14:00
- 4pt Due Date: Fri, Sep 13, 23:59
- 3pt Due Date: Tue, Dec 10, 23:59
- Grading Label: **g21**

This is a GradeBot task. The general rules in section 4.3 (page 68) 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/`.

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)
```

More information can be found by looking at the text book for this class.

3.3 oR: Random Number

- Status: Officially Assigned.
- Discussed: Wed, Sep 11.
- 5pt Due Date: Fri, Sep 13, 16:00
- 4pt Due Date: Mon, Sep 16, 16:00
- 3pt Due Date: Tue, Dec 10, 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 4.1 (page 67) apply.

If you are requesting a regrade, this is the required subject line:
`cis101 regrade oR lastname, firstname`

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

Warning: I am going to give you some information. 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 cis101.2135 folder. Create a sub-folder named "random" and in it create a file named "index.cgi".

This file, "index.cgi", 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:

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

The 0755 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>
</head><body>
<h1>Don's Random Number Generator</h1>
<h2>$random</h2>
</body>
```

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/

3.4 oD: Dice

- Status: Officially Assigned.
- Discussed: Fri, Sep 13.
- 5pt Due Date: Mon, Sep 16, 16:00
- 4pt Due Date: Wed, Sep 18, 16:00
- 3pt Due Date: Tue, Dec 10, 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 4.1 (page 67) apply.

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

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

Summary: Make a web program that rolls two (or more) 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 named "dice" and in it create a file named "index.cgi".

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>
<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.

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 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. But please do have at least six image choices, and at least two displayed at a time.

3.5 cM: Mad Lib

- Status: Officially Assigned.
- Discussed: Mon, Sep 16.
- 5pt Due Date: Wed, Sep 18, 16:00
- 4pt Due Date: Fri, Sep 20, 16:00
- 3pt Due Date: Tue, Dec 10, 23:59
- Deadline: Tue, Jan 15, 23:59
- Grading Label: **cM**

This is a command-line task. The general rules in section 4.2 (page 68) 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.

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.

3.6 g41: Numeric Decision

- Status: Officially Assigned.
- Discussed: Wed, Sep 18.
- 5pt Due Date: Fri, Sep 20, 16:00
- 4pt Due Date: Mon, Sep 23, 16:00
- 3pt Due Date: Tue, Dec 10, 23:59
- Grading Label: **g41**

This is a GradeBot task. The general rules in section 4.3 (page 68) 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/`.

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)
```

More information can be found by looking in chapter 20 (and previous chapters) of the text book for this class.

3.7 g42: Birthday

- Status: Officially Assigned.
- Discussed: Mon, Sep 23.
- 5pt Due Date: Wed, Sep 25, 16:00
- 4pt Due Date: Fri, Sep 27, 16:00
- 3pt Due Date: Tue, Dec 10, 23:59
- Grading Label: **g42**

This is a GradeBot task. The general rules in section 4.3 (page 68) 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/`.

Chapter 25 (page 128) of the text book talks about and, or, not, and related operations that might be helpful with this task.

Task: Tell how many years old a person is. Assume today is Feb 25, 2013. This will probably involve several if statements.

Sample Execution:

```
  1:  "Please enter your name: " (no \n)
in>    ........................"Michelle"
  2:  "What month (number) were you born? " (no \n)
in>    ...................................."10"
  3:  "What day were you born? " (no \n)
in>    ........................"25"
  4:  "What year were you born? " (no \n)
in>    ........................."1984"
  5:  "Ah. You were born on 1984-10-25."
  6:  ""
  7:  "Michelle, you are 28 years old."
eof  (end of output)
```

Note: Line 5 was added after the assignment was originally given.

3.8 g51: Phone Book

- Status: Officially Assigned.
- Discussed: Wed, Sep 25.
- 5pt Due Date: Fri, Sep 27, 16:00
- 4pt Due Date: Mon, Sep 30, 16:00
- 3pt Due Date: Tue, Dec 10, 23:59
- Grading Label: **g51**

This is a GradeBot task. The general rules in section 4.3 (page 68) 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/`.

Chapter 26 (page 132) of the text book 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)
```

3.9 g34: Celsius

- Status: Officially Assigned.
- Discussed: Mon, Sep 30.
- 5pt Due Date: Wed, Oct 02, 16:00
- 4pt Due Date: Fri, Oct 04, 16:00
- 3pt Due Date: Tue, Dec 10, 23:59
- Grading Label: **g34**

This is a GradeBot task. The general rules in section 4.3 (page 68) 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/`.

Task: Convert temperature from Fahrenheit to Celsius. The formula is: celsius = (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)
```

3.10 oM: Mad Lib

- Status: Officially Assigned.
- Discussed: Wed, Oct 02
- 5pt Due Date: Mon, Oct 07, 16:00
- 4pt Due Date: Wed, Oct 09, 16:00
- 3pt Due Date: Tue, Dec 10, 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 4.1 (page 67) apply.

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

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

Task: Make an online Mad Lib story generator.

Advice: Review chapter 10 of the text book.

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. Look under "olin" in the index of the textbook for an explanation of this subroutine.

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.

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.

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.

3.11 oF: Farm

- Status: Officially Assigned.
- Discussed: Mon, Oct 07
- 5pt Due Date: Wed, Oct 09, 16:00
- 4pt Due Date: Fri, Oct 11, 16:00
- 3pt Due Date: Tue, Dec 10, 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 4.1 (page 67) apply.

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

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

Task: Print pictures of plants. Use a loop.

Requirements:

(a) Your program must display and accept a (numeric) input field. Use autofocus (required!) to place the cursor in that field.

(b) 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.

(c) 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.

(d) Your loop must not exceed some specified limit of iterations. You can pick your limit in the range of 4 through 9 iterations. If the user request is larger than your limit, your loop must use your limit instead.

Strongly Suggested (Should have been required):

(a) Include your name in an h1 at the top of the page.

(b) Clear the numeric field between runs. Do not carry forward the latest entry.

Suggestions:

(a) Use the olin subroutine provided previously.

(b) Pick something amusing for your crop and for the name of your garden.

3.12 g61: While

- Status: Officially Assigned.
- Discussed: Mon, Oct 14
- 5pt Due Date: Wed, Oct 16, 16:00
- 4pt Due Date: Fri, Oct 18, 16:00
- 3pt Due Date: Tue, Dec 10, 23:59
- Grading Label: **g61**

This is a GradeBot task. The general rules in section 4.3 (page 68) apply, including email subject line and program comment line.

`cis101 g61 lastname, firstname` is the required subject line.

`# cis101 g61 lastname, firstname` is the required comment line.

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

Task: Use a while loop to print numbers.

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

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

Note: if you do not hit the ending point exactly, that's okay. But do not go past the ending point.

Approved spacing and indenting is required for full credit.

Sample Execution:

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

```
 8:  "5"
 9:  "6"
10:  "7"
11:  "8"
12:  "Done!"
eof  (end of output)
```

3.13 oD2: Multi Dice

- Status: Officially Assigned.
- Discussed: Wed, Oct 16
- 5pt Due Date: Fri, Oct 18, 16:00
- 4pt Due Date: Mon, Oct 21, 16:00
- 3pt Due Date: Tue, Dec 10, 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 4.1 (page 67) apply.

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

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

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

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, 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 make it 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.

3.14 g71: Array 1

- Status: Officially Assigned.
- Discussed: Fri, Oct 18
- 5pt Due Date: Mon, Oct 21, 16:00
- 4pt Due Date: Wed, Oct 23, 16:00
- 3pt Due Date: Tue, Dec 10, 23:59
- Grading Label: **g71**

This is a GradeBot task. The general rules in section 4.3 (page 68) 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/`.

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

Start with an empty array named XYZ. 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.

Note: Add names to the array. Do not add the blank line to the array. The size of the array will be equal to the number of names in the array.

Your array must be named XYZ. You must initialize it to be empty.

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

Do **NOT** 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.

Note: Because you are not allowed to tally the names as you read them in, it would be very difficult to do this using indexing. I recommend using push/pop kinds of array manipulation.

Approved spacing and indenting is required for full credit.

Sample Execution:

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

3.15 g72: Roll

- Status: Officially Assigned.
- Discussed: Mon, Oct 21
- 5pt Due Date: Wed, Oct 23, 16:00
- 4pt Due Date: Fri, Oct 25, 16:00
- 3pt Due Date: Tue, Dec 10, 23:59
- Grading Label: **g72**

This is a GradeBot task. The general rules in section 4.3 (page 68) 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/`.

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)
```

3.16 g45: Afford

- Status: Officially Assigned.
- Discussed: Wed, Oct 23
- 5pt Due Date: Fri, Oct 25, 16:00
- 4pt Due Date: Mon, Oct 28, 16:00
- 3pt Due Date: Tue, Dec 10, 23:59
- Grading Label: **g45**

This is a GradeBot task. The general rules in section 4.3 (page 68) 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 difficult one. Read it carefully and think about it carefully.

Objective: Learn how to use if/else skillfully.

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

Task: You are shopping for wedding gifts for a good friend. They have registered their wants one 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)
```

3.17 g44: Phone

- Status: Officially Assigned.
- Discussed: Mon, Oct 28
- 5pt Due Date: Wed, Oct 30, 16:00
- 4pt Due Date: Fri, Nov 01, 16:00
- 3pt Due Date: Tue, Dec 10, 23:59
- Grading Label: **g44**

This is a GradeBot task. The general rules in section 4.3 (page 68) apply, including email subject line and program comment line.

`cis101 g44 lastname, firstname` is the required subject line.

`# cis101 g44 lastname, firstname` is the required comment line.

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

Task: Compare the cost of two phone cards for a telephone call.

Format the amounts to dollars and cents. I recommend that you use sprintf. See also printf (appendix C) in the text book.

```
  $savings = sprintf ( "%.2f", $savings );
  print "blah blah \$$savings blah blah\n";
```

Base your decisions on the formatted amounts. Because of "round-off" problems, two unformatted numbers can be unequal even if they actually should be equal. Example: 1/3=.333, (1/3)*3=.999.

Sample Execution:

```
GradeBot would have engaged your program in this dialog:
  1:  "Welcome to the Prepaid Phone Card Analysis Program"
  2:  ""
  3:  "Tell me about the available phone cards and your calling habits."
  4:  "I'll tell you the best card to use."
  5:  ""
  6:  "Enter estimated call duration: " (no \n)
 in>   ..............................."5"
  7:  ""
  8:  "For the first card"
  9:  "  Enter connect cost per call: $" (no \n)
 in>   ................................"0"
 10:  "  Enter additional cost per minute: $" (no \n)
 in>   .....................................".07"
 11:  "The cost would be $0.35 if you use the first card."
 12:  ""
 13:  "For the second card"
 14:  "  Enter connect cost per call: $" (no \n)
 in>   ................................".49"
 15:  "  Enter additional cost per minute: $" (no \n)
 in>   .....................................".03"
 16:  "The cost would be $0.64 if you use the second card."
 17:  ""
 18:  "You would save $0.29 by using the first card."
 19:  ""
 20:  "Thank you for using the Prepaid Phone Card Analysis Program."
 eof  (end of output)
```

3.18 cHL: High Low Command-Line

- Status: Officially Assigned.
- Discussed: Wed, Oct 30
- 5pt Due Date: Fri, Nov 01, 14:00
- 4pt Due Date: Mon, Nov 04, 14:00
- 3pt Due Date: Tue, Dec 10, 23:59
- Grading Label: **cHL**

This is a command-line task. The general rules in section 4.2 (page 68) apply, including email subject line and program comment line.

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

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

If you get this working while I am in the classroom, please come up and show me so I can grade it immediately. This is better than emailing it to me.

Summary: Program the high-low guessing game to run at the command line.

Task: Write a program. When it starts it should pick a number between 1 and 100 (inclusive). It should then iterate (loop) asking for a guess, and telling whether the guess is too high, too low, or correct. After the correct answer is found, it should start over, picking a new number. This will involve two infinite loops, nested.

Here is an example of what I am looking for, but you can be creative.

```
Let's play High-Low.
I am thinking of a number between 1 and 100.
What is your guess? 50
50 is too high.
What is your guess? 75
75 is too high.
What is your guess? 25
25 is too low.
What is your guess? 37
37 is too low.
What is your guess? 41
41 is exactly right! You win!!
```

```
Let's play again.
I am thinking of a number between 1 and 100.
...
```

3.19 oLT: LocalTime

- Status: Officially Assigned.
- Discussed: Fri, Nov 01
- 6pt Due Date: Mon, Nov 04, 16:00 **
- 5pt Due Date: Mon, Nov 04, 16:00
- 4pt Due Date: Wed, Nov 06, 16:00
- 3pt Due Date: Tue, Dec 10, 23:59
- Grading Label: **oLT**

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 4.1 (page 67) apply.

If you are requesting a regrade, this is the required subject line:
`cis101 regrade oLT lastname, firstname`

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.

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 lines might be helpful.

```
( $s, $m, $h, $d, $mo, $y, ... ) = localtime(time);
$y += 1900; # correct the year
```

** Bonus 6pt Due Date: You can earn 6pt instead of 5pt if you also do the following things by the 6pt due date.

(a) Display the current day of the week as a word (not just a number).

(b) Add at least three submit buttons to your webpage, each specifying a clearly labeled different time zone, such as your home town or someplace famous. When a button is pressed, report the name of the location, and the local time at that location.

http://en.wikipedia.org/wiki/List_of_UTC_time_offsets has useful information about time zone offsets.

3.20 g43: Leap Year

- Status: Officially Assigned.
- Discussed: Wed, Nov 06
- 5pt Due Date: Fri, Nov 08, 16:00
- 4pt Due Date: Mon, Nov 11, 16:00
- 3pt Due Date: Tue, Dec 10, 23:59
- Grading Label: **g43**

This is a GradeBot task. The general rules in section 4.3 (page 68) 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. But if it is a multiple of 100, then it is not. But 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.

You may need to know this: 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. Check for 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)
```

3.21 g75: Tally

- Status: Officially Assigned.
- Discussed: Wed, Nov 13
- 5pt Due Date: Fri, Nov 15, 16:00
- 4pt Due Date: Mon, Nov 18, 16:00
- 3pt Due Date: Tue, Dec 10, 23:59
- Grading Label: **g75**

This is a GradeBot task. The general rules in section 4.3 (page 68) 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)
```

3.22 oHL: High Low Online

- Status: Officially Assigned.
- Discussed: Fri, Nov 15 and Mon, Nov 18
- 12pt Due Date: Mon, Nov 18, 16:00
- 10pt Due Date: Wed, Nov 20, 16:00
- 8pt Due Date: Fri, Nov 22, 16:00
- 6pt Due Date: Tue, Dec 10, 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 4.1 (page 67) apply.

If you are requesting a regrade, this is the required subject line:
`cis101 regrade oHL lastname, firstname`

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

This 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 must request it by email.

Summary: Program the high-low guessing game to run online.

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 submit button, and (e) a hidden field with the number to be guessed. (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 should be between 1 and 100. (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 (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.

Advice: While developing and testing your program, it is convenient to make the secret number into a regular 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 give praise or other commentary based on the skill or luck of the player.

3.23 oB: Boring

- Status: Officially Assigned.
- Discussed: Mon, Nov 25
- 5pt Due Date: Wed, Nov 27, 16:00
- 4pt Due Date: Mon, Dec 02, 16:00
- 3pt Due Date: Tue, Dec 10, 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.

3.24 oF2: Farm 2

- Status: Officially Assigned.
- Discussed: Wed, Nov 27 and Mon, Dec 02
- 12pt Due Date: Mon, Dec 02, 16:00
- 10pt Due Date: Wed, Dec 04, 16:00
- 8pt Due Date: Fri, Dec 06, 16:00
- 6pt Due Date: Tue, Dec 10, 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 4.1 (page 67) 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.

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` and `harvest`. 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. Then I will test it online to see how well it works.

3.24.1 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 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.

3.24.2 Main Program: Call "plant"

Use the following foreach loop to display the names and quantities of the crops.

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

Then include an appropriate "submit button."

3.24.3 Subroutine "Plant"

Do not use any global variables.

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

Display the name of the crop to be planted. (The name of the crop comes from the subroutine's parameter list.)

Display a blank into which a number can be entered. Make the number "sticky." That is, whatever they typed should still be there when the screen is redrawn after they press the submit button. (The quantity comes from a call to `olin`.)

3.24.4 Main Program: Harvesting Call

Use the following foreach loop to display the harvest.

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

3.24.5 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, convert it to a smaller number. Example: if quantity is more than 9, just use 9. At your discretion, complain about the size of the number.

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

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

3.25 oJS: JavaScript

- Status: Officially Assigned.
- Discussed: Wed, Dec 04
- 5pt Due Date: Fri, Dec 06, 16:00
- 4pt Due Date: Mon, Dec 09, 16:00
- 3pt Due Date: Tue, Dec 10, 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 4.1 (page 67) 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.

My calculator has buttons for add, subtract, multiply, and divide.

Fix and customize my example. Then go beyond the example.

The divide button fails when you divide by zero. Fix it so it displays a special error message, not just “Infinity” like JavaScript would normally say.

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 4

Rules For Activities

Contents

My intention is that we will do in-class activities many times through the semester. 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.

Through the semester, this chapter will be modified as new activities are assigned. This chapter gives you a reliable place where you can find information about each task.

Each activity will be discussed in class, and will have one or more due dates and a grading label.

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, Dec 10, 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.

4.1 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 4.4 (page 69) in the construction of your subject line.

4.2 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 4.4 (page 69) 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.

4.3 gXX: GradeBot Task General Rules

GradeBot itself is explained in chapter 5 (page 73).

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 the labels 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 4.4 (page 69) 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 last-name 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.

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.

4.4 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.

4.4.1 To: Line

You can email to `doncolton2@gmail.com`.

You can 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.

4.4.2 Subject Line

The subject line of the email must be as follows:

```
cis101 gradinglabel 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 `gradinglabel` part is replaced by the grading label for that assignment.

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

The `firstname` part is 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
```

4.4.3 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, comment lines start with `#`. So, following the previous example from above, I would have this comment line as the first line of my program.

```
# cis101 p1 Colton, Don
```

Optionally, you can include this line, or one that is substantially the same, before your "subject line" comment:

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

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.

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.

You should avoid having anything else in your email that might make it difficult for me to decide what you intend as your program. I want to be able to assume that your whole email is the program you are submitting.

Specifically avoid including any “reply” comments, but if they are obviously not part of the program, they will be okay.

Specifically avoid a “signature”, but if it are obviously not part of the program, it will be okay.

Chapter 5

Using GradeBot

Contents

`http://gradebot.tk/` is the web interface for what used to be a huge system called GradeBot. What you see at that URL is called GradeBot Lite.

`http://gbot.dc.is2.byuh.edu/` is an alternate URL in case the tk URL stops working.

GradeBot is an automated program tester.

5.1 Testing Your Program

You write your program and upload it or paste it or key it into GradeBot. Then you press a button to have it graded. GradeBot will tell you if your program works properly or not.

If your program works properly, you will see this message:

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

If your program does not work properly, you will see this message:

```
Please fix your program and submit it again.
```

5.2 How GradeBot Tests

The way GradeBot works is that it has a version of each program you are asked to write. It generates some sample inputs, runs its own version, and collects the outputs. That is used to make a script. Your program is expected to behave exactly according to the script.

This makes some serious demands 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.

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.

5.3 Submitting Your Program

Once GradeBot says your program behaves correctly, you can submit it to me, the instructor.

GradeBot does not inform me about the success or failure of your program, nor how many attempts you made. It only reports to you.

Currently I require you to send your program as an email message.

Follow the email rules in section 4.4 (page 69) as you construct your subject line and the body of your message.

GradeBot does not care about style or comments. I tend to care about both. So, when GradeBot says your program is okay, it means it runs okay. It may still need some improvements.

Interpreting GradeBot's Requirements

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 ends with a newline character unless specified otherwise.

Each line ends with a newline character unless specified otherwise.

I said that twice because it is one of the most common mistakes students make.

Numbered lines are shown to designate output that your program must create.

`"Hello"`

If GradeBot expects the line `"Hello"` then you must print the line `"Hello\n"`. You must add the `\n` to indicate that the line is complete.

`"Hello" (no \n)`

If GradeBot expects the line `"Hello" (no \n)` then you must print the line `"Hello"` without any `\n`.

`""`

This means that GradeBot expects a blank line. You must print `"\n"` to make that happen.

`in> .......`

`"in>"` is shown to designate input that your program will be given (through the standard input channel).

`eof  (end of output)`

`eof` means end of file, and indicates that your program must terminate cleanly.

Chapter 6

Programming Style

These rules apply to programs that I need to read, but even for programs that I will not read, you should use good style.

Contents

As your programs become more complex, style becomes important.

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.

6.1 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.

Example: `( 3 + 5 )` is good.

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.

6.2 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`

6.3 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.

6.4 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.

6.5 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.

6.6 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.

6.7 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 7

Exam Questions

Contents

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

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

In the case of GradeBot, only behavior and results are measured. GradeBot does not examine your code to see how you achieved your result. In the case of the exam itself, the human grader does review your code to make sure you achieved your result in the required manner.

7.1 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.

7.2 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.

7.3 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.

7.4 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.

7.5 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.

7.6 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.

7.7 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.

7.8 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.

7.9 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.

7.10 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.

7.11 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.

7.12 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.

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 end of a list.

Example: `$x = shift @x;`

(b4) using `unshift` to add something to the front end of a list.

Example: `unshift @x, "hello";`

7.13 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 $_ }`

7.14 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.

7.15 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] }`

7.16 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)

7.17 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)

7.18 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.

7.19 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.

7.20 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.

7.21 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 $abc;` - creates a local variable named `$abc`.

Example: `my ( $abc );` - creates a local variable named `$abc`.

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`.

7.22 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 8

Final Projects

- Status: Officially Assigned.
- Discussed:
- Deadline: Tue, Dec 10, 23:59

8.1 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/cis101.2135/` is the place to link your project. It is the Student Projects page for this class. Link it to the "proj" slot.

8.2 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 `<img>` 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.

Index