Python as a Teaching Tool

Honors Thesis
Jason Schultz
Advisor: Marty Wolf
April 2009

Since the very beginning Computer Science educators have been divided on how to instruct students. Views have varied on how best to turn conceptional theories into practical application. This has led them to constantly search for more effective teaching tools that aid in accomplishing this end for the student on a consistent and reliable basis. In the arena of commercial programming the emphasis is on productivity in specialized departments using a broad range of languages that meet widely varying and changing needs. There has not been a language capable of meeting the needs of all programmers all of the time. This forces colleges to seek new and more effective tools that achieve these goals. In an ongoing effort to accomplish this, my project was undertaken as a small contribution to the debate. I am endeavoring to ascertain whether Python is an appropriate teaching tool given the parameters just described.

According to Rice and Rosen, Purdue University created the first Department of Computer Science (CS) in 1962. This makes the discipline of Computer Science relatively new. Therefore the theory of teaching (pedagogy) computer science continues to evolve. From the very beginning programming languages were used to reinforce the theoretical concepts taught. At first the choices were limited to the high level languages that were available, these included FORTRAN, COBOL, and BASIC. These first languages were not particularly attractive because they we not easily readability. This all changed in the 1970's when Niklaus Wirth developed Pascal. It was designed to teach programming techniques and topics to college students. As such it was ideal for a computer science course. And it quickly became universally accepted as the best teaching tool available. The main advantage of Pascal was that it was easier to read and

program with, thus leading to more focus being placed on the theoretical concepts. In the 1980's the faculty of many schools started to evaluate whether Pascal was an adequate language to use as an introductory language. Meanwhile object-oriented programming and reusability of previously written code were introduced into the curriculum. This coincided with an increasing importance of object oriented programming in the software industry. Pascal did not consistently and easily deal with these issues. Also the industry as a whole was not using Pascal and therefore students were not as employable as those who learned using "real-world" languages. The faculty at many schools valued languages that provided students "with an opportunity to learn to use a tool that will help address problem solving issues, prepare them for further Computer Science study, and offer them a taste of the Computer Science discipline." [Noon 94] It was during this period that the faculty at many schools chose to change the languages used in CS I. Popular language choices included Ada, C, C++, and Scheme among others. Unfortunately many instructors, specifically those using C and C++, learned that these "real-world" languages are more complex then Pascal, thus requiring more time to be spent on teaching the language and less on the actual goal which is the computer problem solving concepts. This brings us to the issue at hand. Is there a teaching tool that can be used in CS I that is both easy to use as well as being used in industry? That was the question asked by the CS faculty at Bemidji State University in the spring of 2003. After deliberating the trade-offs on this issue they decided to use Python as a teaching tool in CS 2321, Computer Science I.

When discussing which language to use in a CS I course the first question that must be

asked is: What are the objectives of an introductory CS course? William Marion defines it well in his report in Dec. 1999, where he lists five goals for a CS I course.

a) Students should be introduced to problem solving in computer science as it relates to development of a software solution to a problem.

b) Students should begin to understand abstraction and its role in managing complexity of software.

c) Students should begin to develop a working knowledge of at least two software design methodologies.

d) Students should become familiar with the concepts of algorithm, both from the point of view of design and analysis.

e) Students should learn to implement their own design solutions via a modern programming language which provides the necessary structures for an "easy" transition from the design phase to the coding phase.

Also in a survey conducted by Suzanne Levy, one instructor states "It doesn't matter much in CS1 [what language is best to teach introductory programming], since students are still mastering the basics found in most languages. In CS1, the emphasis should be on problem solving, algorithm development, and logical thinking; not on the intricacies or advanced features of a particular language."

With these goals in mind, Python is a particularly attractive option. Python is an interpreted language so students can use the interpreter to sample different aspects of the language to gain a better understanding of that aspect. In addition Python is easy to read. There are two features that lead to this. First, Python does not require variables to be declared prior to use. This is similar to algebra which nearly every entering CS I student

has experience with. Python also uses formatting as a control structure (spaces, tab, and carriage return) instead of punctuation. This contrasts with many languages used in CS I courses. Python also provides an easy to use graphics interface. In addition Python is operating system neutral, which is especially attractive at BSU because CS I is taught in various operating systems. Unlike Pascal, Python is used in industry, by both businesses and government agencies such as Google, NASA, Industrial Light and Magic and John Deere & Company. Lastly Python is absolutely free, so that students can easily download and install it on any computer without charge or violating the terms of the copyright.

The intention of this project was to assist in determining whether Python was an appropriate choice as a teaching tool in an introductory Computer Science course. As such I was invited to develop the labs associated with the initial semester in which Python would be used at Bemidji State University in a CS I course.

Labs

After discussing the process of developing labs with the professor that I would be the teacher's assistant for (Marty Wolf) we decided that each lab would be developed in a three step process. First I would create an outline that included a list of objectives as well as ideas for exercises to be accomplished. These objectives were completed before the class started to help maintain focus on the goal of finishing a certain amount of material. The second step was actually developing and writing the labs so that they met the objectives. This was done directly in HTML code to provide a more polished looking

document for students. An observant reader will notice that I used a format for writing these labs. This format was to step-by-step walk the students through an example then give them an exercise that requires an understanding of the example. The third and final step was to actually complete the exercise myself, that way it would be much easier to determine how realistic the exercise was. All of this was completed before the assigned lab time. The format that I will be using to present this work is to discuss each lab separately before moving on to the next one, the reader is advised that this is not necessarily the order in which the tasks where completed.

Lab 1

Objectives

1. To practice logging onto the CS lab network
2. Be familiarized with the interpreter
3. Be familiarized with the editor
4. Learn process for turning in labs
5. Introduction to input/output

Exercises

1. Write a hello program in the interpreter
2. Write a program to find the value of a cube in the editor

Summary

The main purpose of this lab is to get students familiar with the lab environment. This was especially important because most students would not have any experience with Linux prior to the first lab and they needed to be acclimated to it. However it was very important that minimal time be spent on the operating system that way more time could be spent on the concepts of programming. Also it was important to get students writing code in the first lab so they would be learning by doing, which is the purpose of a lab environment. There is a subtle but purposeful teaching tool used here. That is in one of the code examples the code is not formatted correctly (it should be indented and is not). This was done to try and get the students to read and digest the information and not simply to memorize it.

Results

This lab seemed to accomplish most of the goals set forth. In following weeks there was little confusion regarding the operating system, interpreter, and editor. It might be useful to further explain and work with input/output tools in this lab. However it is not a serious issue as the second and third lab projects take up less then the alloted two hours. Thus allowing for questions to be asked during the extra time In fact it may be preferable to leave some essential things out as it gets the students thinking critically and not just memorizing techniques.

Lab 2

Objectives

1. Understand definite loops
2. Understand how to use arithmetic operations

Exercises

1. Write a program that computes change in denominations
2. Do it n times
3. Do it 100 times

Summary

The main purpose of this lab was to get the students familiar with definite loops. Also the lab helps reinforce the difference between modulus and integer division. We had a discussion regarding whether the text's choice to present if statements later was a good one. I initially believed that it was a poor choice, mainly based on my personal experience. However I have changed my mind on this issue, especially in light of Object Oriented Programming (OOP). OOP stresses reusable code. Code is more reusable when it does not contain if statements.

Results

The primary goal for reinforcing loops worked well. However many of the students continued to have problems with some of the arithmetic operations. This might be caused by the foreign notion of these topics. Therefore they take time to completely integrate into a student's thought process.

Lab 3

Objectives

1. Understand the difference between integer and float data types
2. Understand how to use libraries (math)
3. Understand how functions work and how to use them

Exercises

1. Write a function that computes the distance between two points on a plane.
2. Write a program that receives three points and computes and prints the area of the triangle that these points form.

Summary

The main purpose of this lab is to introduce the use of functions and libraries. Also it was important to begin creating test cases. The idea is to get students used to creating test cases as early as possible so that it becomes part of the process instead of a hassle to be completed at the end. I also made a point to create an algorithm before I started to write code to demonstrate the proper way to solve a computing problem. This is done not only to show the an effective way of programming but also to help students understand that most problems can be broken up into smaller easy to solve problems.

Results

Most students grasped satisfactorily the concepts put forth in this lab. The lab does not

cover the sheer power that functions can bring but that should not be unexpected this early in the course. Also many students continued to have occasional problems with differentiating between integers and floats especially in their code. I think this can be mainly attributed to Python's lack of variable definition which leads to students confusing a variable that they thought was a float but in reality it was an integer. The only thing I can think of that would help alleviate that issue is a stricter naming convention. However I don't think that would be prudent because the idea is to keep as much focus as possible on the theoretical concepts.

Lab 4

Objectives

1. Understand how to read from and write to a text file (this was done later)
2. Understand how strings function
3. Understand string operators (concatenation, slicing, indexing)

Exercises

1. Write a program to translate a string in English into a string in Pig-Latin
2. Write a program to translate from a string in Pig-Latin to a string in English

Summary

This was the first road block of the labs. When trying to implement this lab I realized that it could not be done without an if statement. As a result I was forced to change the

exercises to something that could be done with the processes that had been covered previously. After discussing it with the instructor we decided to go with a modified version of Pig-Latin that relied solely on the string's length. We called this half code. There was also a trick in the translator when words were of odd length. This was useful because the translator did not behave like it should at first glance. Therefore we emphasized reinforcement that testing is very important.

Results

Many of the students that found the "hidden" trick in this project (which was most of them) gained a firm understanding of how to parse strings. However those that were not testing may not have found the problem and did not gain as much understanding. While it is normal for student who follow instructions to get more out of their lessons, should the students who are not as diligent be further punished by not getting a chance to learn? Then again this is a natural consequence. This lab was designed for a two hour time period but most of the students completed this lab in a far shorter amount of time. This is a problem at this stage because more could be taught in the time given. Also I feel that the half code design could be improved upon. It was basically a last minute decision because pig-latin would not work. Possible changes might include adding extra characters to help with decoding or changing to something completely different.

Lab 5

This lab was added in between lab four and five because the instructor felt that more time

was needed to cover the material required to complete lab five.

Objectives (added)

1. Understand how to open and close files
2. Understand how to read from and write to a file
3. Better understanding of string operators
4. Use of the % operator for format specification

Exercises

1. Write a program that reads in data from a file and computes some result and returns a value(s) to the screen
2. Write a program that reads from a file, computes some values and writes those values to a file that the program creates.

Summary

I had less time to consider what to include in this lab than many of the other labs because this one was not planned in the first stage of the project. However it turned out well. The main focus here was input/output with files. Also string formatting was used for the first time. Formatting of strings is one of the most difficult concepts that are covered in these labs. String formatting is harder to read and is very different from English.

Results

For the most part the students understood how to work with files. However there seemed

to be a fair amount of confusion regarding formatting. This was not specifically covered in the lab. It would be beneficial to add a section to the lab on formatting strings. This would give further reinforcement of the concepts taught in the classroom before the students were expected to use the technique. Overall this lab could use some reworking. It was too easily completed by many of the students. Adding more specifications especially formatting ones would help students understand the concepts better.

Lab 6

Objectives

1. Understand how graphics objects are used
2. Understand how to use the "from" command

Exercises

1. Experimentation with graphics and drawing them
2. Draw a basketball court in a window

Summary

This was very enjoyable to write and to help students with. Anything that involves graphics seemed to grab students' attention much better. Also with the shapes that can be drawn using the graphics library provided to the students, it is very important to draw shapes in the right order. This is purposefully not covered in the lab as it is a good process for students to think though what to draw and when. One of the examples is how

to draw a semicircle. In fact drawing a semicircle is impossible given the graphics library. To create the illusion of a semicircle the students had to draw another object over half of an existing circle. The main purpose here was to get the students very familiar using the given graphics program. From this point on nearly all output would be required to be sent to a window on the screen.

Results

Generally students understood how everything worked and were comfortable using graphics objects when they completed this lab. Some of them were challenged to complete an added portion of the lab. Most of these students happily completed it. In fact there emerged some competitiveness among them to finish either first or best. This competitiveness is expected in a male dominated discipline. However it alienates many females and for that reason should not be promoted. Computer Science is striving for gender and racial diversity. This is beyond the scope of this paper but it deserved mention.

Lab 7

Objectives

1.Understand how to define a function

2.Understand how to return multiple values from a function

Exercises

1. Write a function that takes the statistics from a batter and then computes and prints the values for OBP, AVG, and SLUG

Summary

The main purpose is to reinforce the function creation and use techniques that had been taught in the classroom. Also the students must have a firm grasp of how to read from and write to a file to complete this exercise, reinforcing previously learned skills. It is said that practice makes perfect. By requiring students to use skills that were stressed in previous labs I am applying that theory; in fact I am accepting it and working from it. There was an idea tossed around concerning a lab that is not present. This idea consisted of writing a program that took a collection of graphs objects and did a mirror image on then revolving around a specific point and a direction. This idea was discarded because of the complexities involved in implementing it.

Results

I found that students understood functions (even functions that returned tuples) fairly readily. This is largely due to the fact that Python makes the creation and use of functions straight forward. By this I mean that Python uses control structures that are much more intuitive than most languages used as teaching tools. In English we use many different control structures. An incomplete list would include a period, a comma, parenthesis, and a simple space. Computer languages use these same control characters. However for readability of code those control structures that provide both control and readability preferred.

Lab 8

Objectives

1.Understand how if else statements work (as well as elif statements)

2.Understand how to use exception handling

3.Understand what a condition is and how to use one (this was added when writing the lab description)

Exercises

1.Write a program that has a menu listing different shapes. The user will decide which object to draw. Then the program should prompt the user for coordinates then validate them for correctness and draw the object.

2.

Summary

The main purpose of this lab is to reinforce the technique regarding if statements and other associated concepts. This topic is presented late in the course, in fact past half way. Many problems that can be solved with an if statement can be solved more efficiently (runtime wise) without them. Therefore the thought is to introduce if statements as late as possible so that students will get used to solving problems without them. There were a few students who had some coding experience and asked about if statements. They were told not to use them. This broadened their problem solving skills.

Results

This lab has an easy task. At this point in the course students had very little problem dealing with if statements. This lab was more about writing functions and using the graphics library. This is appropriate because if statements are not difficult and little time needs to be devoted to them. The extra time that results can be used to continue to reinforce techniques already covered. One of the techniques used to promote modularity was to instruct students that any function longer then twenty lines should be split up into two or more functions. Although this technique was discussed it was never strictly enforced, it was more of a guideline. The more advanced students were identified informally and challenged to write shorter code. Many times what would happen was they would complete the assigned task and then check in to get their challenge. As the students modified their code they would look for redundant code and create a function to eliminate it. This led to more reusable code and therefore helped them create programs faster. This gave them more time to create more reusable code. By experience these students learned that reusable code is faster and easier in the long run. They are better programmers for it, and it would be interesting to see how they perform in higher level CS courses.

Lab 9

Objectives

1.Understand how conditional loops work

2.Better understanding of the graphics library

Exercises

1.Create some kind of animation like a bouncing ball. (The specifics of the lab were not finalized until the lab was written)

Summary

Conditional loops are the main technique that is covered in this lab. The lab also includes a project that is intentionally high (relatively) on the "coolness" factor. The finished lab is visually appealing, because if a project looks good when it is completed there is a tendency to both try harder and enjoy doing the work more. And if students enjoy what they are doing they learn much better.

Results

This lab frustrated some students very much; they seemed to mostly be hung up on how to "make" the circle turn around. This is exactly the situation the Python excels at. Because there is not compile time it is relatively easy to rewrite some code and rerun. Some may say that this makes the programmers lazy because they do not need as much planning if it is so easy to make changes. In some sense that is true but let us review the life cycle of a program. What is the most costly phase of a successful program development? The answer is maintenance. There is no such thing as perfect code; it is all subject to be changed. Therefore students that are used to making changes are better equipped to address the most costly phase of program development.

Lab 10

Objectives

1.Understand how random number generators work

2.Understand how to use random number generators

Exercises

1.Write a program that graphically represents the growth of a tree's rings over time. Be sure each ring's width is variable instead of fixed.

Summary

The lab is less about random numbers and more about continuing to reinforce techniques already covered. Especially since random number generators are simple to employ in Python and look much the same as functions previously covered. Also as students become more familiar with the graphical interface more can be expected of them in the form of formatting and error checking. In the second semester more formatting and graphical consistency were added as requirements for many of the exercises. I believe that this lead to more pride in their work as well as more creativity. That is to say nothing of more stable programs.

Results

This lab was very interesting as it required me to get information regarding growth patterns of area trees. This brings in a secondary item, and that is that in the course of

learning computer science, students learn about other topics as well. It is very important for a computer scientist to have a wide knowledge base. However this is not the place to discuss that issue. Also there were some areas that were not covered explicitly like the precise method to use when starting a tree's "growth". This is normal in a real world situation (clients not knowing what they want). The students learned the primary concepts put forth in this lab very well. Most of the challenges in this lab are centered around graphics and formatting. However this is to be expected as students grow and learn these concepts more deeply.

Lab 11

Objectives

1. Understand how to create a class.
2. Understand how variables work inside the class.
3. Understand how to write constructor, set and get methods

Exercises

1. Write a Timer class; including setHours, setMinutes, getHours, and getMinutes

Summary

Although the students have been working with classes creating them is new to the students. This lab primarily focuses on the creation of a new class. There is also a lot of reinforcement of previous labs, including formatting, conditionals, and error checking. A

potential addition would be to add another function to the class that adds one Timer to another Timer. This would be fairly difficult if the integrity of the data is to stay. Also in Python there is a built in testing function. This function does not affect run-time. This lab also covered how to create and use this testing function. This facilitates testing without needing to write a separate testing file.

Results

The code to create a class is very similar to the code to create a function therefore the transition is particularly easy to make for most students. This lab seems fairly straight-forward and is one of the shortest in terms of instructions but it has a lot of intricacies involved in the implementation. This is especially true in the timeUntil function (refer to Appendix I).

Lab 12

Objectives

1. Learn how to use lists
2. Practice using indexing with lists

Exercises

1. Write a program that maintains a list of customers for a "business".
2. Write appropriate functions for getting a customer's amount owed, making a payment, adding to amount owed, getting total amount owed by all customers.

Summary

This is the first introduction of a fully functional data structure. As the main focus of this lab, lists are especially functional in Python. Lists support indexing, selective insertion, selective deletion, and even sorting. Therefore lists are the main data structure supported by Python and the only one covered in these labs. There is clumsiness to this lab; the functions the students are required to write are not intuitive (Example: addToBalanceDue). It may be possible to rework this lab and add or change some requirements so that is makes more sense to the students. However a complete rewrite might be in order instead.

Results

The first semester this was done I was not happy with my implementation and felt that a complete rewrite was in order. However the instructor for the second semester took what I had, edited it and created a very workable lab. This seems to show my lack of experience when it comes to writing labs like this one. When it comes to what the students learned, it was apparent that this lab worked as it was intended in both semesters. Please note that lists do function much the same as strings and therefore many of the concepts are intuitive for these students.

Lab 13

Objectives

1. Gain a better understanding of a two dimensional list.

Exercises

1. Write a connect game defined in the lab.
2. Write a function to test if a new piece creates a winning combination.

Summary

This is the last lab in the series. In many ways it reinforces much of what has been covered before. This lab makes heavy use of the graphics package. Other topics that are revisited are arithmetic with integers, lists, functions, if statements, loops, and conditional statements. In some sense this is like a capstone project, however it is not designed to be one. This lab as with all the previous labs seeks to build on the experience gained from previous labs.

Results

I could not have been happier with the effort that one of the students put into this project. The student continued to make improvements above and beyond requirements until the student had exhausted any avenue that they could think of short of introducing a computer opponent. How often do university students, who have increasingly less time near the end of the semester, volunteer to make a project better than they need too? This is a wonderful example of why Python makes a good CS I teaching tool. I cannot verify nor do I intend to, that students learn more when they are enjoying what they are doing. This is a good example of a student who undoubtedly is enjoying the work. As far as the

rest of the students, many of them struggled with two dimensional lists but this is not an easy concept to grasp. Most of them reached an acceptable level of understanding. In retrospect this lab was one of the more "successful" ones in so much as students were able to learn at the same time that they were enjoying the process (although to varying degrees).

Lab experience

During the course of writing these labs I learned a lot in the way of how difficult it truly is to write these labs. Generally I took longer to write the lab then it took the students to complete the assignment in them. Some of this is due to the fact that each lab was written in HTML code, which I was not familiar with at the beginning. If the labs are closely examined and critiqued it will be evident that my skill at writing the labs increased as the labs progressed. I noticed this and if given the chance would have rewritten many of the labs. One of the most striking things that I learned was just how important each word is when written for instruction. There can be no mistaken terms (confusing one language with another) or the student might become confused.

Surveys

I conducted two surveys of the students who participated in the initial CS I course that employed Python as a teaching tool. The first was completed at the beginning of the second lab. The second was done at the beginning of the final lab. (The questions and

detailed analysis are included in Appendix II.) The surveys were conducted using the standard Likert scale ranking system with a 0 value being an opt-out response. When writing the surveys many of the questions had a close correlation. This was intentional, as I attempted to monitor any changes in the responses. There were eighteen participants that took the first survey and over the course of the semester that number diminished to twelve when I administered the summary survey.

The surveys revealed that the students had very little previous programming experience. There were a few students who indicated experience with special purpose languages and one who had expericence with Visual Basic. Ironically, students reported a high level of programming ability, even though they had not inidicated much programming experience. Also it was clear that the respondants had no frame ofreference from which to answer the questions. This is especially true for the first survey. Therefore the results are likely not very useful. When applying statistics to the results from the first survey compared to the results from the second survey there is only one question that jumps out. That is whether the student would likely take CS II. There is no question that at least one student decided not to continue on to CS II. This is not unusual; in fact, it is expected that there will be a certain amount of attrition.

Interviews

In an effort to determine if students were able to make a fairly easy transition from Python to C++ as they moved into CS II, I asked for volunteers, who had taken the initial

course and who were taking CS II in the next semester, to be interviewed. The interview was administered near the end of the semester in which these students took CS II.

I interviewed a total of eight students, each at a different time. Each interview took roughly 30 minutes and each student seemed happy to help with my study.

1. Do you enjoy solving programming problems?

The general consensus was a positive one. The students had a wide variety of favorite topics, but ones that came up several times were graphics and games. This was not a surprise because I was in the labs with these students and gauged their response to different lab assignments.

2. What do you remember about Python?

Each student had good things to say about Python. Most of the students focused on how Python made sense. Some even reflected that it was almost too easy. It was also mentioned that Python was a good language for beginners. Also many said that they enjoyed the graphics component. Also almost half expressed an appreciation that pointers were not necessary.

3a. What do you like about Python?

The overwhelming response was that Python was easy.. Others said that it made more sense or was less confusing. In general they all agreed to the ease of Python.

3b. What do you like about C++?

Most of the students focused on the fact that C++ was more powerful and that you got more control. The students also expressed that C++ was widely used.

4. Did you find Python helpful in learning the concepts in CS I?

Each student felt that Python was helpful. They continued to stress the ease and simplicity of Python. Also the students expressed that OOP is easy to understand when implemented in Python.

5. What has been your learning experience in CS II compared to CS I?

Most students expressed that CS II was harder than CS I. Also several of them thought that C++ was not easy to learn, particularly when it comes to classes.

Conclusion

At the beginning of this project I sought to assist determining whether Python was an appropriate choice as a teaching tool in an introductory Computer Science course. The evaluation comes on multiple levels. Firstly, Python has the functionality to preform all of the tasks laid out in the labs. Secondly, it is easy enough to understand for beginning CS students. And finally, the students liked it both during the labs as well as later when they had advanced into more advanced CS topics. There are some issues with drawing any conclusions for this study. The size of the group is far to small, a more appropriate size would be in the thousands. There are no comparative studies regarding other

languages. The study is far too short, four to five years would be more complete, considering that the goal is to produce the most knowledgeable and therefore most employable students as possible. However it is fair to say that Python is an excellent choice for a teaching tool in an introductory CS course and that is fares well when compared to C++.

Topics for further study

- What effects does the first language learned have on the methods that these students will be comfortable within the future?
- Will there be any drawbacks later in these students careers do to using Python?
- Python seemed to be an improvement to C++ in regards to syntax, with there be repercussions when students are forced to learn a more syntactically complex language?
- How will these students preform in the workforce?
- How important was the inclusion of graphics as early as possible?

Epilogue

Since concluding the research on this project five years have passed(it being April 2009). Much has changed since then. Bemidji State University continues to use Python in it's CS I courses. I have been told that they are happy with the results that have been observed. The simple fact that years later they are still happy with their decision seems to say that it was a good one.

When this course was first offered there was only one textbook published that covered Python as a tool in a CS I course. And that book was less then a year old. Since then many more books have been published in the genre. This would seem to indicate that there is greater demand for Python in CS I courses. Although I have no empirical proof, it would seem that faculties at other institutions are coming to the same conclusion. That is that Python makes an excellent teaching tool in a CS I course.

Bibliography

Reed, David, "Incorporating Problem-solving Patterns in CS1", *SIGCSE Bulletin* Volume 30 Number 1, 1March 1998.

Levy, Suzanne Pawlan, "Computer Language Usage in CS1: Survey Results", *SIGCSE Bulletin* Volume 27 Number 3, September 1995.

Marion, William, "CS1: What Should We Be Teaching"?, *SIGSCE Bulletin* Volume 31 Number 4, December 1999.

Zelle, John, Python: "An Introduction to Computer Science", [Franklin, Beedle & Associates], 2003.

Rice, J.R.; Rosen S., "Computer sciences at Purdue University- 1962 to 2000", A*nnals of the History of Computing*, IEEE Volume 26, Issue 2 April-June 2004.

Noon, John P. (ed.) "Teaching CS1: What is the Best Language", , vol. 3, No. 3, 1994.

Pilgrim, Mark, "Dive Into Python", [Free Software Foundation] 2004.

Appendices

Appendix I - Labs

Lab 1

Welcome to Python Labs. These labs are designed to give you an understanding of how to program. We will be focusing on using Python.

Goals of this lab

1. Be familarized with the interpreter.
2. Be familarized with the editor and know the difference.
3. Introduction to input and output.

Let's get started. First you will need to open a terminal window in linux. Then type the command "python". This will start the interactive interpreter. This is were you will experiment with different lines of code. So let's try it out. Type print "<Your Name>" then hit enter. What does the interpreter do? Prints <Your Name> right? in fact it will print whatever is contained in the double quotes. After thinking about this you might wonder, what good is this? if I already know what I want to see. Well, how about if we can change all or part of the output? Try this.

```
def hello(name):
  print "Hello", name
```

Nothing seems to happen. That is because all we did was provide the interpreter with a definition. We can now use that by typing hello("Jason")

The problem with working in this way is that our program definition cannot be saved and used later. We can solve this by writing directly to the disk and then loading the program. This is called using the editor. You can now close the interpretor by pressing and holding down the control key and then pressing the D key.

How about a practice problem we can solve using the editor. I want to know how much area is in a box. I can measure each side and know the length, width and height. I want the program to tell me the area.

Let's start by designing an algorithm. We need to input length then width then height then we need to compute the area and finally output it in a meaningful way

Go to directory /2321/lab1. Type gvim area.py. Then enter the code that I have designed.

```
def main():
print("This is a program designed to calculate the area of a box")
length = input("What is the length? ")
width = input ("What is the width? ")
height = input ("What is the height? ")
area = length * width * height
print "The area of the box is", area

main()
```

Save this as area.py

At the command prompt type: python area.py

```
Verify that the program runs properly and turn it in. To turn an
assignment in, at a command prompt type.


cp FILENAME ~/cs2321-TurnIn
```

Lab 2

Hello again, obviously I didn't scare you off with the first lesson, so let's move on to more useful information.

Goals of this lab

1. Understand how to use arithmetic operations (integer division and modulus)
2. Understand definite loops

Let's do some experimentation so open the interpretor. First let's look at division.

Try print 3/5

What happened? The answer I expected was 0.6 . Let me explain something about "computer math". Python by default treats numbers as integers (whole numbers). Think back to when you were in grade school and didn't know about decimals yet. How did you do division. Probably with whole numbers and remainders. Basically when you divide 2 integers Python rounds the result down. Therefore if the divisor is smaller than the dividend then you get 0. This procedure requires some testing to be familiar with it. Go ahead and do that now.

Now that you feel confident in using integer division let me tell you how to get a decimal as an answer. Make either the divisor or the dividend or both a number with a decimal.

Examples

- print 3 % 5
- print 5 % 3
- print 3 % 5.0

Keep experimenting until you understand how it works. This can be pretty tough so don't be afraid to ask questions.

Now that you know the arithmetic operations needed for this lab, we will move on to the second part of this lab. And that is definite loops. I will only go over something called counted loops. Other kinds of loops will be explained later. A counted loop is a loop that will be done exactly x number of times. It is written like this.

for <variable> in range (<positive number>)

or more specifically like this:

for i in range (10):

Actually the positive number can be a variable but if it is not a positive number the loop will never get done.

Let's experiment in the interpretor.

```
for i in range (10):
 print i
enter
```

I don't know about you but I am experimented out. So let's close the interpretor and start an editor. I want to write a program that will compute the amount of each denomination I should receive when getting change. Let's work out an algorithm.

Print a statement telling the user what the program does

Ask the user how much money they should receive in change
Compute how many quarters, dimes, nickels and pennies are to be received
Print the result in a meaningful way

Sounds pretty straightforward so let's write the code.

```
def main():
  print "This is a program that computes change into least
denominations"
  change = input("Please enter the number of cents to be given: ")
  quarters = change / 25
  change = change % 25
  dimes = change / 10
  change = change % 10
  nickels = change / 5
  change = change % 5
  pennies = change
  print "You should receive", quarters, "quarters and", dimes, "dimes
and", nickels, "nickels and", pennies, "pennies."
  print "Thank you for using this program"
main()
```

Try running it and see if it works. It should if you write it correctly.

The program ends as soon as it gets a result. How about if we want to get more numbers? We can add another loop. The algorithm might look like this.

```
Print a statement telling the user what the program does
Ask the user how many times they want to enter an amount
Do loop n times
  Ask the user how much money they should receive in change
  Compute how many quarters, dimes, nickels and pennies are to be
received
  Print the result in a meaningful way
Print statement thanking the user for using the program
```

This can be accomplished by putting a loop around part of our program.

A possible implementation would be

```
def main():
  print "This is a program that computes change into least
denominations"
  times = input("How many times would you like to run this program? ")
  for i in range (times):
    change = input("Please enter the number of cents to be given: ")

    < The rest of preceding code >
```

```
  print "Thank you for using this program."
main()
```

If you have any questions you should ask them now.

My challenge to you now is to modify the program so that when run it prints every possible amount of change that can be returned from 0 to 99. When you have completed this turn it in to the turn in folder and then you are free to go until next week.

Turn in command

cp <filename> ~/cs2321-TurnIn

Lab 3

Hello and welcome back to another fun-filled lab.

Goals of this lab

1. Understand the difference between integer and float.
2. Understand how to use libraries.
3. Understand how functions work and how to use them.

Last time I talked a little bit about integers. They are whole numbers and can be stored perfectly accurate. The other type is called a float. They represent a decimal number and depending on several factors they can be incorrect. Not to worry they are more than adequate up to about 10 decimal places. Also integers have a far greater range then floats. Remember that Python will choose integer unless you tell it otherwise.

Next there is the use of libraries. These are programs written by someone else, that you can use in your program. An example is the math library. It contains mini-programs called functions that can be used for more complicated numeric operations (ex. square root, pi, sin()). We will be using one of these.

```
sqrt(<float>)
```

Let me explain how a function works. Python takes the things in the () parenthesis sends

it to the function name and puts something else in place of the whole statement. In other words when I write the command print sqrt(4) Python takes the number 4 and sends it to the sqrt function which gives that answer 2. This answer replaces sqrt(4). Leaving print 2.

Let us examine the problem of finding the distance between 2 points on a plane.

First we need to talk about test cases. Are there any conditions that might make our program not work correctly. How about negative inputs, or zero, or 2 points that are the same.

Test Cases
point1 (-2,3) point2 (7,-3) expected results: 10.81665383
point1 (0,5) point2 (4,0) expected results: 6.403124237
point1 (2,4) point2 (2,4) expected results: 0.0

Algorithm design comes next. Here is a possible solution.

Print a statement telling the user what the program does
Ask the user what the X coordinate of the first point
Store that information
Ask the user what the Y coordinate of the first point
Store that information
Ask the user what the X coordinate of the second point
Store that information
Ask the user what the Y coordinate of the second point
Store that information
Determine distance between the points
Print the result in a meaningful way

There is a formula to find the distance between two points. Here it is.
((x1-x2)2+(y1-y2)2)1/2

Now we can start writing the code:
Go to you 2321 directory, make a directory called lab3, go into it and use gvim to open a file called distance.py

```
import math #makes the math library available

def main():
  print "This program computes the distance between 2 points on a
```

```
plane."
  x1,y1 = input("Enter the coordinate for the first point(X,Y): ")
  x2,y2 = input("Enter the coordinates for the second point(X,Y): ")
  print math.sqrt( (x1 - x2) **2 + (y1 - y2) **2 )
main()
```

Now extend the program to accept a third point and compute the area of a triangle. Let us think about test cases, first. What are the extremes? I have chosen 2 test cases. I expect you to add 2 more and type all 4 at the top of your program using comments.

Test case 1: (-2,3)(7,-3)(1,2) expected result: 4.5
Test case 2: (0,5)(4,0)(0,0) expected result 10.0

The formula to find the area of a triangle is (s (s - a) (s - b) (s - c))1/2
Where s = (a + b + c) / 2
and a,b,c = the lengths of each side

When you have completed everything turn in the assignment.

Turn in command

cp <filename> ~/cs2321-TurnIn

Lab 4

Welcome to the lab and another week of action packed thrills.

Goals of this lab

1. Understand how strings work
2. Understand string operations

Let's start the lab by opening the interpreter.

First let me explain something about strings. Strings are really a collection of letters. Each letter can be accessed using indexing. The command is written like this: <string name>[<integer>]. Remember that the indexing starts at 0 and goes up by one. Let's try it.

```
S = "String indexing testing tool"
```

Can you answer these questions?
What letter is at S[3]?
What letter is at S[7]?
What letter is at S[15]?
What is the index of x?
What is the index of l?
What is the length of S?

There is a command that returns the size of a string. It is called len. The syntax looks like this.

```
len(<string name>)
```

Try it now on S.
What were you answers for the questions?
If you think that you had a problem ask.

Now on to a problem (please, hold you're applause). We want to convert a word into something I call Half Code. The word is split half way and the first half is placed after the second half.

Here are some examples:
blue goes to uebl
sample goes to plesam
Python goes to honPyt

First we need to think about test cases. Here are 2 that I came up with. You should add 2 more.
test goes to stte
student goes to dentstu

Now how about an algorithm. I developed a solution to the problem here.

Print a statement telling the user what the program does
Ask the user for a word to be changes to half code
Store the word
Slice the first half of the word and add it to the second half of the word
Print the word to the screen with an appropriate statement

Now that we have an algorithm we can start writing the code. Go into your 2321 directory and make a lab4 directory. Go into that and create encodeHalf.py in gvim.

```
import string
def main():
  print "This program converts a word into half code"
  englishWord = raw_input("Please enter the word to be changes into
half code: ")
  length = len(englishWord)  #the number of characters in englishWord
  halfCodeWord = englishWord[length/2:] + englishWord[:length/2]
  print halfCodeWord
main()
```

How did your testing go? Did everything work correctly? I hope so. Now as some of you might have guessed I would now like to see you write a translator. Create a file named decodeHalf.py. Write your solution with 4 test cases included in comments at the top of the file. When you are done turn the file decodeHalf.py in.

Don't forget to do testing!!

Turn in command

cp <filename> ~/cs2321-TurnIn

Lab 5

Welcome to another week of lab. One thing to keep in mind is that you have to crawl before you can walk. Enjoy.

Goals of this lab

1. Understand how to open and close files
2. Understand how read from and write to a file
3. Better understanding of string operators
4. Use of the % operator for format specification

For this lab we need to be able to open and close a file. The syntax for each command is:

<file handle>.open(<file name>,<options>)
<file handle>.close()

<file handle> represents a variable that refers to the file
<file name> is the name of the file to be opened or closed
<options> options for opening a file, examples "r", "w", "a"

Now I have a file open but how do I read from it? How about using the readline function?

<file handle>.readline()

It returns all the characters of the file until a newline character is encountered. In other words it reads the next line.

Okay let me explain how to write to a file. There is a function called write it looks like this:

<file handle>.write(<string>)

The write function is very strict on the fact that it will only accept a string as a parameter.

Now let us talk about a problem. I want to open a file that looks like this and read it in. Then I want to know the average score for the group. For the test cases you should create a text file with the same format. Now we can move on to the algorithm.

Print an introduction
Ask the user for the name of the file to be read from
Store this name
Open the file given
Read in the data
Add the scores into a sum as each is read
Keep track of the number of scores entered
Calculate the average
Print out the result along with an appropriate statement

Now we can proceed to writing the code.

```
def main():
  fileName = raw_input("Filename: ")
  fileHandle = open( fileName, "r" )

  fileHandle.readline()  # Throw away the first line

  count = 0
  total = 0

  for line in fileHandle :
    name, score = line.split()
    count += 1
    total += int(score)

  print "The average score is %6.1f" %(float(total)/count)

  fileHandle.close()

main()
```

Recall that the % operator inserts a value into it's place. The value is equivalent to whatever the statement is in parenthesis after the string.

Now your assignment is to write a program that reads a file specified by the user(in the same way as previously done) then computes the percentage and puts it in an individual file for each student. The name of each file should be <student name>.grade . Create a testfile containing at least 4 entries plus a total to be used for testing. Make sure that your program works correctly for all test cases and turn your program and test cases in to the usual spot.

Lab 6

Welcome back for another lab session.

Goals of this lab

1. Understand how to use the from statement to import a module
2. Understand how graphics.py objects are used

For the excersices in this lab me need to use a module called graphics.py. First you will need to make a directory called lab6 in your cs2321 directory. To get the module type

```
cp ~/cs2321-Info/graphics.py ~/cs2321/lab6/graphics.py.
```

Now because we will be using graphics.py alot in this lab we can use the from statement. It looks like this.

```
from graphics import *
```

It is used in place of an import statement and provides direct access to the imported commands without having to type graphics. first.

The graphics module provides many objects that can be used. Here are some of them along with syntax for initialization.

```
GraphWin(<optional string>,
        <optional integer>,
<optional integer>)                    #a window that you can draw other
objects in
Point(<integer>,<integer>)                      #X coordinate and Y
coordinate
Circle(<Point>,<integer>)                      #center and radius
Rectangle(<Point>,<Point>)                      #opposite corners of a
rectangle
```

```
Line(<Point>,<Point>)                          #endpoints of a line
Oval(<Point>,<Point>)                          #points form an invisible
rectangle that the oval fits into
Label(<Point>,<string>)                        #centers string on Point
```

There are several commands that can be executed on each object. I am listing some of them here along with syntax.

```
setFill(<color as a string>)      #e.g.
red,green,blue,black,white,yellow
                                              (not every color will work)
setOutline(<color as a string>)
setWidth(<integer>)
draw(<GraphWin>)
undraw()
move(<integer>,<integer>)          #direction to move in X and Y
```

Now that you have some tools go ahead and open the interpreter to do some experimenting. First off you will need to create a GraphWin, which automatically gets opened. Try each object and command and make sure you know how to use them.

Now that you are familar with some of the Graphs Objects I would like to see you write a program that draws the pattern for a basketball court. The dimentions and color scheme will be provided. When you are done turn it in to the usual directory.

Lab 7

Welcome back for another lab session.

Goals of this lab

1. Understand how to define a function
2. Understand how to return tuples from a function
3. Continued use of string and file operations
4. Continued use of formating

Welcome back to the lab. Today we will be writing functions. Functions are very useful for repetitive commands. If for example I want to print "Hello <name>." for everyone in class. I could write a function that does that for me. It might look like this.

```
def Hello(firstName):
  print "Hello %s." %firstName
```

This can be especially useful. Here is an example of it's use.

```
def main():
  fileHandle = open("students.dat", "r" )
  for line in fileHandle:
    Hello(line)
main()
```

This program reads each line from a file ("students.dat") that contains a student's name and prints "Hello <student's name>.". Some of you may have noticed that the statements Hello(firstName) and Hello(line) are not the same. This is true. Let me explain. First of all def Hello(firstName): is a function definition, hence it starts with def. A function definition cannot execute on it's own, it must be invoked. The act of invoking a function is sometimes refered to as calling it. This is exactly what Hello(line) does. Therefore it is formally a function call.

Also a function can return a tuple. This can be seen here.

```
def swap(element1, element2):
  return element2, element1
```

A call to this function might look like:

```
a,b = swap(a,b)
```

Now let's move forward to the assignment. I want to do some computations on baseball players' ststistics. I want you to write a function that takes as arguments: at bats, singles, doubles, triples, home runs, walks, sacrifice flies and hit by a pitch. And returns as a tuple: batting average, on base percentage, total bases, and slugging average. Write an algorthm for this function. Come up with at least 4 test cases. This function should be written in a file called stats.py that is located in a lab7 directory.

Hits = Singles + Doubles + Triples + Home Runs
Batting Average = Hits / At bats
On base percentages = (Hits + Walks + Hit by Pitch) / (At bats + Walks + Hit by Pitch + Sacrifice Flies)
Total Bases = Singles + 2 * Doubles + 3 * Triples + 4 * Home Runs
Slugging average = Total Bases / At bats

When you have completed and tested your function in the interpreter, you should write a program (named statsdrive.py) to read from a file which has an example here. Then you should compute the data using your function and write the output to a file (named output.dat) and print it to the screen. When you are done turn in all files that you created into the usual place.

Also it would be a good time to clean up your Turn-In directory by removing everything except lab7.

NOTE: If you like this lab you might be snake bitten!
Sorry just a little Python humor.

Lab 8

Congratulations, we are now halfway through these labs.

Goals of this lab

1. Understand how if and elif statements work
2. Understand what a condition is and how to use one

First of let me talk about a condition (also called a Boolean expression). When a condition is evaluated it produces either a true result or a false result. This is extremely useful in writing code. It allows your program to make "decisions" while it is running. For instance if you want to evaluate input from a user before you do any computation on it. An example would be to check if the user has input a letters when you asked for a number.

There are 6 relational operators in Python. Each has a different meaning. They are explained on page 203 of your book. You should familiarize yourself with them now.

An if statement is a very useful tool for programmers. There are some problems that require it's use. It looks like this.

```
if <condition>:
  <statement>
```

The semantics of an if statement are, if the condition evaluates to true execute the statement otherwise skip them. An example of how an if statement would be used is here.

```
name = "Lena"
if name == "Lena":
  print name
```

Sometimes we want to execute some different set of statements when the condition evaluates to false. This can be done with an else statement. An else statement extends an

if statement syntactically it looks like this.

```
if <condition>:
  <statement1>
else
  <statement2>
```

How this works is exactly like an if statement except when condition evalautes to false then execute statement2. Here is an example.

```
name = "Ole"
if name == "Lena":
  print name
else:
  print "Your name is not Lena"
```

Now on to the project. This lab will be using graphics.py so you should make sure to import it. Write a program named objectDraw.py that will draw an object in a window. Your program should start by displaying a window containing the following menu options (1.Line 2.Circle 3.Rectangle) and a entry box to make a selection in. After a selection is made (Hint:Look at the Celsius Converter example from the text) another window is opened to input the appropriate coordinates. The coordinates are checked to make sure that the object will fit in the window. If the object will not fit, your program should open a window displaying an error message. The program then opens the window and draws the appropriate shape. You can find a function that draws a line for you, along with some other helpful functions in this file.

Be sure to test you program. When you are done turn your program in.

Lab 9

Congratulations, we are now halfway through these labs.

Goals of this lab

1. Understand how conditional loops work
2. Better understanding of graphics objects

Hello and welcome to the lab. This week you will be working with indefinite loops. These are loops that are not done a set number of times. They continue until some conditional evaluates to false. This is done using the while loop. The syntax looks like this.

```
while <conditional>:
  <statement>
```

Semantically it tests the condition and if true the statement is executed. Then it starts over. There is a danger of creating something called an infinite loop. That is a loop that will never stop. To avoid this we need to make sure that the conditional is brought to a state closer to evaluating to false. An example is when the condition is x < 42. Each time thought the loop, x must be changed to a value closer to 42.

Here is an example.

```
x = 40
while x < 42:
  print x
  x += 1
```

This will print 40 and then 41. Notice that it does not execute the loop when the conditional fails.

This lab makes use of graphics.py. You should make sure that you are famliar with all of it's functions.

Some of the Shape functions

- setFill
- setOutline
- setWidth
- draw
- undraw
- move
- clone

There is a function called sleep that will supply some help in this lab. It is in the time module. It looks like this.

```
sleep(<number>)
```

What sleep does is pause the system for a number of seconds equal to the number provided. You can supply decimal arguement.

The assignment for this lab is to open a window for input. The user will be asked what size to make the a display window and what size to make the circle. The window is created including 2 "goal posts" located the top of the screen. Each "goal post" is to be located 1/3 and 2/3 the way across the screen. Then the user must supply a mouse click for the initial center of the circle. The circle is drawn in the correct position and filled

with any color. The "ball" should then move across the screen until it gets to any side. At which time the "ball" should change directions unless a position between the "goal posts" reached in which case the program will end. Your implementation should be written in a file called bouncingBall.py. Make sure that your "ball" does not move to fast and that you use functions appropriately.

Be sure to test you program. When you are done turn your program in.

Lab 10

Hello and welcome to the lab.

Goals of this lab

1. Understand how random number generators work and how to use them

In this lab I will be discussing random numbers and random number generators. Random number can be useful especially when writing a program that simulates real life in some way. Almost all games use them. These numbers are generated using one of several algorithms that I will not discuss here.

Well then, there is a module called random that contains functions ready to be used. The first of these is random(). It does not take any arguments and returns a float f such that $0 \leq f < 1$.

Try experimenting with it in the interpreter until your are familiar with its use.

The second function that I want to talk about from the random module is the randrange() function. It takes an arguments 2 integers and returns a random integer in the range specified. Here is an example of how it might be used.

```
from random import randrange
i = randrange(1,6)
j = randrange(1,6)
print "You have rolled", i + j, "on 2 dice."
```

Use the interpreter to use this function until you know understand how it works.

Now on to the assigned task. You should write a program called treeGrowth.py that simulates the ring patterns of different types of local trees. The first thing that your

program should do is open a window with a list of trees. The user should be required to input a selection. An incorrect selection should be handled in a gracious way. Then open a new window and draw rings in the center to repesent the "growth" of the tree, adhering to its growth rate and maximum size. Your program needs to accomodate the three different growth rates outlined in the table that follows. Be sure to test you program. When you are done turn your program in.

Lab 11

Hello and welcome to the lab. This week we will be working with the class structure.

Goals of this lab

1. Understand how to create a class
2. Understand how variables work inside a class
3. Understand how to write constructors, set and get methods

All of you are familiar with several classes. Some of the examples are the Circle, Point, Text and GraphWin. A class consists of 2 different parts, data members(variables) and member functions(methods). These parts individually or collectively can be called attributes.

The methods of a class define what a class can "do". A GraphWin can setCoords or getMouse. Common methods are set methods and get methods. A set method are used to change the value of a data member(s). A get method returns a value of the class . An example is the getY of the Point class. This method returns the value of the Y coordinate of the instance of a point. A class usually has other methods. These might include an area method or a grow method for a tree class.

There is a special kind of method called a constructor. A constructor initializes the internal variables in an instance of the class.

A class's internal variables are sometimes called data members. These are variables that are stored in the class. A car class might have a data member that saves the car's model.

For this lab you should create a timer class. Save it in Timer.py. The timer class should

have the following methods and behaviors. It should also have a self-contained testing program (if __name__ =="__main__). You can maintain the data with any data members that you see fit.

Lab 12

Hello and welcome to the lab. This week we will be working with the list structure.

Goals of this lab

1. Learn how to use lists
2. Practice using indexing with lists

A list is an ordered collection of objects. These objects can be integers, strings, user-defined classes, or even other lists. Lists can be very useful. Recall back to Lab 5 when you computed grades for a class, what if you wanted to save all the information from each student. Without a list that would prove very difficult. But with a list and a simple class to hold the information from each student this can be done much more simply.

To access an element of a list, indexing is used. Recall using indexing on strings. Lists work in exactly the same way. In fact all of the functions that can be used on strings can be used on lists as well.

There is also additional operations that lists support but strings do not. One of these is the assignment operator. Therefore lists can be changed. There are tables containing operators supported by list on page 341 and 343 of your text.

For the assignment you will be maintaining a business' billing records. Save your solution in custCredit.py. I have developed a class to keep track of customers. It can be found in the ~/mjwolf/pub/cs2321/Lab12 directory. It is named Customer.pyc. The following table shows the methods and what they due.

Method Name	Method Behavior
Constructor(0 arguements)	Creates an instance of the Customer class
setCustID(integer)	Changes the customer's ID number to the argument
setName(string)	Changes the customer's name to the argument
setAddress(string)	Changes the customer's

	address to the argument
setCreditLimit(integer)	Changes the customer's credit limit to the argument
setBalanceDue(integer)	Changes the customer's balance due unless it would be greater than credit limit in which case it throws an exception
getCustID()	Returns the customer's ID number
getName()	Returns the customer's name
getAddress()	Returns the customer's address
getCreditLimit()	Returns the customer's credit limit
getBalanceDue()	Returns the customer's balance due
addToBalanceDue(integer)	Adds the argument to the balance due unless it would be greater than the credit limit in which case it throws an exception
__str__	When Customer should be converted into a string python does it automatically

What you need to do is create a main file that creates and displays a window. This window should have 6 menu options. They are to get a customer's credit limit, make a payment, add to a balance, get the total amount owed by all customers, save database, and exit program. Your program should read in all customer's from a file named customers.dat. When asked for a customer's credit a window should be displayed asking for the customer's ID number, and then a window is opened to display the information. To make a payment a window is displayed that asks for a customer's ID number, and then a window is opened to display the information. To add to balance due a window is opened that asks for a customer's ID number and the amount to be added. When the total amount is asked for a window is opened to dislay this amount. When save is selected all the information should be saved to customers.dat. Exit program should do just that.

Be sure to test you program. When you are done turn your program in.

Lab 13

Hello and welcome to the last lab of the semester.

Goals of this lab

1. Gain a better understanding of lists using a 2 dimensional list

We have covered many aspects of writing programs. You should have enough skills to complete this one last lab. But first I want to clarify a point concerning lists. In lab 12 I said that a list could contain another list. This is called a 2 dimensional list. Pictorially you might think of a 2 dimensional list as a grid. Therefore this syntax would be correct.

```
myList1 = [1,1,1]
myList2 = [2,2,2]
myList3 = [3,3,3]
myOtherList = [myList1, myList2, myList3]
```

At this point myOtherList would look like this pictorially.

```
1  1  1
2  2  2
3  3  3
```

For the assignment in this lab you will implement a connect 4 game. This will be done in a game window. If you want an example you should look at Gnect. It is a game installed on the Linux side of the computers in the CS lab. Your program should be called connect.py. This game consists of 2 players alternately dropping discs into columns, gravity pulls them to the bottom. When either player gets 4 discs in a row(vertically, horizontally, or diagonally) then that player wins.

Your implementation need only support human versus human. Please note that the board is only 7 boxes by 6 boxes. The markers can be any color as long as they are visibly unique. After each mouse click a new marker of the appropriate color should appear in the window and your program should check to see if anyone have won.

A thoughtful design of the solution to this problem divides into 3 distinct parts. The first part creates and displays the window with dividing lines. The second gets a mouse click and draws a disc in the lowest empty position in that column. The third determines if the newly added disc creates a winning solution.

In drawing the window I found that squares of size 80 pixels by 80 pixels is about right. Otherwise this part seems very straightforward.

For the second part I first capture a mouse click into a point. Then I move that point to the upper-left of the box that it is in. By maintaining a list that stores the number of discs already in a column, it is straightforward to determine the appropriate location to draw the new disc.

To check for a winning solution I maintain a 2 dimensional list containing the color of each disc or a 0 for an empty box. The list is called grid, color is the current color, posX is the row index of the current disc, posY is the column index of the current disc. Your coordinates might be different but the general principal is the same. Here is my implementation for checking the down-left and up-right diagonal. It works much the the extendStreak function did for hot streaks and cold streaks, except that it checks for the same color of disk in both directions. There are 4 functions of this type, one to cover each of the 4 directions.

```
def checkDownLeft(grid, color, posX, posY):
  X, Y = posX, posY
  while(X >= 0 and Y >= 0 and grid[Y][X] == color):
    X -= 1
    Y -= 1
  endX, endY = X + 1, Y + 1
  X, Y = posX, posY
  while(X <= 6 and Y <= 5 and grid[Y][X] == color):
    X += 1
    Y += 1
  if(endX + 3 > X):
    return True
  return False
```

Be sure to test you program. When you are done turn your program in.

Appendix II - Surveys

Question 1

Pre-question: What programming languages have you used in the past?

Rationale: This is an attempt to capture the level of programming experience the participants possessed.

Pre-average: 1.39

Pre-median: 1

Analysis: These number may very well be skewed higher then might be expected because all responses were considered valid including two instances of HTML, four instances of Karel++, one of Labview, one of Mathematica, one of batch, and one respondent who listed both VB6 and VB.net. If these less then fully functional languages were thrown out the average would be below one and the median would fall to zero. Even without employing that tactic eight of the eighteen responses are blank (meaning no previous experience). In general I would say this was a fairly inexperienced group although there were one to three exceptions. Also after comparing the quantity answered on this question and the values collected from question #3, I was surprised to learn that several respondents listed no languages yet felt their programming skill was very high.

Question 2

Question: Rate your problem solving skills.

Rationale: Attempt to capture student's perceptions of their own problem solving skills.

Anticipated results: I expected to see the result here increase as the less confident students removed themselves from the class and the students who stayed became more confident in their skills.

Pre-average: 3.72 Pre-median: 4

Post-average: 3.50 Post-median: 3.5

T-Test Value: .45

Analysis: The average and median seem to indicate that students were less confident in their problem solving skills at the end of the class then they were at the beginning. This might be due to the fact that they are more educated and understand the question better at the end of the course. However the t-test value tells us that there is no statistical difference in the values. The results are especially striking when it is considered that the less confident students are the more likely to have dropped out.

Question 3

Question: Rate your programming ability.

Rationale: In conjunction with question 1, this would give me a better picture of the level of programming experience in the students. I am also interested in the difference from start to finish of the coarse.

Anticipated results: Much the same as the previous question, I expected this to improve.

Pre-average: 2.81 Pre-median: 3

Post-average: 3.67 Post-median: 3.5

T-Test Value: .084

Analysis: Although there is no statistical difference in the results, it does appear at first glance that the students improved over the course of the semester. This is due to the students becoming better programmers while their perception of a good programmer changed.

Question 4

Question: Rate the likeliness you will take CS II.

Rationale: The hope of this project is to maximize the number of students who will continue on to CS II therefore the results here are vitally important.

Anticipated results: It is expected that the results between the surveys will remain consistent.

Pre-average: 4.83 Pre-median: 5

Post-average: 4.41 Post-median: 5

T-Test Value: 9 E -5

Analysis: The result is very surprising at first glance. All the students answered either four or five on the preliminary survey however there was both a three and a one on the summary survey. Also the t-test value shows that there is a statistical difference in the values. After a bit of reflection it was realized that one of the students for whatever reason would not be taking CS II. The reason this student changed their mind is unknown and it would be impossible at this point to obtain that information.

Question 5-15

Rationale: I am looking for differences between the two surveys as well as between the questions on the individual surveys.

Anticipated results: An increase in rating was expected.

Analysis: There is no statistical difference in any of the results and frankly the results are flawed after thinking about them. The students cannot completely understand the questions and have little or no outside experience to draw on to answer in a meaningful way. It is interesting that all but one of the results had an increase in average on the summary survey.

Question 5

Pre-question: Anticipates positive effect Python will have on your enthusiasm to continue taking Computer Science courses.

Pre-average: 3.88

Pre-median: 4

Post-question: Positive effect Python had on your enthusiasm to continue taking Computer Science courses.

Post-average: 3.73

Post-median: 4

T-Test Value: .3

Question 6

Pre-question: Anticipated usefulness of Python to understand arithmetic operations.

Pre-average: 3.31

Pre-median: 3

Post-question: Usefulness of Python to understand arithmetic operations.

Post-average: 3.92

Post-median: 4

T-Test Value: .61

Question 7

Pre-question: Anticipated usefulness of Python to understand strings.

Pre-average: 3.67

Pre-median: 3

Post-question: Usefulness of Python to understand strings.

Post-average: 4.08

Post-median: 4

T-Test Value: .63

Question 8

Pre-question: Anticipated usefulness of Python to understand Boolean expressions.

Pre-average: 3.31

Pre-median: 3

Post-question: Usefulness of Python to understand Boolean expressions.

Post-average: 3.91

Post-median: 4

T-Test Value: .58

Question 9

Pre-question: Anticipated usefulness of Python to understand conditional statements.

Pre-average: 3.56

Pre-median: 3.5

Post-question: Usefulness of Python to understand conditional statements.

Post-average: 4

Post-median: 4

T-Test Value: .85

Question 10

Pre-question: Anticipated usefulness of Python to understand functions.

Pre-average: 3.63

Pre-median: 3.5

Post-question: Usefulness of Python to understand functions.

Post-average: 4.08

Post-median: 4

T-Test Value: .74

Question 11

Pre-question: Anticipated usefulness of Python to understand loops.

Pre-average: 3.65

Pre-median: 4

Post-question: Usefulness of Python to understand loops.

Post-average: 4.08

Post-median: 4.5

T-Test Value: .65

Question 12

Pre-question: Anticipated usefulness of Python to understand classes.

Pre-average: 3.44

Pre-median: 3

Post-question: Usefulness of Python to understand classes.

Post-average: 4

Post-median: 4

T-Test Value: .86

Question 13

Pre-question: Anticipated usefulness of Python to understand object-oriented design techniques.

Pre-average: 3.67

Pre-median: 4

Post-question: Usefulness of Python to understand object-oriented design techniques.

Post-average: 3.92

Post-median: 4

T-Test Value: .64

Question 14

Pre-question: Anticipated usefulness of Python to understand lists.

Pre-average: 3.6

Pre-median: 4

Post-question: Usefulness of Python to understand lists.

Post-average: 4

Post-median: 4

T-Test Value: .79

Question 15

Pre-question: Anticipated usefulness of Python to understand recursion.

Pre-average: 3.5

Pre-median: 3

Post-question: Usefulness of Python to understand recursion.

Post-average: 3.58

Post-median: 4

T-Test Value: 1

Question 16

Post-question: In your opinion how prepared are you for CS II.

Rationale: One of the goals of this course is to prepare students for further instruction in CS courses.

Anticipated results: It was expected that these results would mirror those from questions two and three of the summary survey.

Post-average: 3.55

Post-median: 3

Analysis: Although the average and median is lower then both question two and three, the individual answers differ by at most one value for all but one student. Also a t-test done on these values shows no statistical difference in the results. Therefore we can safely assume that there is no difference.

Appendix III – Interviews

I am listing quotes from the students that I interviewed. The quotes that are listed are a representative sample of what was discussed.

1. Do you enjoy solving programming problems?

"I like accomplishing something."

"Oh hell yeah."

"CS focuses on the process, I want the answer."

2. What do you remember about Python?

"You don't have to deal with pointers."

"It's free, that is the best thing about it."

"It didn't seem like too complicated of a language."

"Simplicity of it."

3a. What do you like about Python?

"Basic concept is outlined rather well."

"How easy it is to code."

"It's easy."

"More English based."

"The syntax made sense."

3b. What do you like about C++?

"Variety."

"++ operator."

"I really don't like the language."

"Error messages aren't too bad."

"More power in C++."

"Once you know C++ you can more easily jump into anything else."

"You get better understanding of processes."

4. Did you find Python helpful in learning the concepts in CS I?

"Yes, very much so."

"It's perfect for CS I because you don't get caught up in syntax."

"Yes, because it was more English based it is easier to code."

5. What has been your learning experience in CS II compared to CS I?

"Learning C++ was difficult but knowing Python did help."

"There was more concepts in CS I and more actual code in CS II."

"Concepts are harder and language is lower so you have to make up the difference and I don't enjoy that."

"CS I was more of a struggle because I had never programmer before. CS II is easier because I am more experienced now."

"CS I was at a slower pace compared to CS II at first but by the end they are about the same speed."

"CS II is more general with more abstract stuff. I knew something about C++ so learning it was easy."

"CS I helped build a goos foundation of problem solving skills that I could use in CS II. Classes are still more difficult in C++."

"I grasped the concepts better in CS I then in CS II partly because Python is easier to pick up."