Week Six Reading Notes

A lot of material is covered in Week 6. We assume that if you made it through the exam, you will have the skills as well as the confidence to learn at a somewhat faster rate. There is so much we hope to teach!

Video: “Exceptions”

Although Lecture 10 is about Classes, it begins with the important concepts of "Exceptions". Before diving into the Python use of exceptions, let's take a step back. Early on in the class, it was noted that an algorithm or a computer program is like a recipe -- a sequence of steps for the computer to follow. Here is a recipe for chocolate chip cookies:

- Step 1: Preheat oven to 350 degrees F (175 degrees C).
- Step 2.1: Cream together the butter, white sugar, and brown sugar until smooth.
- Step 2.2: Beat in the eggs one at a time, and then stir in the vanilla.
- Step 2.3: Dissolve baking soda in hot water.
- Step 2.4: Add to batter along with salt.
- Step 2.5: Stir in flour, chocolate chips, and walnuts.
- Step 2.6: Drop dough by large spoonfuls onto ungreased pans.
- Step 3: Bake for about 10 minutes in the preheated oven, or until edges are nicely browned.

That sounds delicious, but what if something goes wrong? What if I drop the whole egg into the bowl in Step 2.2 (including the shell)? I probably should take out the shells before beating the egg! Or what if I don't have any walnuts? Well, that probably does not matter that much. But if I do not have any flour, I might as well give up.

These are **exceptions**. If you were writing the recipe for someone who was very literal and you needed to give precise directions, where would you tell that person about the exceptions? If the directions about the exceptions were after the recipe, then the person might just charge ahead beating the egg with its shell, and only after the cookies were done baking, would that person see the directions about the exceptions. On the other hand, it would be very hard to read the recipe if every line had a long explanation as to what could go wrong and what to do. If we did that, Step 2.2 might read as follows:

- Step 2.2: Beat in the eggs, but if the shell falls in then scoop it out before beating. Beat the eggs one at a time, but if two at a time fall in do not worry about it. Then stir in the vanilla, but if you do not have vanilla you can use almond extract or rum, but don't drink the rum.

Python leaves this choice to the programmer by providing a "Try / Exception" mechanism, as explained in the first video in Lecture 10. Python tries to do everything in the Try block, but if something goes wrong, then execution jumps to the exception block.

There is a famous mind-numbing saying in English: "The exception proves the rule". This is very misused, often in silly ways. For example,

- "I always do all the finger exercises and problem sets" says a proud 6.00x student

- "But you did not do Problem Set 4!" says the student's virtual roommate

- "See, the exception proves the rule!" replies the student with a smug expression on his face, that fortunately, none of us can see.

The real meaning, according to Wikipedia[1], is that the exception confirms the rule when there is not an exception. You are to first read the discussion forums when you get stuck, except during the exam period. The exception is during the exam period. When it is not during the exam period, the rule holds. The Wikipedia page may explain this better. My point is that Python exceptions are like this. The try block is fully executed except when there is an exception.

You will hear the technical term "**raise an exception**". This means, "to make an exception happen". If you drop two whole eggs, shell and all, into the batter, you are raising an exception. In Python, the programmer can raise an exception using the Python keyword "raise" (very clever). Exceptions can be raised by the Python system, such as when dividing a number by zero[2] or by the operating system, such as when the program tries to open a non-existent file.

Finally, I want to remind you to pay attention to the third part of the `try/except` construct. The `finally` block is *always* executed, either when the `try` block is finished or the `except` block is finished.

Video: Object Oriented Programming Overview

The second video of lecture 10 is an introduction to objects, as used in object oriented programming. The words in this video may confuse you, but once you get used to talking about objects and declaring them in Python, the confusion will disappear.

A "`class`" in Python is a way of specifying an **object**. You should not be too surprised by this, after all, in Python, a procedure is specified by using "`def`". By now, that should no longer seem strange, even though it was kind of hard to wrap your mind around at first. Sometimes you may observe that the word "class" and "object" are used for one another; they are the same thing.

See if you can catch Prof. Terman using the phrase "willy nilly." According to the web site http://www.phrases.org.uk/meanings/willy-nilly.html, the word "nill" is the opposite of the word "will". Willy means "I am willing" and Nilly means "I am unwilling". In other words, I don't care what Willy Nilly means!

All kidding aside, object oriented programming is one of the most important innovations of computer programming in the past few decades. It may take some time, but I am sure it will influence how people think about many different fields. Python is very "clean" in that it treats everything as an object. In fact, you've already seen objects before! Strings, lists, tuples, and dictionaries are all Python objects!

Notes For The Remainder of Lecture Sequence 10

---

[1] http://en.wikipedia.org/wiki/Exception_that_proves_the_rule

[2] Try entering the following Python code into your interpreter, and see what happens:

```
>>> 1/0
```

Programming is all about applying operations to data. This is familiar to us. In elementary school, we learn about arithmetic operations, such as addition and subtraction. Take two numbers and apply the addition operation. When sitting in a high chair, children learn about applying gravity to food -- the operation is dropping and or throwing and it is applied to whatever can be held in ones hand.

Early programming languages followed this pattern. There are binary operations, such as plus and multiply, and unary operations, such as minus[3] and the logical operation of NOT. Some of these operations can be applied to lots of different types of things. Addition makes sense not just for floating point numbers, but also for strings, characters, and even lists! Then there are a whole bunch of non-arithmetic operations that are needed, such as opening and reading a file, calling a procedure with a bunch of parameters, or appending an item to a list.

After many years of programming and many different programming languages, a new, more uniform way of viewing operations began to emerge. It came from the question: “given an **object**, what are the operations that can act on it?” Rather than taking two numbers and adding them together, we take one number and add another number to it. In other words, the **object** is a number. The number has a bunch of operations that can act on it, one of which is addition. Let's not say the addition operation works on integers, floats, strings, lists and maybe other things we forgot about. Instead, we say there are different addition operations for each type of object, such as for Integer objects or String objects.

There are some operations that behave similarly, such as addition, but there are others that are more tailored to the object type, such as *append* for lists or *read* for files. Each of the object types that come with a language has their own set of operations. Types that come with the programming language are called **primitive types**. Primitive types in Python include strings, integers and lists. Python is very accommodating and allows the programmer to define new object types and their associated operations.

As you continue following the 6.00x course, we hope that you learn new things in each class. In Python, the word "`class`" is the way of defining new types. Just think of a class as a way of learning something new, what that new thing is, how it works, its structure, and what you can do with it. Keep this in mind, as you follow the class in which you will learn about classes.

Another thing to keep in mind is that in many schools, each class is held during its own time period. A period (“.”) is also an important concept in Python classes. Given an object, one can specify its operations (or *class methods*) by using a period. After listening to the lecture, periods will take on a whole new meaning.

I used to do a lot of bicycle riding and remember climbing mountains. When I would see a tall hill, I would get nervous that I would not be able to make it to the top. I would remind myself that hills always look steeper and taller from the base. I need to be halfway up the hill to judge it properly. That helps me make it to the top, since it does not look so bad from the middle and looking back down, I see how much I have done already. As I reach the summit, I realize that it was not a summit, but just a local high spot, after which the road flattens out of a bit and then goes up even higher and steeper. Learning is very much like this. For many, this point in the course is just like my experience in attaining the first "summit” – you made to this point and you have become a better climber, so now it is time to start climbing steeper. Keep with it!

Professor Terman's lectures are different both because he is a different person than Professor Grimson, and because the material is different. It is simply more advanced than the material of the first five weeks. You may have to listen to the lectures more than once, but it is certainly

---

[3] Also called “negative”, as in `-4`

worthwhile. These lectures may focus on Python, but the ideas of object oriented programming are found in many modern programming languages[4].

Lecture Sequence 11

The first video segment of Lecture 11 is a review of Lecture 10. Do not continue until it is understood, and when you can follow it, feel proud of yourself for making the leap! This is challenging material and you may not get it the first few times around. That is okay, but do keep trying!

Professor Terman's lectures are filled with many gems of advice and reasons why certain choices are made. It may be most helpful to watch them at least twice, first to get the big picture ideas and then a second, closer watching to understand the bits of advice and explanations.

There are some example applications that are designed, coded, and used to highlight many important principles of object-oriented programming. As the lecture proceeds, the example gets more involved. It is important to not go on too soon. Also, realize that someday soon, you too will be working out designs that share many features with the examples.

You will be asked to write a class from scratch. It may just be a folk theorem but there is some belief that most people who program only modify existing classes, so after this assignment, you will already be more advanced than half the programmers in the world!

Good luck! Remember to try your hardest and you will persevere.

- Larry Rudolph

[4] An object-oriented language you may have heard of is Java. In fact, in Java one cannot simply write a function; you must understand the concept of object-oriented programming to write a single line of Java code, as everything in Java must be an object!