Python - Lesson 2

Comments, Variables, Conditions, Control Flow, Standard Library, Calling Functions, Input/Output

Previously… in Lesson 1

- Computer programming
  - Languages, computers and humans are all dumb.
  - Python translates between human and machine.
- Programming languages
  - Difference in approaches between Assembly, C/C++, Java, Python, Bash etc.
  - Python allows us to get things done quickly, though it takes more effort for the computer to understand.

Previously… in Lesson 1

- What code looks like
- How to get Python on your computer
- Two methods of running: interpreter and script
- Data types - integers, floats, strings
- Basic maths - +, -, /, *, **, %
- Printing stuff - ‘print’ statement

I'll Take That as a Comment

- Writing code can be hard.
  - Reading code is HARDER!
- If I look back at code I wrote last year, I'd have no idea what it does.
  - Unless I made some comments about it.
- Code comments are normal English. You can put there almost anywhere in your code to explain WHY you have done something.

Comments

- There are two types of Python comments.
  - Single line comments - start your comment with a ‘#’.
  - Multi line comments - start and end your comments with a triple quote. This is called a ‘docstring’.
  - Good comments explain why you did something.

```
# This is a stupid comment, this code is obvious.
print 'Hello world' # Comments don’t have to be on their own line.

"""
This is a good comment, the code below is complex.
"""
x = 1 or fx(56, 10024, k2(**d, math.pow(kk, yd)), lambda x: x * xs(x))
```

Variables

- Last week we explored the idea of printing arguments

```
print "Hi! my name is %s." % "Simon"
```

- But what if we want to use ‘Simon’ in many places? It sucks typing it out every time.
- What if don’t know who’s name we want to print when we run this statement?

Variables

- The solution to this problem is to assign ‘Simon’ to a variable.
  - We assign values using the ‘=’ operator.
  - We can use a variable in the same way as we use any other value.

```
name = "Simon"
print "Hi! my name is %s." % name
print name, "is a stupid face!"
print name + "'s username is 'sallen'."
```

Variables - Why?

- We don’t need to know the value of the variable when we use it.
- If we change the value, we only need to change the value of the variable, not every place we use that value.
- Values can be looooooong. Variables save us from typing too much.

Go With the Flow

- Variables are fun, but still pretty useless on their own.
- We might need to make decisions on the basis of what a variable’s value is.
- We might want to do something many times. It sucks to write the same code over again.
- Control flow makes variables useful.

Compare the Pair

- Before we talk about control flow, you need to understand how to compare things.
- Python has several comparison operators.
  - x == y
    - Is ‘x’ equal to ‘y’? Also, note the double ‘=’.
  - x != y
    - Is ‘x’ not equal to ‘y’?
  - x < y
    - Is ‘x’ less than ‘y’

Compare the Pair

- x > y
  - Is ‘x’ greater than ‘y’?
- x <= y
  - Is ‘x’ equal to or greater than ‘y’?
- x >= y
  - Is ‘x’ equal to or less than ‘y’?
- x or y
  - Is ‘x’ or ‘y’ true?
- x and y
  - Is ‘x’ and ‘y’ true?

Compare the Pair

- x is y
  - Does ‘x’ point to the same place as ‘y’ in the computer’s memory?
    - This is different to x == y, but we will cover this later.
- not x == y
  - The ‘not’ keyboard reverses the condition.
    - In this case, this is the same as x != y.
- As in maths, you can place groups of conditions inside parenthesis.
  - not (x > y and (z == k or b > 0))

The ‘if’ Statement

- The ‘if’ statement is used to make decisions
  - We give it a condition, if it’s true, some code runs.
  - We need to end the ‘if’ line with a colon.
  - We need to indent the ‘block’ inside an ‘if’ statement.
  - We can put ‘if’ statements inside ‘if’ statements.

```
if name == 'Simon':
    print 'name is Simon!'

    if age == 42:# This is a nested ‘if’ statement.
        print 'age is 42!'

print 'we are finished making decisions!'
```

Oppan Gangnam Style

- We saw on the previous slide that code inside the ‘if’ statement needs to be indented to form what is called a ‘block’.
  - The Python style standard dictates that 4 space characters should be used to indent code.
  - A lot of Python developers disagree with this standard and state a tab should be used instead!
- My advice is to use tabs to indent code.

If, elif and else

- The ‘if’ statement has two optional friends.
- The ‘elif’ statement means “else if” and allows you to run some other code instead if another condition is true.
- The ‘else’ statement allows you to execute some code if the ‘if’ or ‘elif’ conditions are not met.

The ‘if’ Crowd

```
if name == 'Simon':
    print 'name is Simon!'

    if age == 42: # This is a nested if statement.
        print 'age is 42!'

    elif age == 53:
        print 'age is 53!'

    else:
        print 'age is something else'

    print 'we are finished with this nested decision'

elif name == 'Kramer':
    print 'name is Kramer!'

else:
    print 'name is something else'

print 'we are finished making decisions!'
```

The Standard Library

- Python programmers are lazy.
- Why code something when someone else has already done the work for you?
- This is where the standard library comes in.
  - Python’s standard library comes with several million lines of code that you can ‘call’ upon.
    - We’ll be covering the standard library more later.
  - To call this in-built code, we use functions.

Calling Functions

- A function is a block of code which can be ‘called’ any number of times by other code.
  - We’ll have a look at writing some later on!
- To run a function, just type the function’s name followed by any arguments the function accepts in parentheses.

```
car = "maibatsu Monstrosity"
print()    # Calling a function without any arguments.
print(car) # Calling a function with an argument.
```

Calling Functions

- Some functions need to be given arguments, some don’t.
- Functions can have any number of arguments.
- When a function finishes executing, it returns a value back to the block that called the function in the first place.

Calling Functions - Return Values

- A function like getPersonAge(name) would return a person's age.
  - We can use the return value, or give it to a variable.
- A function like setPersonAge(name) probably won't return anything.

```
age = getPersonAge('Simon')
print age # should return a number.

value = setPersonAge('Simon', 25)
print value # should return nothing (a 'NoneType' in Python terminology).
```

Input/Output

- Computers communicate with you in three main ways.
  - Standard input (stdin) - normally from the keyboard.
    - This is how you give the computer input.
  - Standard output (stdout) - normally to the terminal.
    - This is how the computer gives you output.
    - We've already seen this - the 'print' statement.
  - Standard error (stderr) - normally to the terminal.
    - This is where the computer puts errors.

Need Input

- In Python, you can give the computer input with the ‘raw_input’ function.
- You can give ‘raw_input’ an optional prompt for the user to answer.
- ‘raw_input’ returns what the user types in response.

```
name = raw_input("Enter your name: ")
print "You entered: %s" % name
```

Homework

- At this point, you know how to code Python!
  - Well… at least good enough to try it out for yourself.
  - Don’t quit your day job ;).
- Exercise:
  - You are writing a ‘corrupt cop’ simulator for the AFP. The simulated cop accepts bribes, which are given to the program via stdin, and writes to stdout an appropriate message, as if the cop is talking to you. If a bribe is under $3000, the cop is unimpressed. If the bribe is equal to or more than $3000, the cop will try and haggle. If the bribe is equal to or more than $6000, the cop will accept your offer and delete your parking offenses. Comment your code!