Python and Programming Basics

For Horizons Computer Science 2014

Introduction:

The following was adapted from an MIT introduction to computer Science course found online. It teaches you the basic constructs of programming (which you may already have a good feel for) and gives examples of each of their uses.

Our Programming Environment:

Programs are run by our computer. Each different programming language requires a different "environment" to run. To run our "Python" programs today we will be using an online environment. This allows us to avoid installing python on every computer here. Here is the link to the environment:

Click Here: http://www.tutorialspoint.com/execute_python_online.php

Our Task Today

Today you will be stepping through the examples provided in this PDF and reinforcing the basic programming knowledge you have gained over our last few meetings. I recommend you have this PDF open on one half of your screen and the compiler/ environment to run your program open in the other half. Although this PDF shows what the script will display when run, please type these programs yourself and try to get them running on your own, then check if your output matches the output of the examples.

Have Fun:

Feel free to stop at any time and play around with the programming stuff you have learned. For example, once you learn about loops, try making a loop that counts to ten thousand!

Section One: The Basics

3. **Writing Programs.** Recall that a program is just a set of instructions for the computer to execute. Let's start with a basic command:

print *x*: Prints the value of the expression *x*, followed by a newline.

Program Text:

```
print "Hello World!"
print "How are you?"
```

Output:

```
Hello World!
How are you?
```

(The compiler needs those quotation marks to identify strings – we'll talk about this later.) Don't worry, we'll be adding many more interesting commands later! For now, though, we'll use this to have you become familiar with the administrative details of writing a program.

5. **Variables.** Put simply, variables are containers for storing information. For example:

Program Text:

```
a = "Hello World!"
print a
```

Output:

```
Hello World!
```

The = sign is an assignment operator which says: assign the **value** "Hello World!" to the variable *a*.

Program Text:

```
a = "Hello World!"
a = "and goodbye.."
print a
```

Output:

```
and goodbye..
```

Taking this second example, the value of *a* after executing the first line above is "*Hello World!*", and after the second line its "*and goodbye..*" (which is what gets printed)

In Python, variables are designed to hold specific **types** of information. For example, after the first command above is executed, the variable *a* is associated with the **string** type. There are several types of information that can be stored:

- **Boolean**. Variables of this type can be either *True* or *False*.
- **Integer.** An integer is a number without a fractional part, e.g. *-4*, *5*, *0*, *-3*.
- **Float.** Any rational number, e.g. *3.432*. We won't worry about floats for today.
- **String.** Any sequence of characters. We'll get more in-depth with strings later in the week.

Python (and many other languages) are zealous about type information. The string "*5*" and integer *5* are completely different entities to Python, despite their similar appearance. You'll see the importance of this in the next section.

6. **Operators.** Python has the ability to be used as a cheap, 5-dollar calculator. In particular, it supports basic mathematical operators: +, -, *, /.

Program Text:

```
a = 5 + 7
print a
```

Output:

```
12
```

Variables can be used too.

Program Text:

```
a = 5
b = a + 7
print b
```

Output:

```
12
```

Expressions can get fairly complex.

Program Text:

```
a = (3+4+21) / 7
b = (9*4) / (2+1) - 6
print (a*b)-(a+b)
```

Output:

```
14
```

These operators work on numbers. Type information is important – the following expressions would result in an *error*.

"Hello" / 123 "Hi" + 5 "5" + 7

The last one is especially important! Note that Python just sees that 5 as a string – it has no concept of it possibly being a number.

Some of the operators that Python includes are

- **Addition, Subtraction, Multiplication**. *a+b*, *a-b*, and *a*b* respectively.
- **Integer Division**. *a/b*. Note that when division is performed with two integers, only the quotient is returned (the remainder is ignored.) Try typing *print 13/6* into the interpreter
- **Exponentiation** (a^b). a ** b.

Operators are evaluated using the standard order of operations. You can use **parentheses** to force certain operators to be evaluated first.

Let's also introduce one string operation.

- **Concatenation**. *a+b*. Combines two strings into one. *"Hel" + "lo"* would yield *"Hello"*

Another example of type coming into play! When Python sees a + b, it checks to see what type a and b are. If they are both strings then it concatenates the two; if they are both integers it adds them; if one is a string and the other is an integer, it returns an error.

Write the output of the following lines of code (if an error would result, write error):

print 13 + 6 Output: _____

*print 2 ** 3* Output: _____

*print 2 * (1 + 3)* Output: _____

print 8 / 9 Output: _____

Section Two: Repetition, Repetition, Repetition, ...

Tips/Notes:

- The **if** statement executes a sequence of statements only if some condition is true. This condition can be anything.

- **elif** / **else** is optional. Remember that at most one block of statements is executed. **else** occurs if none of the above conditions are satisfied.

```
1  a = 10
2
3  if a > 10:
4      print "wonderful, its over 10",
5  elif a < 10:
6      print "super, its under 10",
7  else:
8      print "cool, it is 10!"
9
```

Tips/Notes:

- A **while** loop allows us to repeat a sequence of statements many times.
- You can use almost *anything* for the condition. Not every while loop has to be a simple counter.

Problem 6: This is not a loopy title

What is the output of the following code? *If the code does not terminate, write error.*

Program Text:

```
a = 5
while a < 8:
  print "X",
```

Output:

LOOPS FOREVER

Program Text:

```
a = -1
while a < 3:
  print "X",
  a = a + 1
```

Output:

XXXX

Day 3: Nested loops

Tips/Notes:

- This isn't anything new, but we can put loops inside other loops. We can also do fairly crazy things: nest a **while** in an **if** in a **while** in another **while**.
- The **break** keyword exits the innermost loop.

Problem 8: Freebie!

What is the output of the following code? *If the code does not terminate, write error.*

Program Text:

```
a = 0
while a < 3:
  while True:
    print "X",
    break
  print "O",
  a = a + 1
```

Output:

XOXOXO

Section Three: Functions

Day 2: Functions

Tips/Notes:

- A **function** is just a named sequence of statements. We usually *define* functions at the beginning of code – definitions just associate the name with the sequence of statements.
- Functions can take parameters (within the parenthesis suffix) and can return information via **return**
- **return** is NOT a function. Like **if**, **while**, .. its a *keyword*: a basic command of the language.
- You can find out more information about functions using the *help(x)* function, e.g. *help(sqrt)*. Remember to write *from math import ** first.

Problem 10: Sanity Check

What is the output of the following code? *If the code does not terminate, write error.*

Program Text:

```
def f(a):
  a = a + 5
  return a

b = 0
f(b)
print b, ",",
b = f(b)
print b
```

Output:

```
0,5
```

Problem 11: Last but not least (somewhere in the middle)

You know that functions can call other functions, right? Here's an interesting fact – functions can call themselves!

Program Text:

```
def f(x):
  print "X",
  if x <= 1:
    return 1
  else:
    return x+f(x-1)
```

Section Four: Special Functions

Random numbers in programs can be used for a variety of of things, like playing a guessing game against the computer!

```
1  import random
2
3  random_number = random.randint (0,10)
4  print (random_number)
5
```

The above code would print a random

User input can also be important to programs allowing them to process data provided by the users. The following code sets the variable check_name equal to whatever the user inputs, it even gives the user the prompt message to ask for the input.

When we have that variable set you can do whatever you want with that information. You could even ask the user for multiple inputs one after one another, maybe try asking a few details like your name, age, grade, etc.

```
1  check_name = raw_input("What is your name? ")
2  print "Welcome to Horizons", check_name
3
```

Congratulations!

You have completed the Python lesson, you are now free to experiment with any of the aspects of python you have learned in Horizons, have fun!