
PYTHON ASSIGNMENT #2

Quick Review of Assignment #1

TERMS:

Variables: A place to store values in Python. Variables can be named any combination of letters, numbers, and underscores, as long as they don't start with a number. You can assign a value to a variable using the = operator (mysum = 4 + 1 would store the value 5 in the variable mysum)

Types: Variables have types. Variables of different types react differently with each other.

Floats and Integers: These are the two basic types of numbers in Python. Integers are whole numbers without decimals, such as -7, 0, or 42. Integers are useful for counting things. Floats are decimal numbers such as 1.5 or 3.14. Floats are most useful for doing calculations. You can do all the usual calculations with numbers in Python (+, -, *, /, %, **)

Strings: Pieces of text, such as "Hello World!" You can make a string by putting text in quotations. You can add (concatenate) strings together ("Hello" + "World" would give you "HelloWorld") and you can multiply strings ("Hello" * 3 would give you "HelloHelloHello").

Booleans: Values that are either true or false.

Module: Any Python file. Modules have a .py extension.

FUNCTIONS:

int(): Turns floats and strings into integers. For floats, int() simply snips off the numbers behind the decimal point – it does not round the numbers.

float(): Turns integers and strings into floats.

str(): Turns whatever you put into the () into a string.

print "message": Prints messages to the command line. Multiple messages can be printed in the same print command; just separate them with commas (eg: print "I have", 5 + x, "carriers",)

raw_input("prompt"): Reads a string from the keyboard after asking the question in the string "prompt"

Okay, that's a lot of typing. Is there is a better way Mr. Pelletier? YES! Let's define a function that prints out the repeating part of the song. This is how that would look like:

The def keyword begins a function definition.

stupidson() is the name of our function. We can use this function later by calling it's name. Function names MUST be one word!

These funny brackets here might look useless, but you'll see what you're supposed to do with them later.

The colon tells python that the rest of the code following is the definition of the function.

```
def stupidson():  
    print "This is the song that doesn't end,"  
    print "Yes, it goes on and on, my friend."  
    print "Some people started singing it, not knowing what it was,"  
    print "and they'll continue singing it forever just because..."  
    print
```

Notice how the function definition is indented? This is important! Python knows that all of the indented code is part of the function definition. Once you are done defining your function, stop indenting. Python figures out the rest. Although IDLE will auto-indent for you, if you ever do it yourself make sure you only use SPACES, not TABS. If you mix spaces and tabs, your program will FAIL horribly and you won't know why.

After we define our function, instead of re-typing the lyrics over and over again, we can just call the function a bunch of times, like this:

```
stupidson()  
stupidson()  
stupidson()  
stupidson()  
stupidson()
```

Wow, that is a lot easier! (There's also an easier way to repeat this function call, but one thing at a time)

What do the Funny Brackets Do? Functions and Arguments

What if you wanted to make a function to sing the Happy Birthday song? You can't hard code it because you don't know in advance who's birthday it is. Rather than write a different function for every possible name in the universe, you can pass your function arguments. Arguments (or "args" as programmers call them)

```
def happybirthday(name):  
    print "Happy birthday to you!"  
    print "Happy birthday to you!"  
    print "Happy birthday dear", name  
    print "Happy birthday to you!"
```

You can call this function for different people's birthdays now:

```
happybirthday("Dark Templar")  
happybirthday("Segata Sanshiro")
```

How the `return` Keyword Makes Functions More Versatile

Suppose we wanted to make a mathy function that found the reciprocal of a number. As far as you know so far, it would look like this:

```
def reciprocal(x):  
    print 1.0/x
```

 } This kinda sucks

But what if I wanted to use the reciprocal function in another equation? Maybe I wanted to add 1 to the reciprocal of x, or find the reciprocal of the reciprocal of x like this:

```
1 + reciprocal(2)  
reciprocal(reciprocal(2))
```

 } We want to be able to do things like this!

Currently our function won't work that way. When we call `reciprocal(2)`, the function prints something. You can't add 1 to a print command! Instead of printing the answer, we can instead make the function **EVALUATE** to the answer using the return keyword.

```
def reciprocal(x):  
    return 1.0/x
```

 } "return" lets us do these things!

Now you can include `reciprocal()` in equations! Also, you can still print out the answer. Just do it like this:

```
print reciprocal(2)
```

Some Built-In Python Math Functions

You don't have to write every function yourself. For example, a treasure trove of math functions live inside a module called "math.py" You can import the code from this module into your modules by using import:

```
from <module name> import <function name>
```

Fill in the <>'s with the right names, and you will be able to use these functions in your programs.

```
from math import sqrt
```

Some math functions include:

FUNCTIONS IN MATH.PY	DESCRIPTION
<code>sqrt(x)</code>	Evaluates the square root of x
<code>sin(x)</code>	Finds the sine ratio of angle x
<code>cos(x)</code>	Finds the cosine ratio of angle x
<code>tan(x)</code>	Finds the tangent ratio of angle x
<code>pi</code>	Evaluates very closely to the number pi

To import all of these functions, you can list them separated by commas:

```
from math import sqrt, sin, cos, tan, pi
```

Note: By default, degrees are measured in radians in Python, but the math module has ways of converting between degrees and radians. Can you guess what those functions names are?

Some Examples of More Complicated Equations in Python:

Math	Python
$y = -x + 2b$	<code>y = -x + 2*b</code>
$area = \frac{1}{2}\pi r^2$	<code>From math import pi area = 0.5*pi*r**2</code>
$f(x) = x + 1$	<code>def f(x): return x + 1</code>

EXERCISES

Exercise 1: oldmacdonald.py (5 marks) The instructions for this exercise are located in the oldmacdonald.py file available on my web site, but basically you're going to use functions to print out the song "[Old Macdonald Had a Farm](#)". The output of this program should be as follows:

```
Old Macdonald had a farm, e-i-e-i-o!  
And on this farm there was a duck, e-i-e-i-o!  
With a quack quack here and a quack quack there,  
Here a quack, there a quack, everywhere a quack quack!  
Old Macdonald had a farm, e-i-e-i-o!
```

```
Old Macdonald had a farm, e-i-e-i-o!  
And on this farm there was a cow, e-i-e-i-o!  
With a moo moo here and a moo moo there,  
Here a moo, there a moo, everywhere a moo moo!  
Old Macdonald had a farm, e-i-e-i-o!
```

```
Old Macdonald had a farm, e-i-e-i-o!  
And on this farm there was a mutalisk, e-i-e-i-o!  
With a woosh woosh here and a woosh woosh there,  
Here a woosh, there a woosh, everywhere a woosh woosh!  
Old Macdonald had a farm, e-i-e-i-o!
```

(Your program should also produce a forth verse with your own chosen animal and sound)

Exercise 2: distance.py (5 marks) The instructions for this exercise are located in the distance.py file available on my web site. Here is the distance formula:

$$x = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

The output of this program should be as follows:

```
>>>
The distance between (0,0) and (3,4) is 5.0
The distance between (-3,-4) and (3,4) is 10.0
The distance between (0,0) and (0,0) is 0
```

Exercise 3: compositeFunctions.py (6 marks) Write python functions that define the following mathematical functions:

$$f(x) = x - 1$$

$$g(x) = 2x$$

$$h(x) = 3x + 1$$

Each function should RETURN the answer when you plug in a number for x. For example f(3) would return the number 2.

After you have defined these functions, in the same module write a series of print statements that evaluate the following:

1. `f(10)`
2. `g(6)`
3. `h(3)`
4. `f(g(2))`
5. `g(h(10))`
6. `f(g(h(11)))`

EG: for #1, you would code: `print "f(10) = ", f(10)`

1 mark for each correct answer!