Introduction to Computer Programming with python™

Lesson 2: Building a Smarter Snake

Python has a great memory – just like your computer! Let's make our snake buddy remember things for us and recall them when needed.

Hands-on labs

You'll learn programming fundamentals: variables, types, simple input/output, and comparisons. You'll also practice writing, saving, and running your very first programs.

Exercise 2-1: Types and variables

Python understands different kinds of values. A value might be a *string,* which is programmer-speak for text; a number; or a true/false value, also known as a *boolean.*

Here's an example of a string:

```
'Snakes are awesome'
```

Did you notice those quotation marks? They can be single or double, as long as they match. Strings have to be marked off, or *delimited*, by quotation marks. **QUESTION:** Can you guess what the empty string would look like?

Here are some examples of numbers:

```
-256
3.14159
4.5e+12
```

Those are, respectively, an integer, a *floating-point number* or *float*, and a number in scientific notation (4.5×10^{12}).

The boolean values are pretty easy:

```
True
False
```

HINT: Python can also equate other types to true and false booleans! The number `0` is considered `False`, and `1` (or any non-zero number) is the same as `True`. An empty string is considered `False`, and any non-empty string as `True`.

There are other types, too, but we don't have time to cover all of them – different kinds of sets such as *tuples, lists,* and *dictionaries,* as well as powerful *classes*. (Feel free to research this on your own!)

Python can operate on strings as well as numbers. For example, you can add two strings using *concatenation.* (That's a mouthful!) Let's try some example expressions:

```
'Hello' + 'there'
'goo' * 4
'spam, ' * 5 + 'and eggs'
```

What does this tell you about how Python deals with spaces in strings?

Exercise 2-2: Simple input and output

Python uses *functions* to do all sorts of complicated things. Every function has a name, which is how you refer to it, and the name is followed by parentheses. In many cases, the parentheses may have one or more values inside. These are called *parameters* and they're passed to the function for it to take some action.

Input and output are among the most common types of functions seen in Python programs. Input functions allow the user to give some value to the program, while output functions allow the program to display or send information elsewhere.

The `print()` function allows you to send output to the console. Inside the parentheses you can include whatever value you want to output. Try some output on your own:

```
print(5 + 2)
print('5 + 2')
print('5 + 2 =', 5 + 2)
print('Python rocks')
print (3 * 'Rah ')
print('It\'s a', 21, 'gun salute')
```

Notice how the space in the fifth example doesn't affect the output. Try it both ways to confirm. Also, pay close attention to the sixth example. What's the deal with that backslash character? Why do you think it's there?

Now let's see how Python can take input from a user at a prompt:

```
input('Give me an R: ')
```

Feel free to answer however you like. Anything you put into the Python command line can have a *return* *value*. What is the return value for the `input()` function?

The `input()` function is useful, but what if you want to do something with that value later? What if you don't know yet what exactly should be printed, as we did above? This is where *variables* come into play.

Exercise 2-3: Variable awesomeness

Think of a *variable* in Python as a box in which you store something. The = symbol allows you to do the storing – it puts a value in the variable "box." A variable needs to have a name as well. The only rules are that you can't use the special names that Python reserves for itself. But it's a good idea to use a name that you can remember in your program and makes sense when people read it.

You can assign all sorts of things to variables. For each of the following problems, see if you can predict what Python will do after the last line. Take the challenge – don't run the commands until you write your prediction! Then see what happens.

```
x=3
print(x*7)
```

What do you think the return value is? __________

```
myval = input('Enter something: ')
print('You entered', myval)
```

What do you think the return value is? __________

```
print(myval * x)
```

What do you think the return value is? __________

Did any of the answers surprise you? Why do you think that happened?

Exercise 2-4: Your first program

Remember how we said a program is just a set of instructions that the computer executes in order? You're going to write your first program now. Good programs always have an objective or goal. The goal of your program is to ask the user for their name, and then to say hello back to them.

Open the Text Editor to write your program. When you are ready, save the program as a file named *hello.py* in your home folder. To try your program out, open a Terminal. Then run the following command:

```
python3 hello.py
```

If you get an error, and can't figure out what happened, raise your hand for some help.

Exercise 2-5: Comparing numbers

There are many expressions and operators available in Python for making comparisons. You can compare numbers and strings, as well as other things. Try these examples, and then make up some of your own. Before you try each comparison, write the return value you think the computer will output (*True* or *False*):

```
x = 'Charlie'
x is 'Fred'               ____________
x is not 'Fred'           ____________
x == 'Charlie'            ____________

x = 4
x is 3                    ____________
x is 4                    ____________
x == 'Charlie'            ____________

y = 7
x == y                    ____________
x != y                    ____________
x is not y                ____________
x > y                     ____________
x <= y                    ____________
```

For the next problem, write a comparison expression for *y* that will return a *True* value:

```
x = 5
y = x + 3
y == _____
```

Exercise 2-6: Comparing strings

You can also compare strings. String comparisons are based on the value of the characters they contain. For instance, `A` comes before `B` so the value of `A` is less than the value of `B`. When comparing two strings, the value of the first character of each is compared, followed by the value of the second, and so on.

Try these comparison expressions, and again write your predictions for *True* or *False* before you run the expression:

```
s1 = 'Charlie'
s2 = 'Fred'
s1 == s2
s1 < s2
```

Feel free to try some of your own comparisons of strings.

Bonus Exercise: Apples and oranges

What happens if you try to compare a string with a number?