The Components of a Computer Program

Functions, Variables, and Logic

The instructions within a computer program have three main components: ***functions***, ***variables***, and ***logic***. Broadly speaking,

- ***Functions*** do things (like multiply two inputs together and output the result)
- ***Variables*** store things (like numbers, text, or the outputs of functions) and
- ***Logic*** makes decisions on the basis of conditions that are either true or false ('if the number is < 0, print out the word 'negative").

Sometimes functions do things to inputs that a user types out. All useful functions, though, produce an output. A variable is simply a named place to store a value that can change.

Data Types

Variables always have a type, which is the ***data type*** of the value the variable stores. Data types are types of things, with certain common features, that are built into Python. For example, the Python code:

```
1   numberOfGiraffes = 4
```

Makes a variable, calls it 'numberOfGiraffes', and puts the ***integer*** (i.e., whole number) 4 into it. The '=' sign is called the ***assignment*** operator, and assignments should be read from right-to-left:

the thing on the right, whatever it may be, is being put into the variable whose name is on the left. The variable numberOfGiraffes is an integer variable because it stores an integer.

Variable Names

Variable names are:

- Decided by us (we could've made a variable called 'numGiraffes' or 'favouriteGuys', instead)
- Case-sensitive
- Must start with a letter
- Can't have any gaps in them

Because of the kind of thing numberOfGiraffes stores (namely, an integer) we can do mathematical operations on numberOfGiraffes; like subtraction and division. This is because, in Python, all integers have certain features and abilities in common, like the feature of being positive or negative, and the ability of being addable to other numbers.

On the other hand, the Python code:

```
1  firstName = 'Harry'
2  myAnimals = ['pig', 'parakeet', 'cat', 'aardvark']
3  JesseHappy = True
```

Makes three variables, the first called 'firstName', the second called 'myAnimals', and the third called 'JesseHappy'.

Strings

The type of the variable firstName is a ***string***: a sequence of characters, numbers, or special characters such as punctuation or whitespace, all enclosed in quotation marks. Other strings include 'Hello World!', 'hello World!', 'this is a string' and '874hfasl'. Like all data types, all strings have certain common features and abilities built into them.

Lists

The variable myAnimals is a list variable because it stores a ***list***. A list is another data type recognized by Python, and is just a collection of objects enclosed within square brackets. Like with integers and strings, all lists have certain common features: their elements are ordered by indices, starting from 0 (for example, the element of myAnimals at index 0 is the string 'pig', and

the element at index 2 is the string 'cat') their elements can be changed, and they can have the same item more than once (so I could add 'pig' again to the end of myAnimals, if I wanted).

Bools and Control-Flow

The variable JesseHappy above is a Boolean variable because it stores a ***bool***. Bools – also known as Booleans, after the logician George Boole – have one of two values: True, or False. Bools allow us to make ***control-flow structures*** that prevent certain instructions from being executed *unless* some bool variable or condition evaluates to True. Control-flow structures are so-called because they control the flow of our piecemeal instructions, which are executed, by default, in order from the first instruction to the last. This is because computers execute programs ***linearly:*** something called the ***interpreter*** goes through each instruction line by line and executes each (or tries to; if there's a syntactical error, for example, it'll stop). Jesse is the name of a dog, so we made this variable to store information on his well being. Check out this code:

```
1  while (not JesseHappy):
2      feed(Jesse)
3      pet(Jesse)
4      walk(Jesse)
5  exit()
```

Notice the use of indentation here. Assuming feed(), pet(), and walk() are functions we've defined that work on dogs, the above code will feed, pet and walk Jesse *if and only if* the variable JesseHappy's value is False, and hence the condition (not JesseHappy)'s value is True. Those instructions to feed, pet and walk Jesse on lines 2, 3 and 4 will *only* execute if the condition (not JesseHappy)'s value is True, which will only be the case if the variable JesseHappy has the value False (poor Jesse). The word **'not'** is a word built into Python, and simply reverses the truth-value of the Boolean variable or condition that follows it. As soon as JesseHappy's value is True, the value of (not JesseHappy) will be False, and the exit() instruction will execute.

Because Python is ***dynamically typed***, we can make a variable (for example numberOfGiraffes) but a certain type of thing in it (for example an integer) and then, later, put a different type of thing entirely into it (for example, a bool). Cool, huh? By contrast, ***statically typed*** programming languages allow us to only put one type of thing into a given variable that we make.

---

print() function

Python has a handy way for us to check what the values of variables and Boolean conditions are: its built-in print() function. Not all functions are built into Python (we can create our own using Python). Some of the built-in functions can be freely used, while others have to be imported in useful blocks of ready-made code others have written called ***libraries***. Print() is one of the most basic and easiest to use built-in functions. If we run this:

```
1  print(numberOfGiraffes)
```

Then Python handily outputs for us:

```
4
```

Comparison Operators

Because this is the value we gave to that variable. We can also compare items, and the items of variables, with Python's fantastic range of built-in operators. Check out this example of the '**==**' **operator** (which checks for equality) and the '**!=**' **operator**, which checks for inequality. Using these operators, we make those Boolean conditions we've seen, whose value is either True or False:

```
1  print(4 != 3)
2  print(numberOfGiraffes == 4)
```

The above code, when run by the computer, outputs:

```
True
True
```

This is because the code prints the value of the condition (4 != 3) which is True, followed by the value of the condition (numberOfGiraffes == 4), which is also True (because the value of the numberOfGiraffes variable is indeed 4).

Fun isn't it! Being shown basic features of Python is like being shown some basic phrases in German or Japanese, or any language you don't understand: there's a logic to it, a similarity to what you already know, but it takes time. It's also thrilling and opens so many new doors.

Mathematics in Python

Now, since all operations (like mathematical addition '+' or subtraction '-', division '/' or multiplication '*') only work on certain *types* of operands (like integers) we have to be very aware of the *data types* that our operations accept. Operators in programming are fussy about these. Sometimes, one and the same symbol can do different operations when applied with operands of different types. For example, the code:

```
1  print(4 + 3)
2  print('4 '+ '3')
```

Outputs:

```
7
43
```

Why is this? Well, in line 1, '+' implements the mathematical addition operation, and in the second line, '+' implements the ***string concatenation*** operation on the short strings '4' and '3'. **String concatenation** just joins strings together to make a new, longer string. So although the '43' in our output above looks like a number, it won't behave like one: it's actually just the string '43', with two characters, the numeral '4' followed by the numeral '3'. If we tried to divide it, multiply it, or do anything mathematical to it, Python would *throw us an error*: essentially telling us: 'Hey! This isn't the kind of thing you can do that with!'.

Type-Conversion

Luckily, though, we can ***cast*** (or type-convert) our variables with Python's built-in functions for ***type-conversion***. We can also check the type of our variables with the built-in function type(). So the code:

```
1  print(type(numberOfGiraffes))
```

Outputs:

```
<class, 'int'>
```

Which just means that the variable numberOfGiraffes is an integer variable, which makes sense, because we chose to put the number 4 into it.

Check out this code:

```
1  numberOfGiraffes = 4
2  text = str(numberOfGiraffes)
3  print('I have ' + text + ' giraffes.')
```

On line 1 above, we're assigning 4 to the numberOfGiraffes variable. (This is the value that variable already had, but re-assigning it the same value doesn't do any harm). Remember: always read these assignment instructions (involving '=') from right to left: the thing on the right is going into the variable whose name is on the left. On line 2 we're converting the value of numberOfGiraffes to a string, using Python's built-in **str()** function. On the same line, I'm putting that new string into a new variable, called 'text'. I'm then printing the result of applying the string concatenation operator, using the '+', on 'I have ', our string variable text, and the string ' giraffes'. Here's what's produced:

```
I have 4 giraffes.
```

You can check out Python's range of built-in functions for type conversion here. (This resource is a great reference to use when working with built-in functions for type. Feel free to skip over any reference to number bases for now.) Finally, variables don't have to be manually defined. We can get inputs from the user of our program with the built-in ***input()*** function. Note though: in Python 3.7, user inputs are by default strings, so if we want to do mathematical things to user inputs, we have to cast those inputs to number types. Within the brackets of the input() function, we can put a prompt; such as 'please enter a number: '. For example, the code:

```
1  userOpinion = input('What do you believe? ')
2  print('You think that ' + userOpinion + '.')
```

Outputs:

```
What do you believe?
```

If the string 'giraffes are sly guys' is then typed out on the keyboard and input by the user, we get the following output:

```
You think that giraffes are sly guys.
```

Conclusion

Well done! You've made it through this intense intro to computer programming and Python. You'll apply all of this new knowledge to Python lessons and projects you'll work on throughout the rest of the course. We've seen some nice examples of the basic components of computer programs - functions, variables, and logic - but done in the language that's relevant to you: Python. All other programming languages implement functions, variables, and logic, but the syntax - the rules for grammatical correctness - vary. The silver lining, though, is twofold: first, Python is one of the most versatile, powerful, and intuitive languages out there, and second, once you learn one programming language, the rest are so much easier to grasp.

If you're curious, you can read more about type conversions here, data types here, and control flow structures here. But don't worry; none of this is obligatory. We've done enough for us to progress for now. Let's crack on!