Advanced Python - Session 2

User input, functions, loops, conditional statements

Learning outcomes

- Get input from a user
- Write a function (also known as a “method”)
- Create and query an array (also known as a list)
- Write an iterative statement (a “for” loop)
- Use logic and Boolean operators
- Write a conditional (“if/else”) statement

Recap from last session

- What is a string?
- What is a variable?
- What is a formatted string?
- How do you print a formatted string to your command line?
- What are the two number types we learned about last week and what’s the difference between them?

Let's get started!

Getting input from a user

From last week’s session we now know how to create variables and how to print them, but what if we don’t know what the value of our variable should be, because we need to get it from a user? By using the method raw_input() we can prompt a user for information.

Task

1. Create a new file called **my_name.py**. Copy and paste the code below into the file, and run it in your command line:

```
print "What’s your name?"
name = raw_input()
print "Hello {}!".format(name)
```

Cool, right? Now we can get information from a user and use Python to do stuff with it.

Introducing functions

During this course you might hear the words “method” and “function”. We use them interchangeably, because they’re basically the same thing but different programming languages gravitate towards one name or the other.

A function is a block of code with a name assigned to it. Functions are really handy, because you only have to write them out in full once, and then you can use them again and again anywhere in your program by “calling” them (more on what that means in a bit).

The concept is a little bit like a paragraph. If you think back to English class in school, a paragraph is “a distinct section of a piece of writing, usually dealing with a single theme and indicated by a new line, indentation, or numbering” (Google Dictionary).

In Python, a function is a really similar idea – it’s a distinct block of our code that deals with a single task/theme, and is formatted in a particular way. Just like a paragraph, a function can be one statement or many statements long.

Here’s an example of a really simple function:

```
def hello_world():
        print "Hello World!"
```

Any time you write a function in Python, it’ll have the following things:

- def at the very beginning, so that Python knows the indented code below is part of a function (you’re “defining” your function below this line).
- A unique name, with no spaces in it. It’s important that you don’t have two functions with the exact same name in your code. You also don’t want to name your function “function”, “sum” or “python” (or later on in this course “email”, “mailgun”, or “tweepy”). If you name your function after a Python library or package or built-in function, Python will get really confused and your code won’t run.
- You’ll also need to put brackets and a colon right after the name.

Notice how, after the first line of code in a function, the other lines are indented. In Python, this is very important – it’s how the code interpreter in your computer knows what’s part of your function and what isn’t.

Calling a function

A function by itself doesn't actually do anything. You have to "call" it by its name, to run the code inside it. Calling a function is really easy! All you have to do is type the name of your function with () after it, like this: hello_world()

Notice that the function call is NOT indented, because it's not part of the function definition!

Task

1. Create a new file called **my_first_function.py** and write a function that prints your name in the command line, instead of Hello World!

Add another line of code so that your function also prints the answer to 2 + 2

Arguments

We've all had those moments when our computer says "no" to our code and we say "but my code looks good! Why isn't it working?!" – but that's not the kind of argument we're talking about here!

Every function you write in Python will have () after its name. Sometimes, you might want or need to put some values/variables/parameters between those brackets, so that they can be used inside your function. The values inside those brackets are called "arguments". Sometimes you might seem them referred to as "args". Depending on the function you're using, you might have no arguments, one argument, or more than one argument. If you're writing your own function, you get to define this!

Let's use the built-in Python function range() as an example. We don't have to write the code for it because the lovely people who came up with Python have done that for us already. From Python's documentation, we know that range() can take 1, 2, or 3 arguments (you can read more about this function and how it works here).

Task

1. Create a new file called **my_arguments.py**, paste the code below into it, and run it. What do you think range() is doing when it's given 1, 2, and 3 arguments?

2.

```
print range (10)                    #one argument
print range (1,10)          #two arguments
print range (1,10,2)                #three arguments
```

More fun with functions

Here's a function that adds two numbers together and prints out the answer. Remember, it doesn't actually DO anything until we call the function using the statement add_two_numbers().

```
def add_two_numbers():
        number1 = 1
        number2 = 2
        answer = number1 + number2
        print "{} plus {} is {}".format(number1, number2, answer)

add_two_numbers()
```

Here's another function that adds two numbers together. Notice how this function takes two arguments: number1 and number2.

```
def add_two_numbers_from_args(number1, number2): #These can only be variable names
        answer = number1 + number2
        print "{} plus {} is {}".format(number1, number2, answer)

add_two_numbers_from_args(5,10)      #These can be variable names or actual numbers
```

Task

1. Copy the code immediately above into your **my_arguments.py** file, and run it.
2. What happens when you change the values of the arguments you used when you called the function?
3. What do you think happens if you don't put 2 arguments in when you call the function add_two_numbers_from_args?

You must use variable names for your arguments when you're defining your function (you can't put the actual values of the numbers or strings there). This is so you can re-use your functions, because the actual values for the arguments are given when you call the function, just like you see in add_two_numbers_from_args(5,10). Python's smart, and knows that when you call that function, the first value inside the bracket is number1 and the second value is number2.

Returning stuff from functions

You can use functions to do things like crunch numbers or manipulate data, and then "return" information from your function so that you can use it somewhere else in your program.

Functions can return a number, a string, a list, or even another function (we'll learn about one called render_template() later on in this course). Today, we're going to learn how to return a simple number value from your function.

Let's have a look at the code below:

```
def add_two_numbers_and_return_value():
        number1 = 1
        number2 = 2
        answer = number1 + number2                              # answer = 3
        return answer                                           # 3

returned_value = add_two_numbers_and_return_value()

print returned_value
```

The variable names answer, number1 and number2 only exist inside this function. So, even though you're returning answer from your function, it's only the *value* associated with the variable that's being returned (i.e. 3) not the actual variable *answer = 3*.

In order to be able to use this value we just returned, we need to assign a new variable name to it so we can do stuff with it. That's what we're doing with returned_value = add_two_numbers_and_return_value(). Notice how we can then do stuff with returned_value after that, like print it.

If you're returning something, the return statement should be the last line you want to run in your function. Once the Python interpreter reaches that line, it won't execute anything in the function after it (it'll go back to your main program and run the rest of the statements you have there).

Task

1. What do you think returned_value will be if you don't have a return line in your function? Copy the code from the example above into your **my_arguments.py** file and run the file. Then remove only the line return answer and see what happens.

Lists (also called arrays)

Just like a shopping list, you can use a list in Python to store items. You can create a list just like you create a variable:

```
empty_list = []`
list_of_numbers = [0, 1, 2, 3, 4, 5]
my_favourite_fruits = [“pineapples”, “oranges”, “bananas”]
mixed_list = [15, “sunshine”, “jumper”, 4, “sky”]
```

Here are some things to remember about lists:

- List items go between square brackets
- Items in a list are separated by commas
- Strings in a list must be in quotation marks

You can access an item in a list by using its name and its index number, like this:

```
print my_favourite_fruits[0]
```

In the world of programming, the first item in a list is always at position 0 (its “index” position). This means that [0] is the first item in the list, [1] is the second item, and so on.

Using “logic” in programming

Logic is an important concept in computer programming. Just like a “Choose Your Own Adventure” game, we can write programs that can take a specific action based on whether a condition is satisfied (“True”) or not satisfied (“False”).

An example we use every day is signing into a service. If both your username and password match what’s in the service’s database when you’re signing in, you’re logged in; but if they don’t you’re not allowed access.

In a Choose Your Own Adventure game, you might ask a player to choose between Door #1 and Door #2, and what happens next in the game is based on which door the player chooses.

Here’s a list of logic operators and what they “evaluate”:

OPERATOR	WHAT IT CHECKS
and	Are both/all conditions true
or	Is at least one condition true
not	(self explanatory)
!=	not equal to
==	equal to
>	greater than
>=	greater than or equal to
<	less than
<=	less than or equal to
True	Is it true
False	Is it false

Just like in maths class, if you have things inside of brackets to evaluate, deal with the things inside the brackets first:

STATEMENT TO EVALUATE	ANSWER
1 == 1	True
1 == 0	False
(1 == 1) and (1 == 0)	(True) and (False) → False

“For” loops

“For” loops are also called iterative loops, because they iterate (run through their code repeatedly) until there’s nothing left to iterate through.

This concept makes more sense if we go back to what we learned about lists earlier.

Imagine you’re going to a coding party with some friends from class, and you need to go to the supermarket to get supplies. Keeping with the Python theme of your party, you bring your laptop with you, make a digital “shopping cart” in Python, and add stuff to it using .append(), which you read about in Python's documentation.

Before you get to the checkout counter, you want to print the contents of your cart, to make sure you have everything. Here’s how you’d do that in Python, using a for loop:

```
my_shopping_cart = ["cake", "plates", "plastic forks", "juice", "cups"]

for item in my_shopping_cart:
        print item
```

The only condition the Python interpreter cares about in your for loop is whether there’s anything left to iterate through. Since you’ve only got 5 items in your shopping cart, your loop will only run 5 times. Python’s clever that way.

A few helpful things about for loops:

- The code that’s part of your for loop must be indented, just like when you write a function. Just like a function, make sure you also don’t forget the colon at the end of the first line!
- For loops don’t return anything (they’re not functions)
- This is the format that you need to use when you write a for loop:

```
for x in name_of_thing_to_iterate_through:
        print x
```

The x can be called anything – you could call it “x”, or “bananas”, or “count”, or “item” – it’s basically just a placeholder name. But the name of the thing you’re iterating through is important – it’s the name of your list, so that Python knows what to iterate through.

You can do lots of things with for loops, including adding things to lists and deleting things from lists (or files, and other things). The homework for this session has some more exercises you can explore.

Conditional statements (“if” statements)

We talked about logical operators a bit earlier, and how they’re useful because we can use them to write programs that can take a specific action based on whether a condition is satisfied or not.

You can think of each conditional statement as a different “branch” or “path” of code that you can follow – a bit like a fork in the road, you have to choose one path to follow because you can’t be on both at the same time.

If you have a conditional statement that evaluates to True, then Python will execute the block of code that immediately follows the if statement. Where the statement evaluates to False, Python will skip that block of code. Just like a function and a for loop, the code you want to run when a condition is True must be indented.

Task

1. Copy and paste the code below into a new file called **numbers.py** and run the file.

```
number = raw_input("Enter a number between 1 and 10: ")
number = int(number)      #Converts the input string to an integer

if number > 10:
        print "Too high!"

if number <= 0:
        print "Too low!"
```

1. What happens if you enter a number that’s greater than 10?
2. What happens if you enter a number that’s less than 0?
3. What happens if you enter a number between 1 and 10?
4. What happens if you remove the line with number = int(number)?

If none of your conditions are met, Python will simply carry on through the rest of your code. In the example above, we didn’t write any code for the situation where the user enters a number between 1 and 10, so you didn’t see any sort of helpful message or feedback in your terminal.

Notice how we used int() to convert the user input into a number. raw_input() treats all input as a string, so we used information in Python’s documentation to figure out how to convert the user input to a number. You don’t need to worry about this right now, but it’s an example of something useful you can do with Python.

Else statements

An else statement is like a catch-all for when none of your conditions are satisfied. That means you can only have one else statement, and it can’t exist on its own without an if statement. If you think you need more else statements, you actually need more if statements!

Using else statements isn’t mandatory, but they’re handy for debugging because they can help to catch unexpected behaviour in a program.

Advanced Python - Session 2

Task

Using the code we just looked at above:

1. Add an if statement that prints a message if you enter a number between 1 and 10

Instead of the if statement you just wrote, use an else statement to do the same thing (this is a bit of a trick question!)

You might come across something called an elif statement ("else if"), for example, in your homework exercises. You can ignore it, and use if instead because it does the same thing as an if statement.

Best practice tips for writing functions:

- Try to give your functions descriptive names so it's easier to understand what they do just by looking at the name (you can also use comments to help with this)
- Try not to write giant functions! It's best to keep each function related to a specific task – this makes them easier to reuse.
- Sometimes it helps to sketch out on paper what you're trying to do before you start coding

Homework

1. Review the examples from today's session to make sure you understand them. If you get stuck, don't be shy about asking an instructor for help!

2. Complete exercises 11, 12, 19, 21, 27, 29, 30, 21, 32, and 34 on Learn Python The Hard Way. Don't worry about doing the study drills that ask you to write things out 50 times or upside down – the point of the homework is to help reinforce what you've learned in class, so use your judgement on what's most helpful for your learning journey.