Functions

(adapted from Andy Colley's Repl.it curriculum)

What Is A Function?

A function is a set of actions that is given a single name (identifier) in a very similar way to a variable or list.

They help us break our program down into smaller individual sections that are easier to test and reuse. This becomes more important as we start to write longer and more complex programs.

A function has to be defined (created) before it can be called (used) by the program.

There are lots of built in functions in Python, but this guide is focused on how to create your own. The code for defining (creating) a function looks like this:

```
def function_name():
    #Code to run when function is called goes here
    return data # this is optional, it is not always needed
```

A function can be called at any time during the program by typing its identifier. The code for calling a function looks like this:

```
function_name()
```

This is really useful if your program needs to perform the same task multiple times at different points during the runtime - instead of writing the code for the task multiple times, we would create one function and call it as many times as we need. In this set of lessons we will be creating functions that carry out tasks using the skills we have previously learned (input, output, loops, etc).

A function can also return a result (data) to the main program as its final action. The code to return data goes inside the definition part of the program like this:

```
return data
```

When the function is called, we usually store the returned data in a new variable. The code for this is below:

```
new_variable = function_name()

print(new_variable)
```

PLEASE NOTE - a lot of the functions in the following tasks are very simplistic. They've been broken down to show how to define & call functions but you wouldn't really write functions for these tasks. Functions become progressively more useful as your programs become more complex.

Functions - Key Concepts/Vocab/Misconceptions

A function is just a ‘package’ for code. You can write any code inside a function, including calling other functions (we’ll get on to that bit later).

- A function has to be defined (created) before it can be called (used) by the program.
- A function can be called at any time during the program by typing its identifier.
- Functions allow us to code common tasks once and reuse them many times. This helps make our program smaller and more efficient.
- Function names are created using lowercase letters with underscores between words. This is not a syntax rule but it is the common convention. Using this instead of camel case helps us differentiate between functions and variables/lists when we are reading the code.
- A function will not run when it is created (using the def command). It has to be called in the program.
- A function can have many parameters. They are separated by commas in the brackets when the function is defined.

You may hear the terms ‘procedure’ or ‘subroutine’ referred to in other programming languages. In more complex languages these are other ‘flavors’ of functions that work in slightly different ways. However, at this level Python handily combines them all into functions.

1 - Functions With Input & Output

Tasks

- Task and instructions
- Example solution

Task 1

```
 # The program below uses functions.  Add comments to predict the output in the order that it has been coded here.

def say_hi():
    print("Why hello there!")

def offer_drink():
    print("Would you care for a spot of tea?")

def offer_food():
    print("Biscuit?")

def say_bye():
    print("Cheerio then.")


offer_drink()
say_hi()
offer_food()
```

`

Task 2

- Rewrite the code to call the functions in the correct order so that the output makes sense chronologically.

- Call the correct function so that the program says goodbye to the user.

Task 3

- Define a new function that tells the user a joke (you decide on the function name and the joke). Call it in a sensible place in the program.

2 - Functions That Return A Result

Functions can also return a result (some data) to the main program.
The code for this is:

```
    return data
```

This line of code is usually the last one in a function. If a function has multiple ways it can end, you may also have multiple return statements.

To call a function that returns data, create a variable in the main program and assign the function to it. Here is an annotated example of the whole process.

Creates & names the function

```
def adder():
    # Stores two numbers in two variables.
    num1 = 10
    num2 = 15

# Adds the variable contents together and returns the total to the main program

    return num1 + num2

# Calls the adder function and stores the data returned
output_num = adder()

# Outputs the data in the outputNum variable
print(output_num)
```

Note the indentation levels in the code above. Make sure you are clear what is part of the function and what is outside the function.

Tasks

- Task and instructions
- Example solution

Task 1

```
 # Add comments to explain what the output from this program will be and how you know.

def math1():
    num1 = 50
    num2 = 5
    return num1 + num2
```

```
def math2():
    num1 = 50
    num2 = 5
    return num1 - num2

def math3():
    num1 = 50
    num2 = 5
    return num1 * num2

output_num = math2()
print(output_num)
```

Task 2

- Adapt the code from one of the functions above to create a new function called 'multiplier'.
- The user should be able to input two numbers that are stored in variables.
- The function should multiply the two variables together and return the result to a variable in the main program.
- The main program should output the variable containing the result returned from the function.

3 - Functions With Arguments

In addition to getting data out of functions, we can put data in. We do this using parameters and arguments. (These are sometimes lumped together under "arguments" but there is a distinction between the two.) You can think of parameters as variables used by the function. They are named in the brackets after the function name when the function is defined. If there is more than one parameter, they are separated by commas.

When a function is called, the variable in the parentheses that is "fed" into the function is called an argument. In short, parameters are inside the function and arguments are outside the function.

In this first example we will feed data into the function by using a specific number as an argument. This is done by typing it specifically into the brackets when we call the function. This function has one parameter called num1. We will put the number 42 into the function.

```
def add_five(num1):
    print(num1 + 5)

add_five(42)
```

The second example performs the same task, but this time the user can input a number rather than have it fixed to 42. A variable called user_input is used to store the value typed by the user. That value is then used as an argument when the function is called, passing it into the function as num1.

```
def add_five(num1):
    print(num1 + 5)

user_input = int(input("Enter a number"))
add_five(user_input)
```

We can make this code more efficient by combining the last two lines as shown below.

```
add_five(int(input("Enter a number")))
```

For now, consider sticking with 2 steps for this until you feel more comfortable with the control flow.

Tasks

- Task and instructions
- Example solution

Assignment - Calculator

Define four functions - add, subtract, multiply, divide that add (or multiply etc) two numbers and return the result. Each should have two integer number arguments.

The user is asked to input two numbers. These numbers will be passed as arguments into one of the functions.

The user is asked to input 1 to add, 2 to subtract etc.
If they input 1, call the ‘add’ function, input 2 calls the ‘subtract’ function etc.

Output the returned result as part of a sentence.