Lesson 4: Functions, Libraries, Dictionaries

Recap of Last Lesson:

- *Lists*
    - We can use 1 variable to store many pieces of information using a list.
    - When we have a list variable, we can do things like lookup values in the list (using [] operator), change the values, add new values, etc.
- *Loops*
    - We use loops because we need to repeat certain actions in code.
    - In Python there are two types, while loops and for loops.
    - While loops will run as long as its condition is True.
    - For loops will run until it has hit the last item in a list.

Motivation for Functions

We used loops to repeat steps in code, but that only works when we have a condition. Sometimes we have certain pieces of code that we use so often that we can't just put it in a loop. For example, we do something at startup and then sometime later we have to do it again.

```
# Program start
# Calculate how much money we can spend
chequeing_accnt_balance = 500
savings_accnt_balance = 1000
credit_card_debt = 350
amount_we_can_spend = chequeing_accnt_balance + savings_accnt_balance + credit_card_debt
# .... some time goes by ....
# We bought something....
purchase_price = 50
credit_card_debt += purchase_price
# We need to recalculate how much we can spend
amount_we_can_spend = chequeing_accnt_balance + savings_accnt_balance + credit_card_debt
```

Question:

- What if this program needs to also consider money in a different bank account?
    - We need to update the code at program start, and later on when we recalculate it. If we only update it in one place, we will have a logical error.

Another reason we need functions - sometimes we want to be able to **almost do the same thing again…**

```
apple_price = 10.99
tax_rate = 0.13
purchase_price = apple_price * (1 + tax_rate)


orange_price = 8.99
tax_rate = 0.13
purchase_price = orange_price * (1 + tax_rate)
```

In this example we are doing the same calculation, but **with a different parameter… one being apple_price and the other orange_price.** This problem gets worse if we have 10s, 100s, or 1000s of products to calculate the price of.

Functions

What are they?

Functions are pieces of code that can be **invoked/called** at any time. We can pass **parameters/arguments** to functions as a way of changing their behaviour.

How do we write them?

```
def print_welcome_message():
    print("Welcome to functions!")
```

In Python **we use the def keyword** which tells Python we would like to define a function**.** We **then give our function a name**, in this case print_welcome_message. We then **define any arguments**, put a colon (:) and then define the code of the function.

How do we call functions?

We call/invoke functions by calling their name, and passing any parameters to them that they need inside of brackets ().

```
print_welcome_message()
```

Parameters and Return Values:

If we want a function to have parameters, we define their names in a list inside of the brackets in the definition. The code inside a function can then use those parameters.

A **return value** is a value that is given back to the caller.

```
def calculate_total_cost(sticker_price, tax_rate):
    total_cost = sticker_price * (1 + tax_rate)
    return total_cost
```

When we call calculate_total_cost, we give it a price and tax rate. It **returns the total cost to us** and we can then **assign it to a variable** (or do nothing with it, or use it immediately like in a print statement)

```
cost = calculate_total_cost(10.99, 0.13)

print(calculate_total_cost(10.99, 0.13))
```

Default Parameter Values:

```
# With a default tax rate
def calculate_total_cost(sticker_price, tax_rate=0.13):
    return sticker_price * (1 + tax_rate)
```

```
cost = calculate_total_cost(10.99)
```

Questions:

- Is print() a function? What does the print function do?
- What would happen if we created a python file with 1 function in it but it isn't called?
- How could we change the default tax_rate?
- What value will `cost` have if we call `cost = calculate_total_cost(“A string”, 5)`?

Activity:

- Write a program that prints a rectangle of X’s on the screen based on parameters **length** and **width.** Call it 3 times with different l/w values to confirm it works.

Libraries

What are they?

Libraries are what we use when we'd like to use **someone else's code**!!! If I asked you to write a function that calculates the sine of an angle, you'd have a tough time… but thankfully somebody else has figured it out already. All we need to do is **import their code into ours** and **invoke their code**. When we build cooler projects that use graphics, or connect to other services over the internet, we will use libraries to make our lives easier.

How do we use them?

We use an **import** statement in our code to bring library code into our own. Here's an example:

```
import math
for angle in [0, math.pi/2, math.pi, math.pi*3/2, 2*math.pi]:
    sin = round(math.sin(angle), 4)
    print("The sine of {} is {}".format(angle, sin) )
```

Because we imported the "math" library we were able to use some of it's code - here we used math.pi (a constant value) to get the value for pi, and math.sin(), which is a function that returns the sine of an angle in radians.

Python Standard Library

When you installed Python you also installed a copy of some standard libraries that Python have created for us to use (see list here: https://docs.python.org/3/library/index.html).

The standard library has functions that help us do things like:

- Open, create, or edit files
- Make HTTP and other networking requests/connections
- Debug our code
- Do many things at the same time… which we won't learn for a while

Other ways to write import statements

```
from math import pi
from math import sin as sine_of_angle
sine_of_angle(pi)
from math import * # now we wouldn't have to write math.__ ,
```

Activity:

- The **random** library helps us generate random numbers. The **random.rand()** function generates a random float between 0 and 1 and **random.randint(a, b)** will return a random integer between integers a and b.

  Write a program that keeps 'rolling' a die (value between 1 and 6) until the user correctly guesses the value of the die.

  Update the program so that you store each roll in a list. At the end of the game (user guesses correctly), you will print the max, min, and average of the rolls. You should use built-in functions **max(), min(), sum()** and **len()** to do this.

Dictionaries

Motivation

If you were programming and you needed to store information (in variables) about a user's account, which has a username, password, and history (list of webpages they've visited recently) how many variables would you need? **You'd need 3 per account you need to track.**

We want to have 1 variable to hold everything.

What are they?

To find the definition of a word in a dictionary, you look it up using its name.

In code, dictionaries work the same way.. except we call the words **keys** and definitions **values.** When you have a dictionary, you're allowed to add more information to the dictionary, remove some info, or look it up.

How do we use them?

When we used lists, we defined them with []. Dictionaries are defined in a similar way but we use {}.

```
user = {}

user["username"] = "Bob"

user["password"] = "password123"

user["history"] = ["www.facebook.com/profile/uaiuhds",
"www.facebook.com/photo/ausidhfis"]
```

```
coding = {
    "pros": [
        "it's fun",
        "you'll be smarter than all your friends",
        "you can build cool stuff",
        "you can get a job"
    ],
    "cons": []
}
```

Other Fun Facts

Understanding dictionaries is pretty important when you write programs that fetch information over the web. Here's an example "response" that a server might return to us:

{"name":"scott","gender":"male","probability":0.99,"count":31815}

Looks a lot like a dictionary, right?

Where are we in learning "how to code"?

Even though you're never done learning code, we've almost covered every **fundamental programming concept** (that's used nowadays). The only thing left to do is to discuss object-oriented programming, which we'll talk about next time.

Activities/Homework:

Hints:

- **len()** return the length of a list or string that you pass as a parameter (len("ABC") is 3)
- You can lookup individual characters in a string using **[]** operator ("ABC"[1] returns 'B')
- You can use the keyword **in** to check if something is in something (1 in [1,2,3] returns True, "hello" in "hello world" returns True)

1. Write a function that takes in a list and returns the sum of all the numbers in the list. Don't use the built-in function sum().

2. Write a function that takes in some possible password and return True if the password is acceptable, otherwise it returns False. The rules for passwords are:
    a. Must be between 8 and 16 characters long
    b. Doesn't contain any '#', '$', or '%' characters
    c. Does not contain the word "password"
3. Write a function that returns True if the given string is a palindrome (same forwards and backwards). is_palindrome("racecar") == True but is_palindrom("apple") == False.
4. Write a program that asks the user for the name of a file. The program opens the file and prints the contents to the console (refer to https://www.w3schools.com/python/python_file_open.asp)
5. Write a function that takes sticker_price, tax_rate, and return a **dictionary that contains** the sticker_price, tax_rate, and the total cost.
6. Write a function that returns a random school report card. It should look like {"students": [{"name":"", "math": __, "science": __, "english": __, "geography":__}]}

    Note: students is a list, so there could be multiple students on the report card. Your function could return between 1-10 students, each with random names (up to you how to do that), and with random grades between 1 and 100.

7. **CHALLENGE:** Write a program that reads a file called **schedule.json** and which tells the user whether the desired schedule is possible (there are no scheduling conflicts).

    available_classes are the classes offered at school. They each have a name (some string), a start and end time (numbers between 0 and 23.99).

    desired_classes are the names of the classes you want to take.

The file looks like:

```
{
    "available_classes": [
        {
            "name": "1001",
            "start_time": 7,
            "end_time": 8.5
        },
        {
            "name": "1003",
            "start_time": 8,
            "end_time": 11.5
        },
        {
            "name": "500",
```

```
            "start_time": 13,
            "end_time": 15.5
        }
    ],
    "desired_classes": [
        "1001", "500"
    ]
}
```

For this input, the program should print “Thats a valid schedule!”.

If “desired_classes” was [“1001”, “1003”] it would print “That schedule doesn’t work!” because class 1003 overlaps with the time 1001 runs.

Hint: Use the JSON library and the built-in function open() to get the information out of the file into a dictionary:

```
import json

f = open('FILE NAME')

data = json.load(f)
```