technoteach

technocamps

Llywodraeth Cymru Welsh Government

Prifysgol Abertawe Swansea University

Cardiff University Prifysgol Caerdydd

Prifysgol Bangor University

Cardiff Metropolitan University | Prifysgol Metropolitan Caerdydd

University of South Wales Prifysgol De Cymru

Cyngor Cyllido Addysg Uwch Cymru Higher Education Funding Council for Wales hefcw

Prifysgol Cymru Y Drindod Dewi Sant University of Wales Trinity Saint David

Prifysgol Aberystwyth University

Prifysgol Glyndŵr Wrecsam | Wrexham Glyndŵr University

institute of coding in wales technocamps

technoteach technocamps

Python – Data Analysis

Python & MatPlotLib

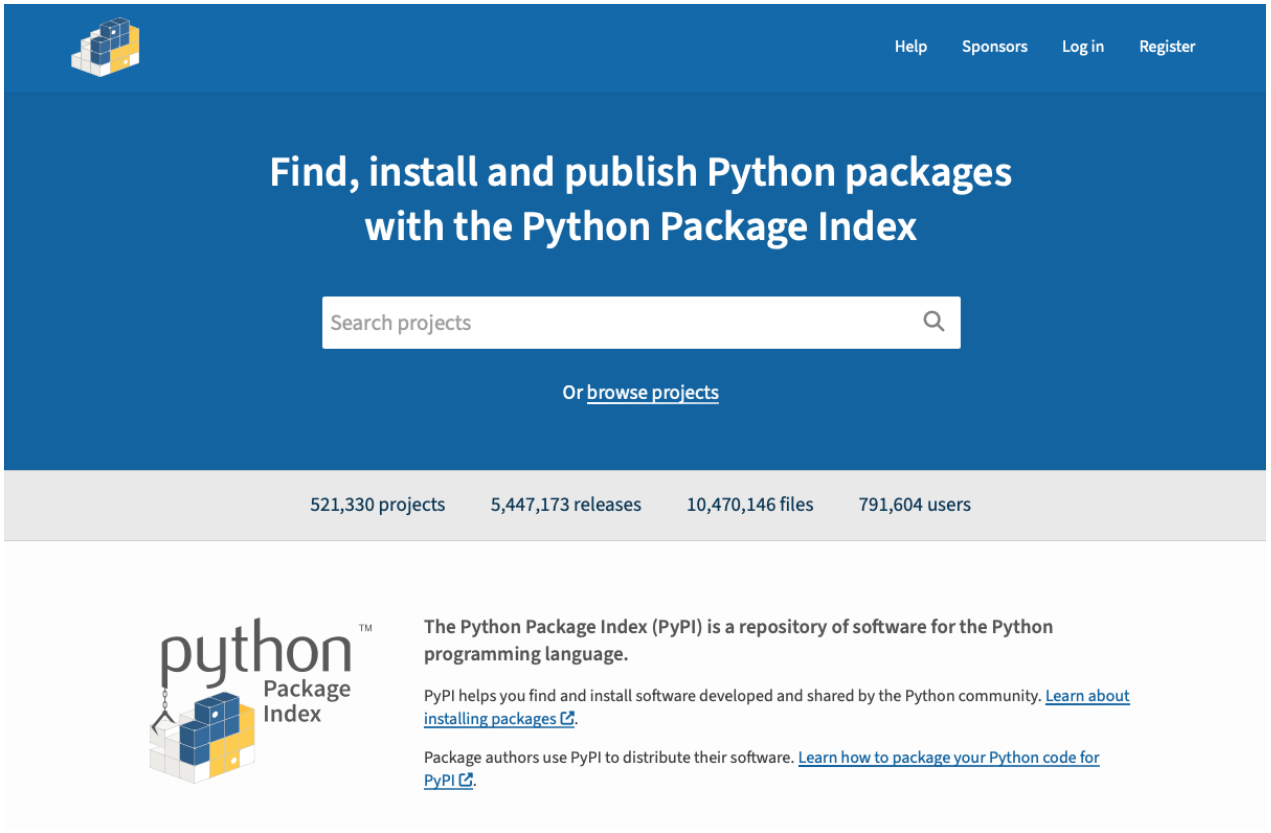
Python is an open-source programming language, meaning that it is free, all the source files are freely accessible, and there is a large developer community writing new libraries and able to offer support.

There are countless libraries available for Python, many of which we will be using in various depths. Today we will be focusing on Python's most popular data visualisation library, called MatPlotLib!

Why Use Python?

- A concise and simple interpreted programming language
- Extensive Standard Library
- Python Developer Community is very large
    - Meaning there are many new and maintained libraries and plenty of individuals able to offer support
- Open-source
    - Meaning that it is free to use and that all the source files are also freely accessible

Repository of Software



Help Sponsors Log in Register

Find, install and publish Python packages with the Python Package Index

Search projects

Or browse projects

521,330 projects 5,447,173 releases 10,470,146 files 791,604 users

The Python Package Index (PyPI) is a repository of software for the Python programming language.

PyPI helps you find and install software developed and shared by the Python community. Learn about installing packages.

Package authors use PyPI to distribute their software. Learn how to package your Python code for PyPI.

https://pypi.org

Python IDEs

All software development could be performed within simple text editors and compiled/executed from the terminal.

However, using an Integrated Development Environments (IDE) offers language specific tools such as: code completion, syntax highlighting, code analysis, code refactoring, project organisation and management, debugging tools, plugins… and many more!

There are many (IDEs) available for Python, find the best for you!

JupyterLab / Google Colab

Today we will be using Google Colab as our development platform. This uses the Project Jupyter Notebook interface for Python.

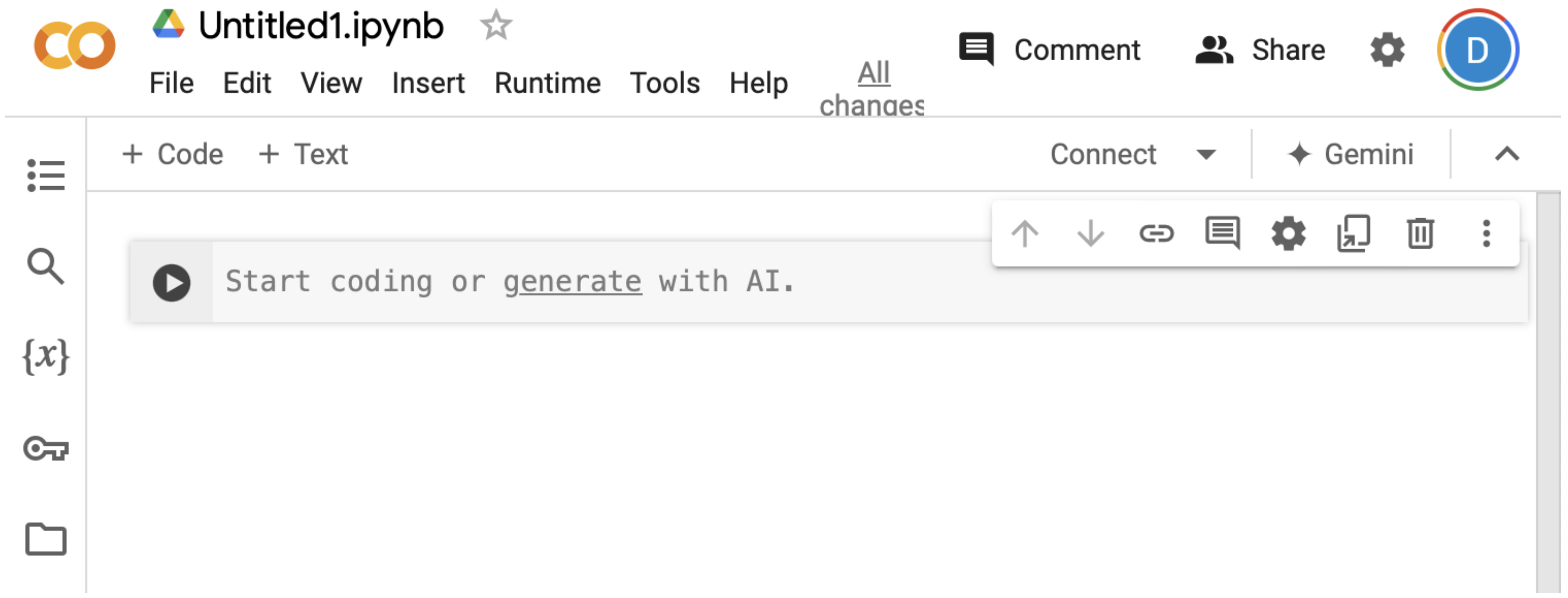
Essentially, JupyterLab allows you to write Python code in small executable chunks, instead of one large executable program.

This is a great tool for learning or demonstrating Python, as you can observe your data as it is altered with each execution.

Colab

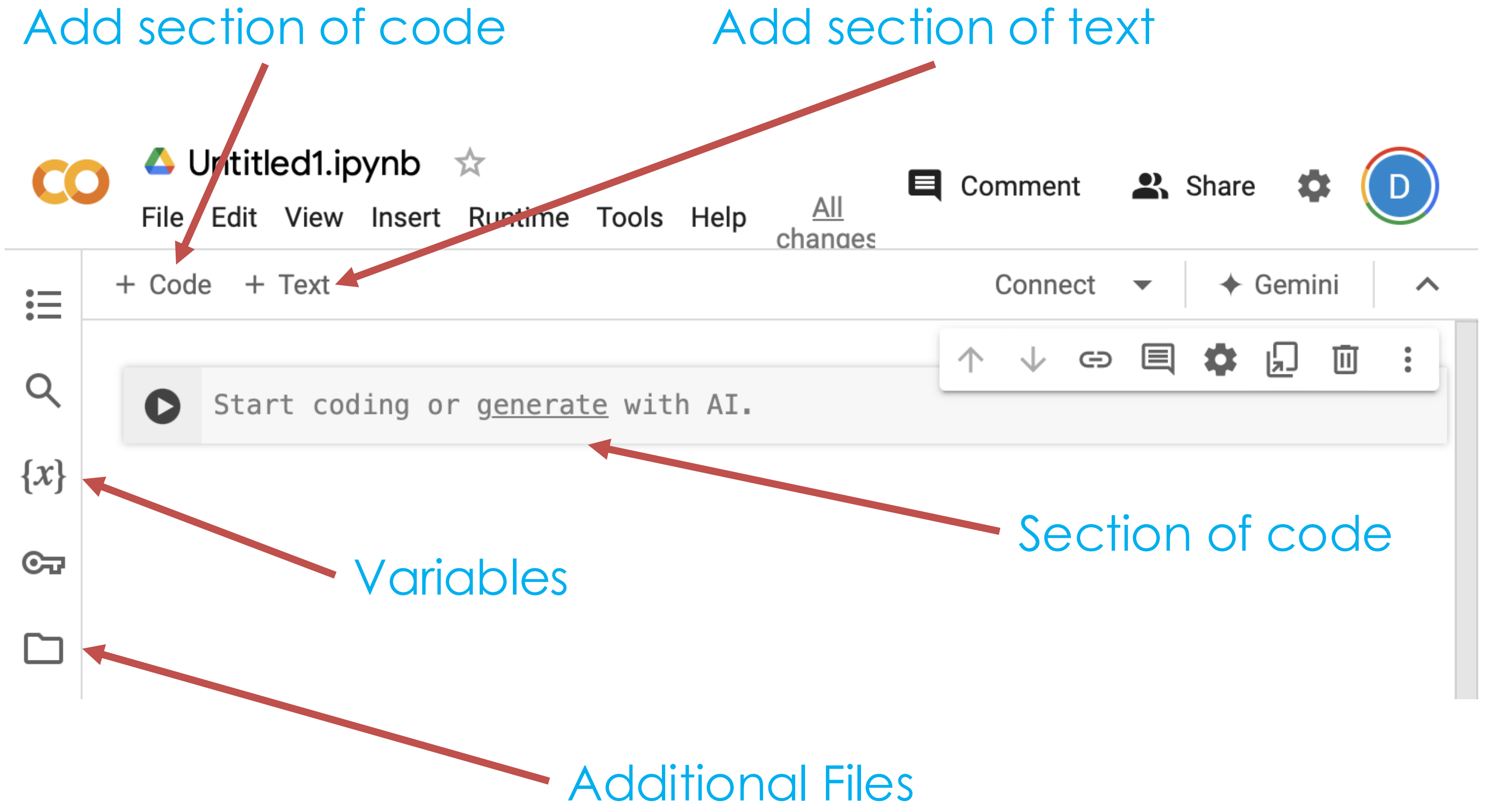
Log into Google Colab with your **personal** email.

Go to **File** > **New Notebook in Drive**.



Start coding or generate with AI.

Colab



Add section of code
Add section of text
Untitled1.ipynb
File
Edit
View
Insert
Runtime
Tools
Help
All changes
Comment
Share
D
+ Code
+ Text
Connect
Gemini
Start coding or generate with AI.
Section of code
Variables
Additional Files

Activity: Output

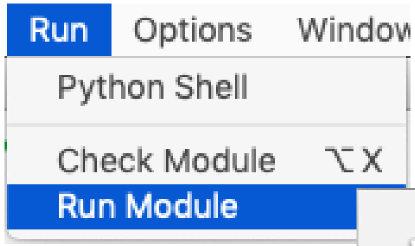
Copy the program below.

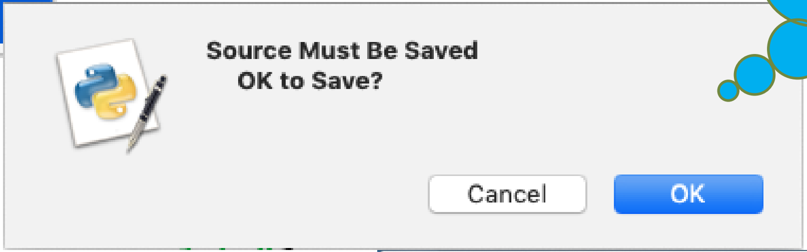
```
print("hello world! ")
print(2+2)
print("2+2")
```

Activity: Output

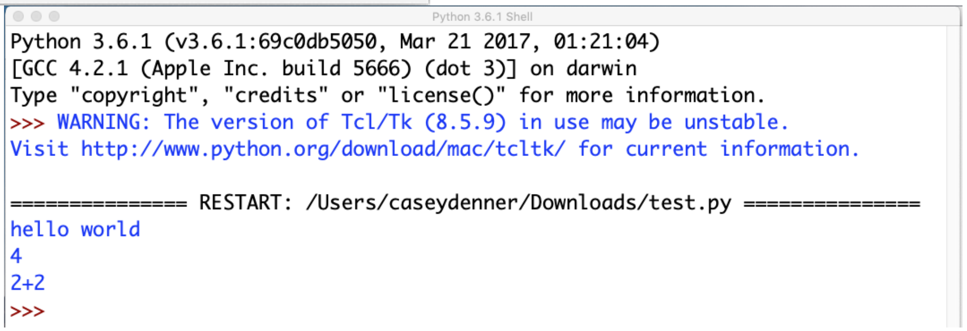
Run your program by clicking Run > Run Module

It may ask you to save your code first.





Source Must Be Saved
OK to Save?

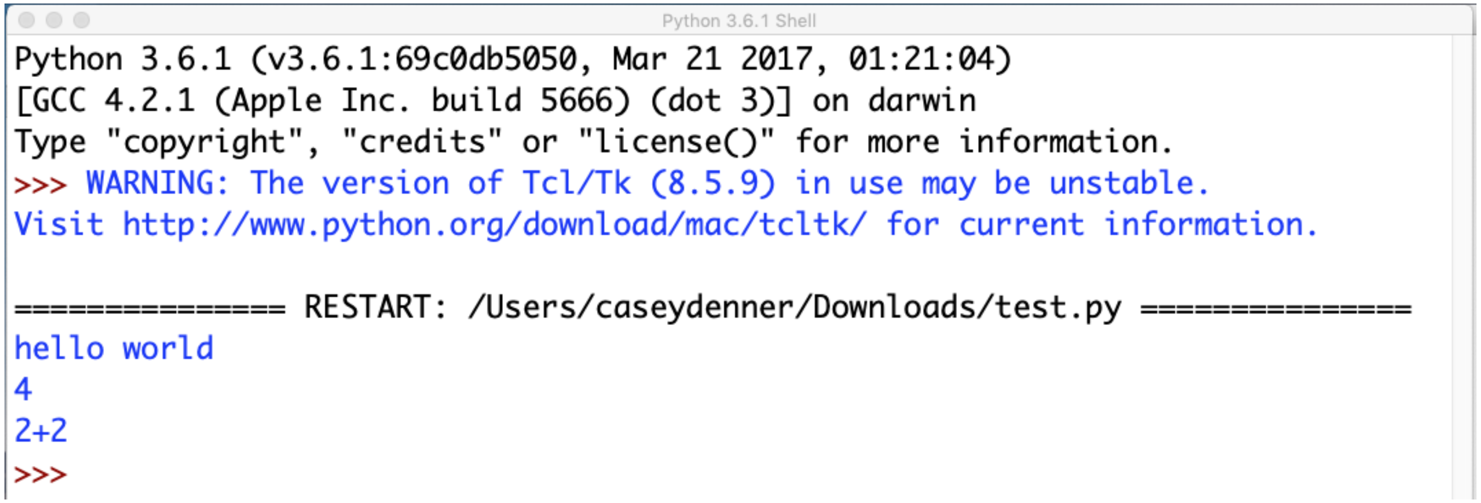


```
Python 3.6.1 (v3.6.1:69c0db5050, Mar 21 2017, 01:21:04)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>> WARNING: The version of Tcl/Tk (8.5.9) in use may be unstable.
Visit http://www.python.org/download/mac/tcltk/ for current information.

=============== RESTART: /Users/caseydenner/Downloads/test.py ===============
hello world
4
2+2
>>>
```

Review Output

What was the outcome of running the program?

What was the difference between line 2 and 3?

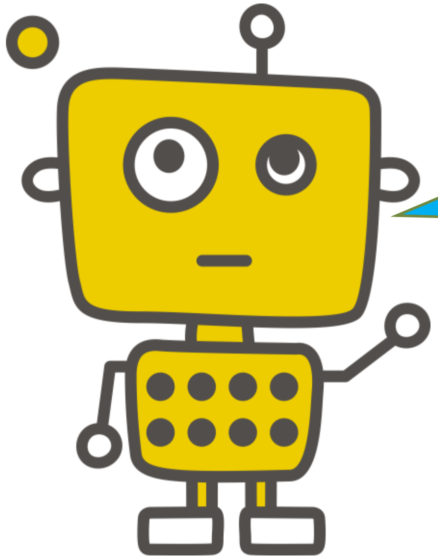


```
Python 3.6.1 (v3.6.1:69c0db5050, Mar 21 2017, 01:21:04)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>> WARNING: The version of Tcl/Tk (8.5.9) in use may be unstable.
Visit http://www.python.org/download/mac/tcltk/ for current information.

=============== RESTART: /Users/caseydenner/Downloads/test.py ===============
hello world
4
2+2
>>>
```

Variables

Variables are a storage placeholder for texts and numbers.

Variables should have logical names as they can be read and edited further on in the code.



Variables should not contain spaces! They should also use camelCase. E.g.
thisIsCamelCase
myAge
myName

Variables

```
name = "Casey"
```

```
name = "Luke"
```

Casey
name
name
Luke
name

Variables

```python
age = 20
name = "Casey"
print(name, "is", age, "years old")
```

Variables

Syntax
Variables in python should be in camelCase. E.g. firstName, lastName, etc. They should also be named sensibly!

Activity: Hello _____

Write a Python program that stores your name as a variable and then prints "Hello *yourname*".

Data Types

Python sets the variable type based on the value assigned to it. It is a **dynamically typed language.**

For example:

```
var = 10 #This will create a number integer assignment
var = "text" #the `var` variable is now a string type.
var = True #the `var` variable is now a boolean type.
```

Hashtags allow programmers to write comments within their code. It will not affect the program nor be visible to anyone except the developer.

If Statements

If statements are conditional statements.
If something is true it will execute the next line of code.

```
age = 15
if age >= 17:
    print("you can drive if you have a licence")
```

What would happen if the above code was executed?

If Statements

What would happen if the below code was executed?

```
age = 17
if age >= 17:
    print("you can drive if you have a
license")
```

If-Else Statements

```python
age = 15
if age >= 17:
    print("you can drive if you have a license")
else:
    print("you are not old enough to drive")
```

If-Elif Statements

```
age = 15
if age >= 17:
    print("you can drive a car if you have a license")
elif age == 16:
    print("you can drive a moped if you have a license")
else:
    print("you're too young to drive anything")
```

If Statements Using “and” and “or”

If statements can also use “and” and “or” to compare values.

```
age = 17
passed = False
if age >= 17 and passed:
    print("You can drive")
elif age >= 17 and not(passed):
    print("You can apply to do your test")
else:
    print("You're too young to drive")
```

passed is a Boolean.
A Boolean is either
True or False.

If Statements Using “and” and “or”

If statements can also use “and” and “or” to compare values.

```
a = 10

b = 9

c = 15

if a > b or a > c:

    print("a condition is met")
```

Activity: Can You Drive?

Write a program that has two variables:

- Your name
- Your age

Using if-elif-else statements check if you're old enough to drive:

- If you are then it should print: yourname is yourage years old. Yourname is old enough to drive.
- If you are not old enough then it should print: yourname is yourage years old. Yourname is not old enough to drive.

Extension: Use a Boolean to check if the person has passed their test and utilise this to check if they can drive.

Input

Python can also ask users for input.

```
name = input("What is your name?")
print("Hello", name)
```

Activity: Can You Drive? Input

Go back to your if statement program. Edit it to ask for the person’s name and age before it prints if they can drive or not.

technoteach technocamps

PyShop

Throughout these workshops you will be writing lots of programs contributing to your PyShop.

You're welcome to either create new files each time and copy and alter the old information, or you can edit the same file continually.

When you see this symbol: The activity is related to the overall PyShop.

Activity: Login

Create a login screen for users. It must do the following:

- Store a username
- Store a password
- Check if the input username **and** password are correct
- If correct print "Welcome"
- If incorrect print "Login failed"

Don't forget to write comments to explain what your program does when needed!

Activity: Login Solution

```
username = "casey"
password = "secret"

inUsername = input("Please enter the username: ")
inPassword = input("Please enter the password: ")
if inUsername==username and inPassword==password:
    print("Welcome")
else:
    print("Login failed")
```

Casting Input

Sometimes programs ensure users' input is of a certain type.

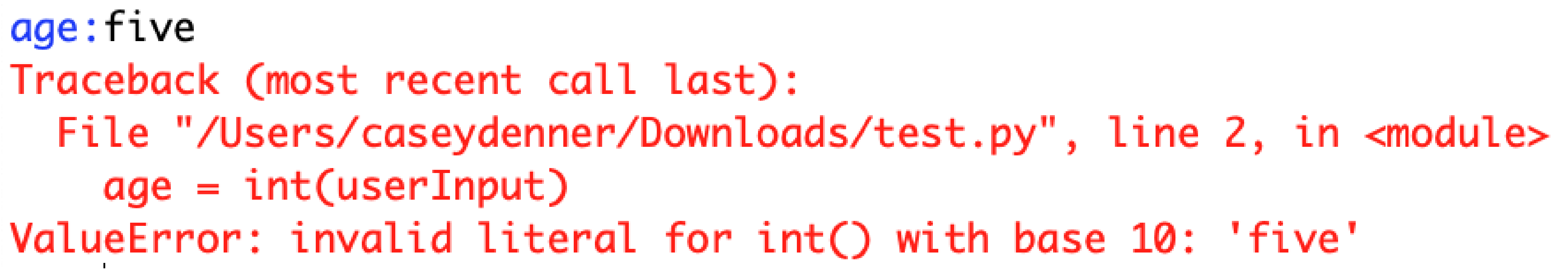
For example, if a user enters an ID number, we'd expect them to type out an integer like 1234 not one thousand and thirty four.

Casting Input

In Python we can cast input to be of a certain type like this:

```
userInput = input("age:")
age = int(userInput)
print(age)
```

However if we were to enter a string then it would throw/raise the following error:



```
age:five
Traceback (most recent call last):
  File "/Users/caseydenner/Downloads/test.py", line 2, in <module>
    age = int(userInput)
ValueError: invalid literal for int() with base 10: 'five'
```

Exceptions

A program can **handle** or **raise exceptions**.

- When the program handles an exception it deals with it and then continues to run.
- When an exception is raised the program comes to a halt and displays our exception to screen, offering clues about what went wrong.

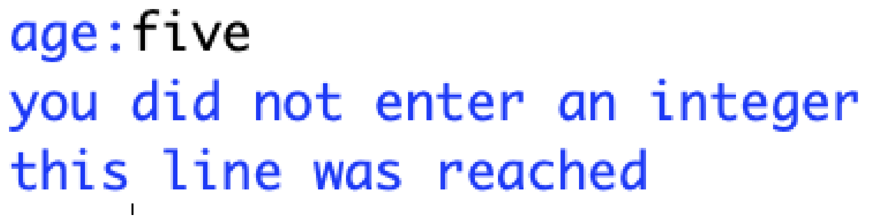
Handling Exceptions

To avoid the ValueError we need to handle the exception.

- The try and except block in Python is used to catch and handle exceptions.
- Python executes code following the try statement as a "normal" part of the program.
- The code that follows the except statement is the program's response to any exceptions in the preceding try clause.

Handling Exceptions

```python
userInput = input("age:")
try:
    age = int(userInput)
    print(age)
except ValueError:
    print("you did not enter an integer")
print("this line was reached")
```



```
age:five
you did not enter an integer
this line was reached
```

Raising Exceptions

Use **raise** to throw an exception if a condition occurs. The statement can be complemented with a custom exception.

Raising Exceptions

```
month = 2

day = 30

if month==2 and day>=29:

      raise Exception("invalid date")

print("This line was reached")
```

Notice this line
was not reached.

```
Traceback (most recent call last):
  File "/Users/caseydenner/Downloads/test.py", line 7, in <module>
    raise Exception("invalid date")
Exception: invalid date
```

Activity: Menu

Extend your login program so that once a user is logged in a menu appears that fulfils the following:

- Gives the user 3 options – 1. View, 2. Add, 3. Delete
- Users must select an option by entering the appropriate number
- When the user enters their option it provides feedback of which option was selected
- If they enter something which is not an option the program should provide feedback
- Input errors must be handled

Activity: Menu Solution

```
username = "casey"
password = "secret"
inUsername = input("Please enter the username: ")
inPassword = input("Please enter the password: ")

if inUsername==username and inPassword==password:
     print("Welcome")
     userInput = input("Please select an option:
                                \n1.View \n2.Add \n3.Delete \n")
     try:
          selection = int(userInput)
          if selection == 1:
               print("You selected view")
```

Activity: Menu Solution

```
        elif selection == 2:
            print("You selected add")
         elif selection == 3:
            print("You selected delete")
         else:
            print("Invalid selection")
      except ValueError:
         print("Please enter an integer to perform a
               selection")
   else:
      print("Login failed")
```

Loops

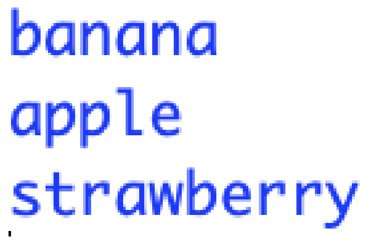
Two types of loops exist:

- For Loops
- While Loops

For Loops

A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).

```
fruits =
["banana","apple","strawberry"]
for f in fruits:
    print(f)
```



```
banana
apple
strawberry
```

For Loops

```
for x in range(6):
   print(x)
```

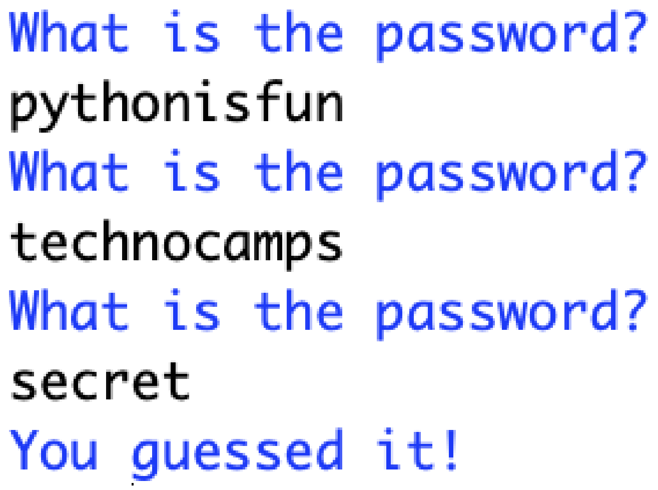
```
0
1
2
3
4
5
```

Notice how the range is 6 but it only prints out the values 0-5!

While Loop

A while loop can execute a set of statements as long as it is true.

```python
password = ""
while password != "secret":
    password=input("What is the password?\n")
print("You guessed it!")
```



```
What is the password?
pythonisfun
What is the password?
technocamps
What is the password?
secret
You guessed it!
```

Activity: Login and Menu Loop

Edit your program so that the login and the menu loops if incorrect input is given. For example:

- If the user enters the wrong password and username they should be prompted to input again.
- If the user enters a number above 3, or some text, for the menu selection they should be prompted to enter again.

There are a few ways to do this it depends on how you write your menuLoop.

technoteach technocamps

Activity: Login and Menu Loop Solution

```
username = "casey"
password = "secret"
loggedIn = False


while not(loggedIn):
    inUsername = input("Please enter the username: ")
    inPassword = input("Please enter the password: ")
    if inUsername==username and inPassword==password:
        print("Welcome")
        loggedIn = True
```

Activity: Login and Menu Loop Solution

```
menuLoop = True
    while(menuLoop):
        userInput = input("Please select an option: \n1.View \n2.Add \n3.Delete \n")
        try:
         menuLoop = False
           selection = int(userInput)
           if selection==1:
               print("You selected view")
           elif selection==2:
               print("You selected add")
```

Activity: Login and Menu Loop Solution

```
                elif selection==3:
                    print(“You selected delete”)
                else:
                    print(“Invalid selection”)
                    menuLoop = True
            except ValueError:
                print(“Please enter an integer to perform a selection”)
                menuLoop = True
    else:
        print(“Login failed”)
```

Functions

There are often sections of the program that we want to re-use or repeat. Chunks of instructions can be given a name - they are called **functions**.

- Programming languages have a set of **pre-defined** (also known as built-in) functions and procedures.
- If the programmer makes their own ones, they are **custom-made** or **user-defined**.

Functions

Parameters can be added and used in functions

```
def print_hello():
    print("Hello")

print_hello()
```

```
Hello
```

```
def print_welcome(name):
    print("Welcome",name)

print_welcome("Casey")
```

```
Welcome Casey
```

Syntax
Functions in python should be lower case and each word should be separated by an underscore. E.g. do_something(), print_hello(), etc.

Functions Execution

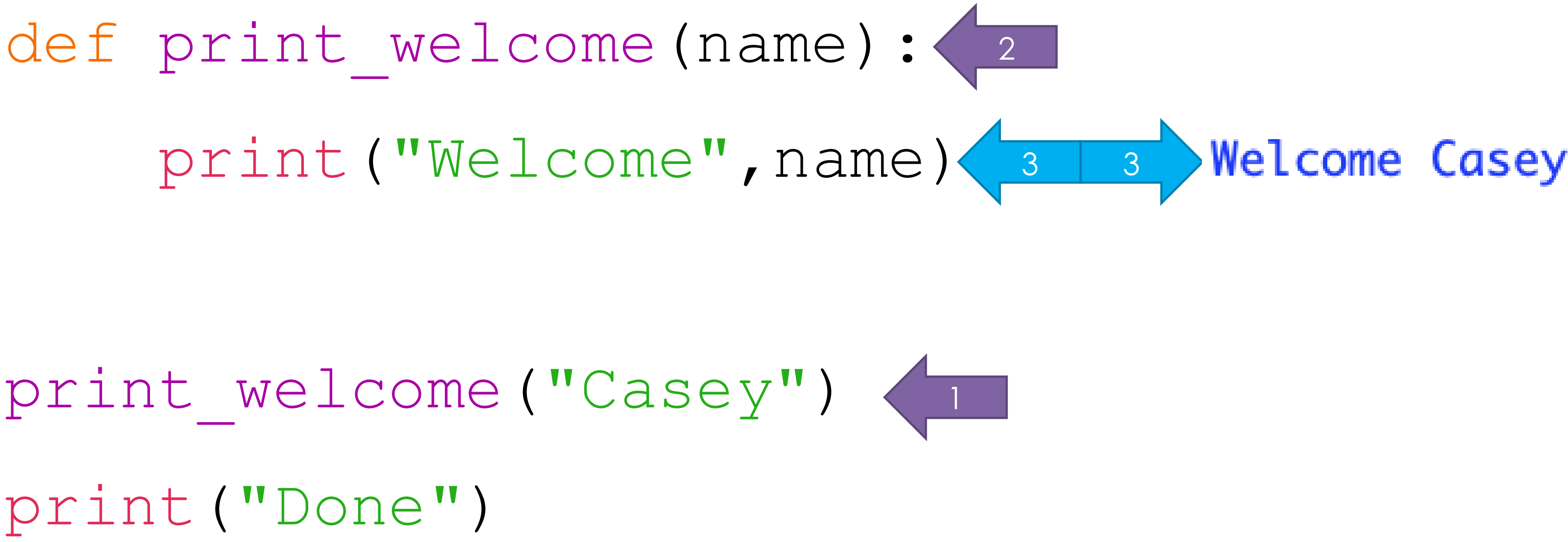
```
def print_welcome(name):
    print("Welcome",name)

print_welcome("Casey")
print("Done")
```

1

Functions Execution

```
def print_welcome(name):        2
    print("Welcome",name)


print_welcome("Casey")        1
print("Done")
```

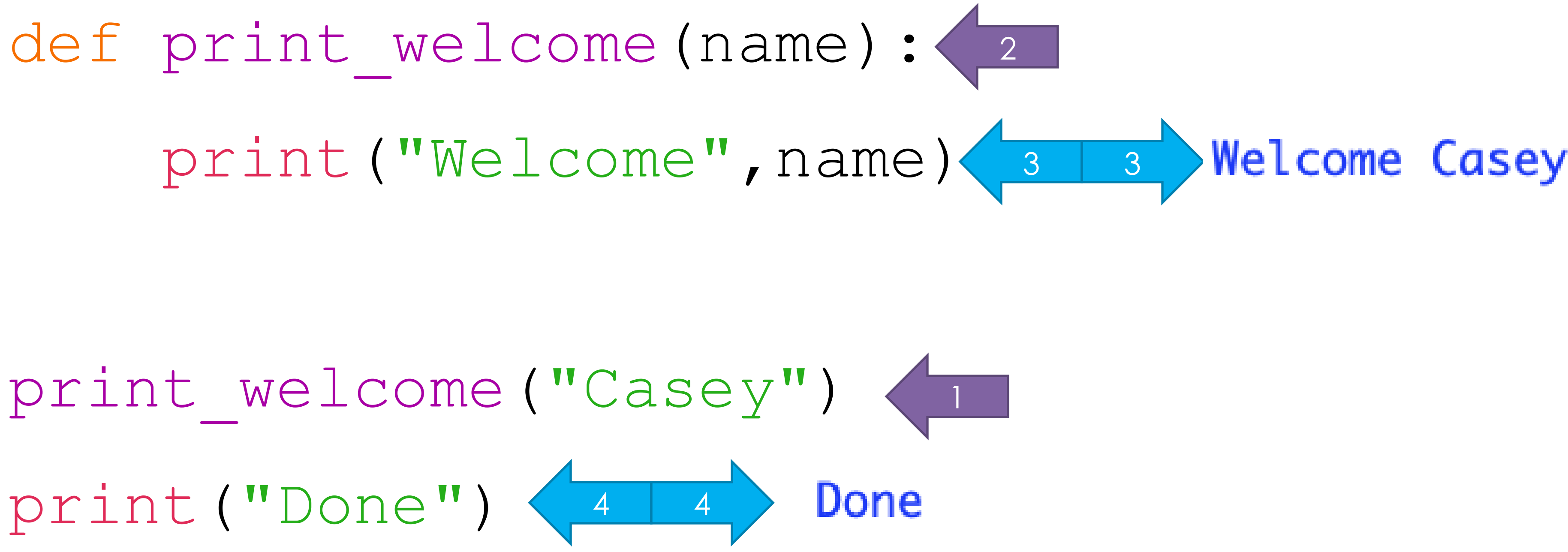
Functions Execution



```
def print_welcome(name):
    print("Welcome",name)


print_welcome("Casey")
print("Done")
```

```
Welcome Casey
```

Functions Execution



```
def print_welcome(name):
    print("Welcome",name)

print_welcome("Casey")
print("Done")
```

```
Welcome Casey
Done
```

Activity: Name and Age Function

- Create a program that asks the user for their name and age.
- Once the name and age has been input you need to pass it to a function that then prints out:
  - "[name] is [age] years old".

Discuss: How can we alter our code to use functions?

Activity: Functions

Edit your existing login/menu program to include functions.

Consider the following:

- Do you need still need the while loops?
- Can you use function calls instead?

Hints:

- Login can be a standalone function that calls the menu function
- The menu can be a standalone function also

Activity: Functions Solution

```
username = "casey"
password = "secret"

def login():
    inUsername = input("Please enter the username: ")
    inPassword = input("Please enter the password: ")
    if inUsername==username and inPassword==password:
        print("Welcome")
        menu()
    else:
        login()
```

Activity: Functions Solution

```
def menu():
    userInput = input("Please select an option: \n1.View \n2.Add
                      \n3.Delete \n")
    try:
        selection = int(userInput)
            …
        else:
            print("Invalid selection")
            menu()
    except ValueError:
        print("Please enter an integer to perform a selection")
        menu()
```

Activity: Functions Solution

```
login()
```

Don’t forget to start the program you’d have to call the login() function!

Data Structures

- The key role of a computer program is to store and process data.
- Any computer software has a **data model** that defines **what** data will be collected and worked on.
- The **data structure** defines **how** the flow of data is controlled in relation to inputs, processes and outputs.
- Data structures can have two main characteristics. Firstly they can be **static** or **dynamic**, and secondly they can be **mutable** or **immutable**.

Data Structures: Lists

Lists have many methods associated with them which we can utilise to manipulate/use the data stored within.

Lists

```
textList = ["a","b"]

numberList = [1,2,3]

mixedList = ["a",1,"b",2]
```

Lists - Accessing an Item

```
textList = ["a","b","c"]
print(textList[1])
```

```
b
```

```
textList = ["a","b","c"]
print(textList[0])
```

```
a
```

The numbers refer to the items index! Note how lists index's start at 0.

Lists - Getting The Length

```
textList = ["a","b","c"]
print(len(textList))
```

```
3
```

technoteach technocamps

Lists - Appending

```
textList = ["a","b","c"]

textList.append("d")

print(textList)
```

```
['a', 'b', 'c', 'd']
```

Append adds an item to the end of the list. To add an entry in a specific location use insert.

Lists – Editing an Item

```python
textList = ["a","b","c"]
textList[0] = "z"
print(textList)
```

```
['z', 'b', 'c']
```

Lists – Removing an Item

```
textList = ["a","b","c"]
textList.remove("b")
print(textList)
```

```
['a', 'c']
```

If the item you want to remove isn't in the list an error is thrown.
If two of the same values are in the list the first one is removed.

```
textList = ["a","b","c","b"]
textList.remove("b")
print(textList)
```

```
['a', 'c', 'b']
```

Activity: Lists

- Create a list that holds the names of 5 people.
- Print the list.
- Remove the first item in the list.
- Append another name to the list.
- Print the third element in the list.
- Change the third element to be "Tilly".
- Print the list.
- Print the length of the list.

Data Structures: Two Dimensional Lists

A list keeps track of multiple pieces of information in linear order, or a single dimension. However, the data associated with certain systems (a digital image, a board game, etc.) lives in two dimensions. To visualize this data, we need a multi-dimensional data structure, that is, a multi-dimensional list.

Data Structures: Two Dimensional Lists

A two-dimensional list is really nothing more than a list of lists (a three-dimensional list is a list of lists of lists). Think of your dinner. You could have a one-dimensional list of everything you eat:

(lettuce, tomatoes, salad dressing, steak, mashed potatoes, string beans, cake, ice cream, coffee)

Or you could have a two-dimensional list of three courses, each containing three things you eat:

(lettuce, tomatoes, salad dressing) and (steak, mashed potatoes, string beans) and (cake, ice cream, coffee)

Two Dimensional (2D) Lists

```
menu=[
     ["prawn cocktail", "spring rolls", "duck pancakes"],
     ["steak and chips", "chow mein", "chicken tikka masala"],
     ["ice cream", "chocolate brownie", "strawberry cheesecake"]
]

myList = [ [0,1,2,3,4],[3,6,1], [2,1] ]
```

2D Lists - Accessing an Item

```
print(menu[1])
```

```
['steak and chips', 'chow mein', 'chicken tikka masala']
```

```
print(menu[0][1])
```

```
spring rolls
```

2D Lists - Getting the Length

```
myList = [ [0,1,2,3,4],[3,6,1], [2,1] ]
print(len(myList))
3
```

```
print(len(myList[0]))
5
```

2D Lists - Appending

```
myList = [ [0,1,2,3,4],[3,6,1], [2,1] ]
print(myList)
```

```
[[0, 1, 2, 3, 4], [3, 6, 1], [2, 1]]
```

```
myList.append([1,6,7,8])
print(myList)
```

```
[[0, 1, 2, 3, 4], [3, 6, 1], [2, 1], [1, 6, 7, 8]]
```

```
myList[1].append(3)
```

```
[[0, 1, 2, 3, 4], [3, 6, 1, 3], [2, 1], [1, 6, 7, 8]]
```

2D Lists - Editing an Item

```
myList = [ [0,1,2,3,4],[3,6,1], [2,1] ]
print(myList)
```

```
[[0, 1, 2, 3, 4], [3, 6, 1], [2, 1]]
```

```
myList[0] = [1,2,3,4,5]
print(myList)
```

```
[[1, 2, 3, 4, 5], [3, 6, 1], [2, 1]]
```

```
myList[1][0] = 1
print(myList)
```

```
[[1, 2, 3, 4, 5], [1, 6, 1], [2, 1]]
```

2D Lists - Removing an Item

```
myList = [ [0,1,2,3,4],[3,6,1], [2,1] ]
```

```
myList[0].remove(1)
print(myList)
```

```
[[0, 2, 3, 4], [3, 6, 1], [2, 1]]
```

```
myList.remove([2,1])
print(myList)
```

```
[[0, 2, 3, 4], [3, 6, 1]]
```

Activity: 2D Lists

- Create a 2D list that holds 3 starters, 4 main meals and 5 desserts.
- Print the list.
- Remove the first item in the starters list.
- Append another dessert to the dessert list.
- Print the third element in the main meal list.
- Print all of the desserts.
- Print the length of the whole menu.
- Print the length of the starters.

Data Structures: Dictionaries

Python provides another composite data type called a dictionary, which is similar to a list in that it is a collection of objects.

Data Structures: Dictionaries

Dictionaries and lists share the following characteristics:

- Both are mutable.
- Both are dynamic. They can grow and shrink as needed.
- Both can be nested. A list can contain another list. A dictionary can contain another dictionary. A dictionary can also contain a list, and vice versa.

Data Structures: Dictionaries

Dictionaries differ from lists primarily in how elements are accessed:

- List elements are accessed by their position in the list, via indexing.
- Dictionary elements are accessed via keys.

Dictionaries

```
cars = {
     "Audi Q5":20000,
     "Volkswagen Polo":5000
     }
```

```
print(list(cars.keys()))
```

```
['Audi Q5', 'Volkswagen Polo']
```

```
print(cars)
```

```
{'Audi Q5': 20000, 'Volkswagen Polo': 5000}
```

Dictionaries - Accessing an Item

```
cars = {
    "Audi Q5":20000,
    "Volkswagen Polo":5000
    }


print(cars["Audi Q5"])
20000
```

Dictionary items are accessed by using their keys as opposed to indexes!

Dictionaries - Getting the Length

```
cars = {
     "Audi Q5":20000,
     "Volkswagen Polo":5000
     }

print(len(cars))
2
```

Dictionaries - Appending

```
cars = {
    "Audi Q5":20000,
    "Volkswagen Polo":5000
    }

cars["Ford KA"]=500

print(list(cars.keys()))
```

```
['Audi Q5', 'Volkswagen Polo', 'Ford KA']
```

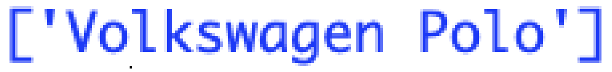
Dictionaries - Editing an Item

```
cars = {
    "Audi Q5":20000,
    "Volkswagen Polo":5000
    }

cars["Audi Q5"]=10000
print(cars["Audi Q5"])
```

```
10000
```

Dictionaries - Removing an Item

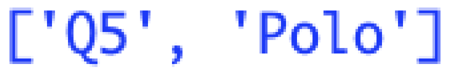
```
cars = {
     "Audi Q5":20000,
     "Volkswagen Polo":5000
     }

cars.pop("Audi Q5")
print(list(cars.keys()))
```



```
['Volkswagen Polo']
```

More Advanced Dictionaries

```
cars = {
    "Q5":{
        "Brand":"Audi",
        "Price":20000
        },
    "Polo":{
        "Brand":"Volkswagen",
        "Price":5000
        }
    }
print(list(cars.keys()))
```



```
['Q5', 'Polo']
```

Activity: Create a Dictionary/2D List

Edit your login/menu program and create a dictionary/2D list to hold the following information:

Key	Name	Price	Number In Stock
1	Dairy Milk Plain Chocolate	80	15
2	Lucozade Sport	180	10
3	Strawberry Laces	30	8

When a user selects view they should be able to see the contents of the dictionary/2D list.

Activity: Create a Dictionary Solution

```
username = "casey"
password = "secret"
shopItems = {
    1:{
        "Name":"Dairy Milk Plain Chocolate",
        "Price":80,
        "NumberInStock":15
        },
    2:{
        "Name":"Lucozade Sport",
        "Price":180,
        "NumberInStock":10
        },
```

Activity: Create a Dictionary Solution

```
    3:{
        "Name":"Strawberry Laces",
        "Price":30,
        "NumberInStock":8
        }
    }

def login():
    …
```

Activity: Create a Dictionary Solution

```
def menu():
    …
        if selection==1:
            print("You selected view")
            print(shopItems)
        …
    except ValueError:
        print("Please enter an integer to perform a selection")
        menu()

login()
```

Activity: Create a 2D List Solution

```
username = "casey"

password = "secret"

shopItems = [

    [1,"Dairy Milk Plain Chocolate",80,15],

    [2,"Lucozade Sport",180,10],

    [3,"Strawberry Laces",30,8],

]


def login():

    …
```

Activity: Create a 2D List Solution

```
def menu():
    …
        if selection==1:
            print("You selected view")
            for i in range(len(shopItems)):
                print(shopItems[i])
        …

login()
```

Markdown

Before we dive into Python, Colab gives us an opportunity to practice another language – Markdown!

This is a simple text formatting language and can be used to add notes to your Notebook.

Your starter packs have a Markdown guide to assist you!

MatPlotLib

There are countless libraries available for Python, many of which we will be using in various depths.

Today we will be focusing on Python's most popular data visualisation library, called MatPlotLib!

MatPlotLib was born out of a desire to recreate MATLAB functionalities within Python.

MatPlotLib

To get to grips with using data structures in Python and the basic plot functions of MatPlotLib, we're going to load in some very basic data.

Our first data set has been shamefully provided by one of our delivery officers, Dan.

Download Data

The first data set that we will use today can be downloaded here:

tc1.me/DansDiet

Once downloaded, these can be dragged directly into your project folder.

Loading

The most common way of loading data of any sort into Python is by opening the file directly **within your code**.

This is the **simplest** way of importing data, so we will only cover it briefly.

It is worth noting, that while it is easiest to place the file in your project folder, you can also open it from any other location on your device by specifying the file path.

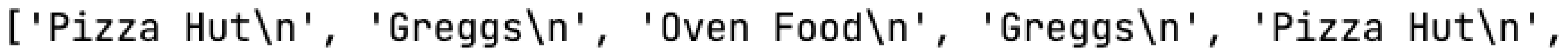
Load in File

To begin we will learn to load in a file:

```
dietSource = open(“DansDiet.txt”, “r”)
diet = []

for line in dietSource:
    diet.append(line)

print(diet)
```

Load in File

Notice how when printed the output contains ‘\n’ at the end of each line that was read into the program:



```
['Pizza Hut\n', 'Greggs\n', 'Oven Food\n', 'Greggs\n', 'Pizza Hut\n',
```

This is the newline character that tells the computer to progress to the next line!

If we want to remove this, we will have to specify it!

Load in File

Change your program to look like this:

```
dietSource = open(“DansDiet.txt”, “r”)
diet = []

for line in dietSource:
    diet.append(line.replace(“\n”, “”)

print(diet)
```

Notice how the ‘\n’ has been replaced with nothing.

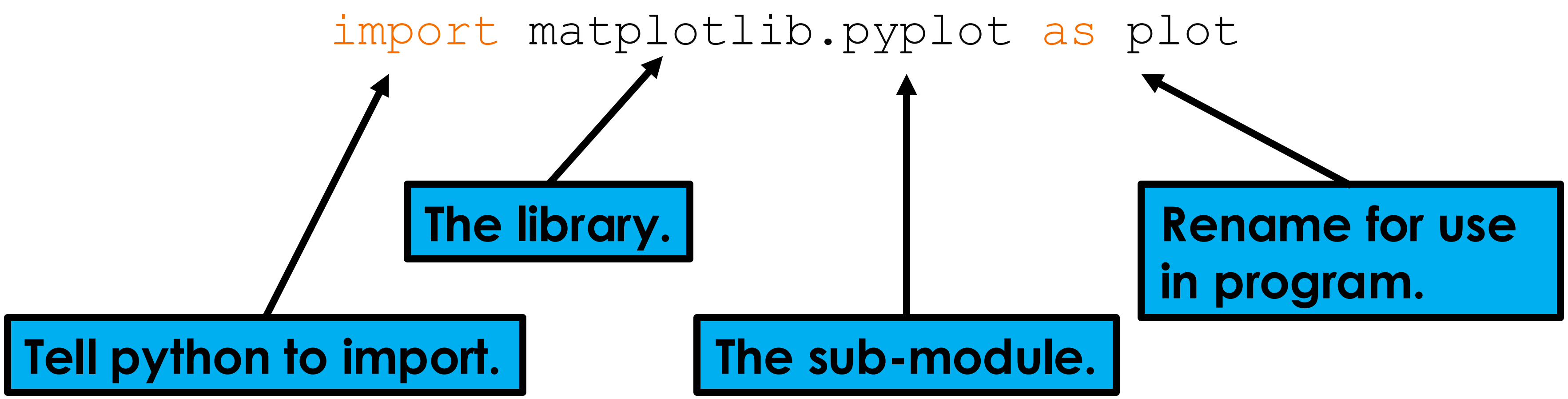
Begin Plotting

In order to plot anything we will need access to these functions!

We are going to be using the MatPlotLib library for Python, so we will need to import this for use in our program.

At the very top of our Python file, we need to add this line::

```python
import matplotlib.pyplot as plot
```



Begin Plotting

To plot the data, we'll need the following code:

```
plot.figure()

fig1 = plot.bar([x[0] for x in dietCounts],
                        [y[1] for y in dietCounts])

plot.show()
```

Create a window for the plot.

These are in line for loops.
The bar plot is being given two lists of values, one from column [0] of the data, the other from column [1].

Display the plot.

Adjust the Plot

Our data will automatically be plotted with a basic graph, however all elements can be adjusted before the plot is shown:

```
plot.xticks(rotation = 90)

plot.subplots_adjust(bottom = 0.25)

plot.show()
```

Rotate the x-axis labels so that they don't overlap.

Resize the bottom of the plot so that the rotated labels still fit.

The same show() as on the last slide.

Adjust the Plot

We can add titles to our chart and axes using these methods:

```
plot.xlabel(“Food Options”)
plot.ylabel(“Number of Visits”)
```

Add a label to each axis.

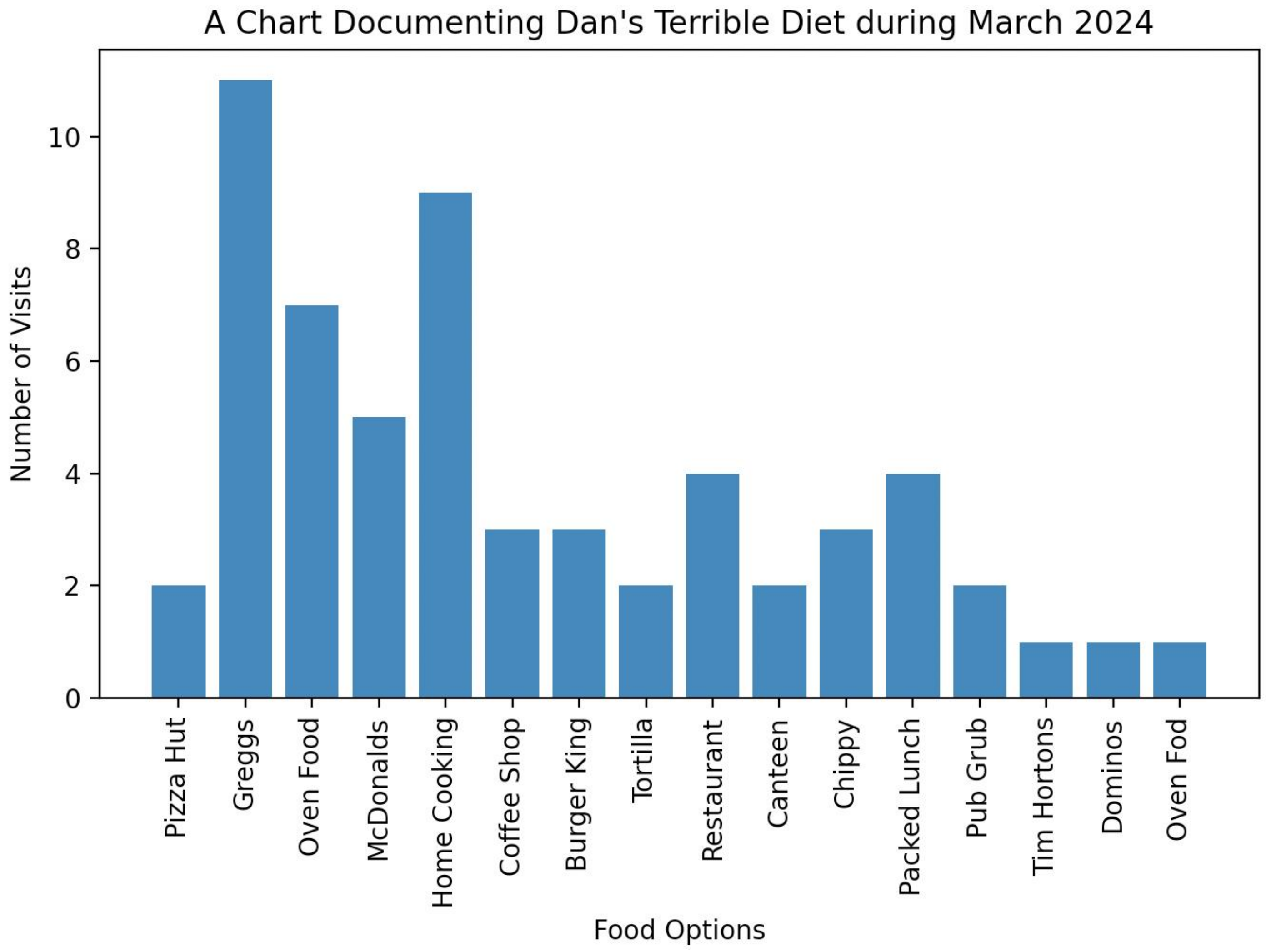
```
plot.title(“A Chart Documenting Dan’s Terrible
           Diet During March 2024”)
```

Add a title to the top of the chart.

```
plot.show()
```

The same show() as on the last slide.

The Plot



A Chart Documenting Dan's Terrible Diet during March 2024
Number of Visits
0
2
4
6
8
10
Pizza Hut
Greggs
Oven Food
McDonalds
Home Cooking
Coffee Shop
Burger King
Tortilla
Restaurant
Canteen
Chippy
Packed Lunch
Pub Grub
Tim Hortons
Dominos
Oven Fod
Food Options

Count Categories

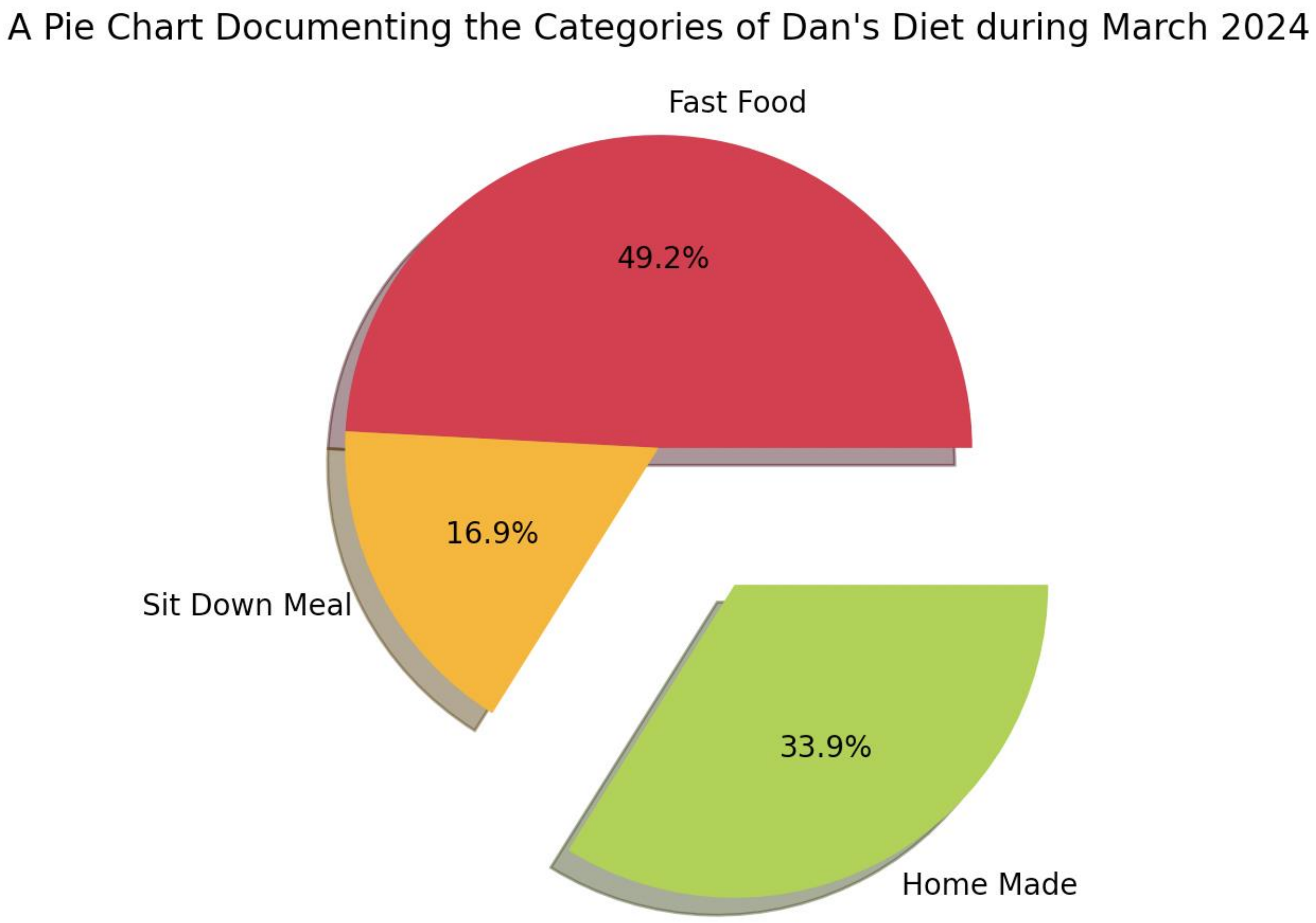
We can also manipulate our data to be in categories:

```
dietCategories = [[“Fast Food”, 0], [“Eating Out”, 0],
                   [[“Home Made”, 0]]
for item in dietCounts:
      for item in diet:
            if ((item[0] == “Greggs”) or
                  (item[0] == “McDonalds”) or …):
                        dietCategories[0][1] += item[1]
            elif ((item[0] == “Pizza Hut”) or
                  (item[0] == “Restaurant”) or …):
                        dietCategories[1][1] += item[1]
            elif ((item[0] == “Home Cooking”) or
                  (item[0] == “Oven Food”) or …): :
                        dietCategories[2][1] += item[1]
```

Count Categories

```
plot.figure()
plot.title(“A Pie Chart of Dan’s Diet Categories”)
explodeVal = (0, 0, 0.5)

fig2 = plot.pie(
      [counts[1] for counts in dietCategories],
      explode = explodeVal,
      shadow = True,
      labels = [labels[0] for labels in
                               dietCategories],
      autopct = ‘%1.1f%%’,
      colors = [‘crimson’, ‘orange’, ‘yellow’])

plot.show()
```

Count Categories



UFO Sightings

Now we will analyse a more complex data set – this time of UFO Sightings.

However, instead of importing the dataset as a file, we will import it directly with an API.

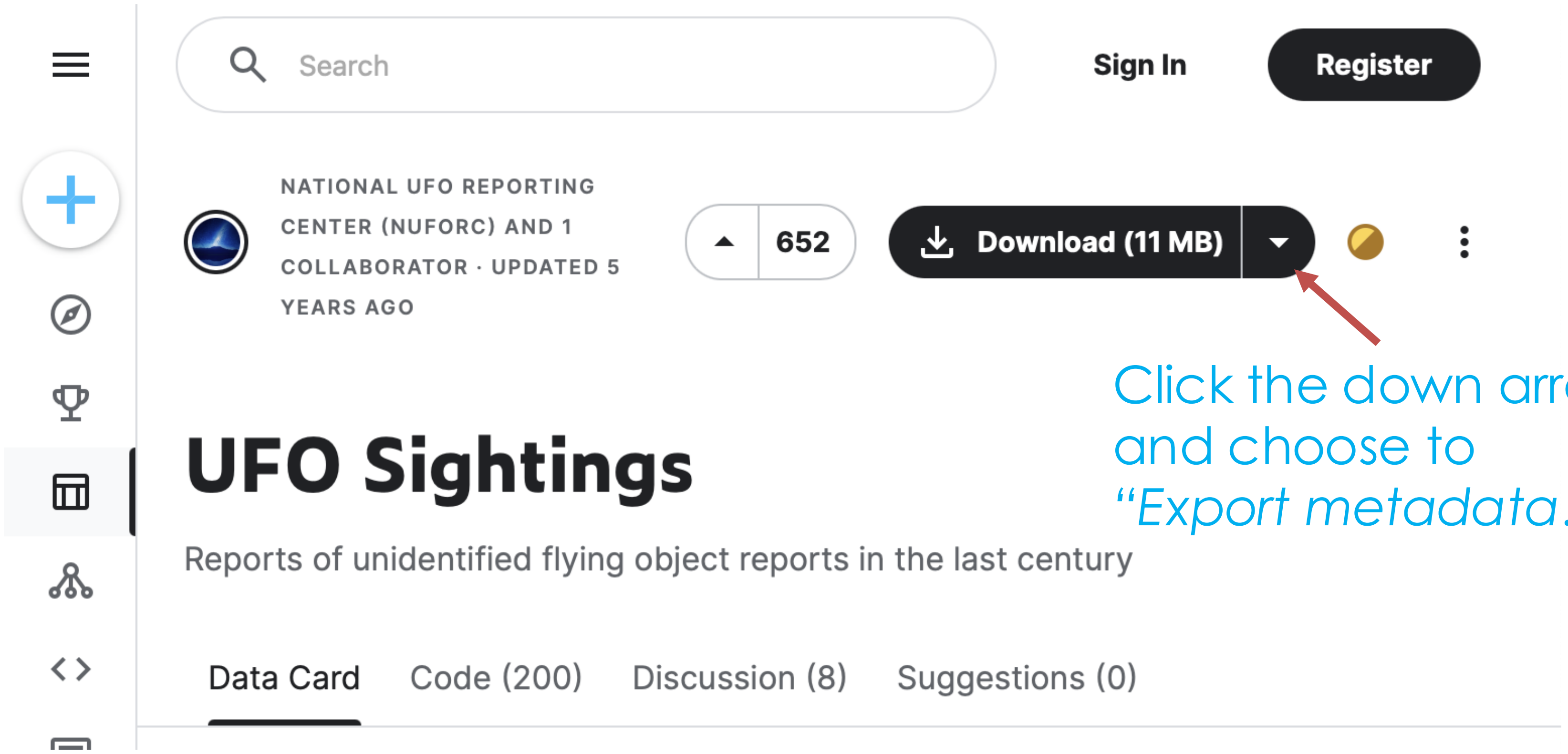
This is a little more convoluted for this example, however it will offer you the experience of using an API in Python.

Begin by heading to this link:

tc1.me/UFOData

Download Metadata

Unfortunately, this webpage does not offer us a direct download page. So, we will have to manually find the link:



NATIONAL UFO REPORTING CENTER (NUFORC) AND 1 COLLABORATOR · UPDATED 5 YEARS AGO

UFO Sightings

Reports of unidentified flying object reports in the last century

Click the down arrow and choose to *"Export metadata…"*

Metadata File

By opening the metadata file, we can find the **contentUrl**:

```
[{"contentUrl":"https://www.kaggle.com/api/v1/datasets/download/NUFORC/ufo-
sightings?datasetVersionNumber=2","contentSize":"10.216
MB","md5":"RaFESnmRn\u002B3NroE/S92GOw==","encodingFormat":"application/zip","
@id":"archive.zip","@type":"cr:FileObject","name":"archive.zip","description":
"Archive containing all the contents of the UFO Sightings dataset"},
{"contentUrl":"complete.csv","containedIn":
{"@id":"archive.zip"},"encodingFormat":"text/csv","@id":"complete.csv_fileobje
ct","@type":"cr:FileObject","name":"complete.csv"},
```

This also, conveniently, tells us that the content is a **zip** file.

This means that in order to retrieve the **csv**, we will have to extract the **zip** archive.

Import UFO Sightings

Firstly, we will request the file from the server via an API call from within our Python code, this requires an import:

```
import requests


response = requests.get(
'https://www.kaggle.com/api/v1/datasets/download/NUFO
RC/ufo-sightings?datasetVersionNumber=2')
```

Extract UFO Sightings

However, the response from the server is actually a **zip** file, for Python to use the data within we will have to extract the contents:

```
import zipfile as zip
import io


archive = zip.ZipFile(io.BytesIO(response.content),
'r')


theUFOFile = archive.open('complete.csv')
theUFORead = io.TextIOWrapper(theUFOFile)
```

Analyse Contents

Now that the file is loaded, we can begin to analyse its contents.

As we know it is a **csv** (comma-separated values) file, the data is likely comma-separated, but we can print to be sure:

```
for line in theUFORead:

      print(line)
```

Extract Data

Now that we know the data format, we can load the data as an array. This code will load each line (entry) of the data as a list, within a list of entries. It will also skip the first line (as it is just headers):

```
allUFOData = []

for i, line in enumerate(theUFORead):

      if i != 0:

            allUFOData.append(line.split(“,”))
```

UFO Sightings

We will try to plot the number of UFO reports that were received at each hour of the day, to see what times are more likely for a UFO to be spotted!

To plot this, we will have to sort the data by the timestamps, and find out how many reports were generated for each hour.

If we were to just plot the data, we would have a tick on the axis for **every timestamp**, not a tick for **each hour** of the day.

To plot this will therefore require us to manipulate the data, and extract the hour from each of the timestamps.

UFO Sightings – Sort by DateTime

To begin we will sort our data by the timestamp. This will require a new library. By first sorting the data by timestamp we can loop through the data easily:

```
from operator import itemgetter

allUFOData.sort(key = itemgetter(1))

timeCount = []
hourStr = “00”
count = 0
```

UFO Sightings – Extract Hours

Now that we have sorted the data by timestamp, we need to pull out each hour:

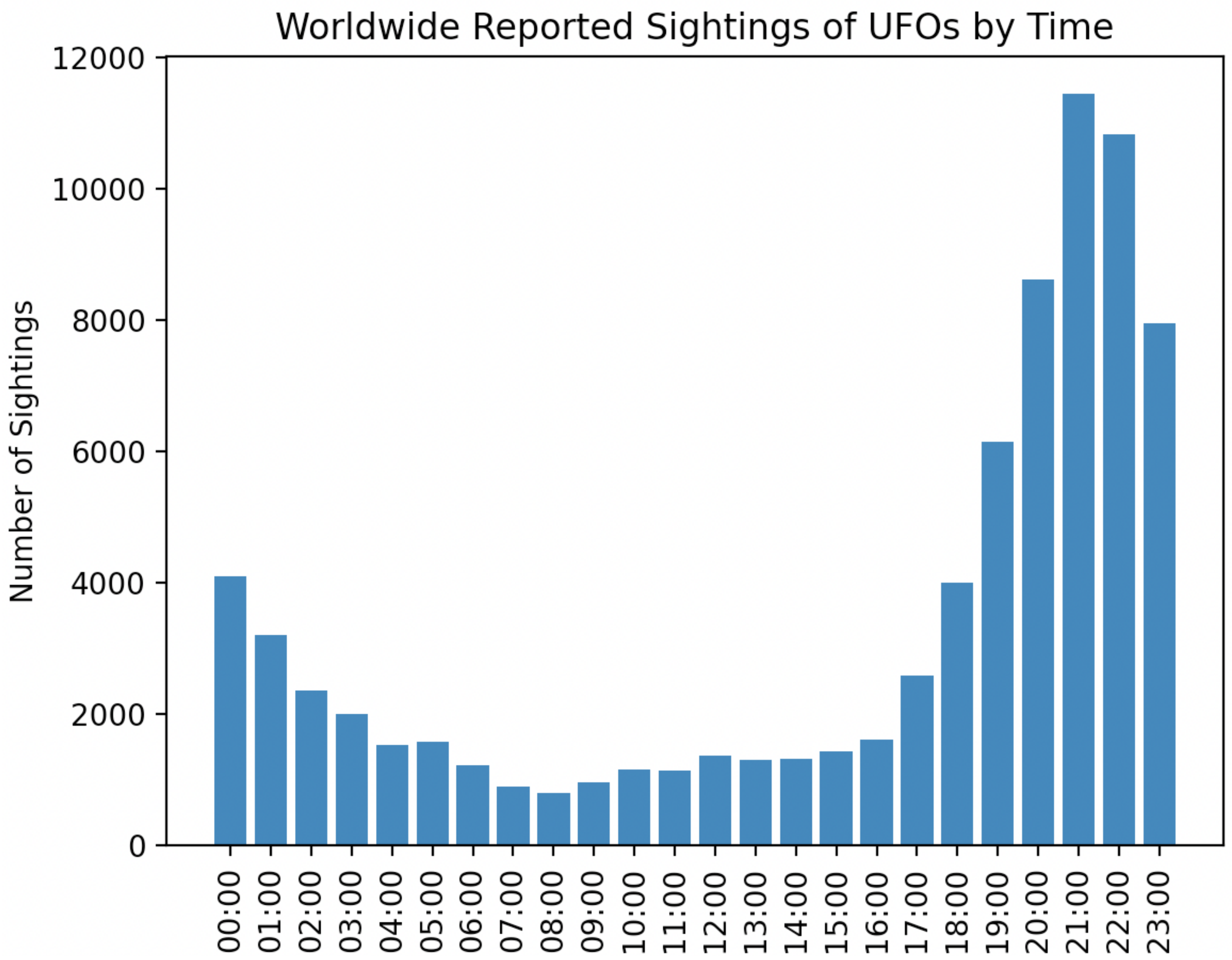
```
for entry in allUFOData:
      if entry[1][:2] == hourStr:
            count += 1
      else:
            timeStr = hourStr + “:00”
            timeCount.append([timeStr, count])
            hourStr = entry[1][:2]
            count = 1
```

UFO Sightings - Plot

With the counts for each hour extracted from our data, we can plot:

```
plot.figure()
fig3 = plot.bar([str[0] for str in timeCount],
                     [count[1] for count in timeCount])


plot.xticks(rotation = 90)
plot.xlabel(“Time”)
plot.ylabel(“Sightings”)
plot.title(“Reported Sightings by Date”)
```

UFO Sightings



UFO Sightings by Country

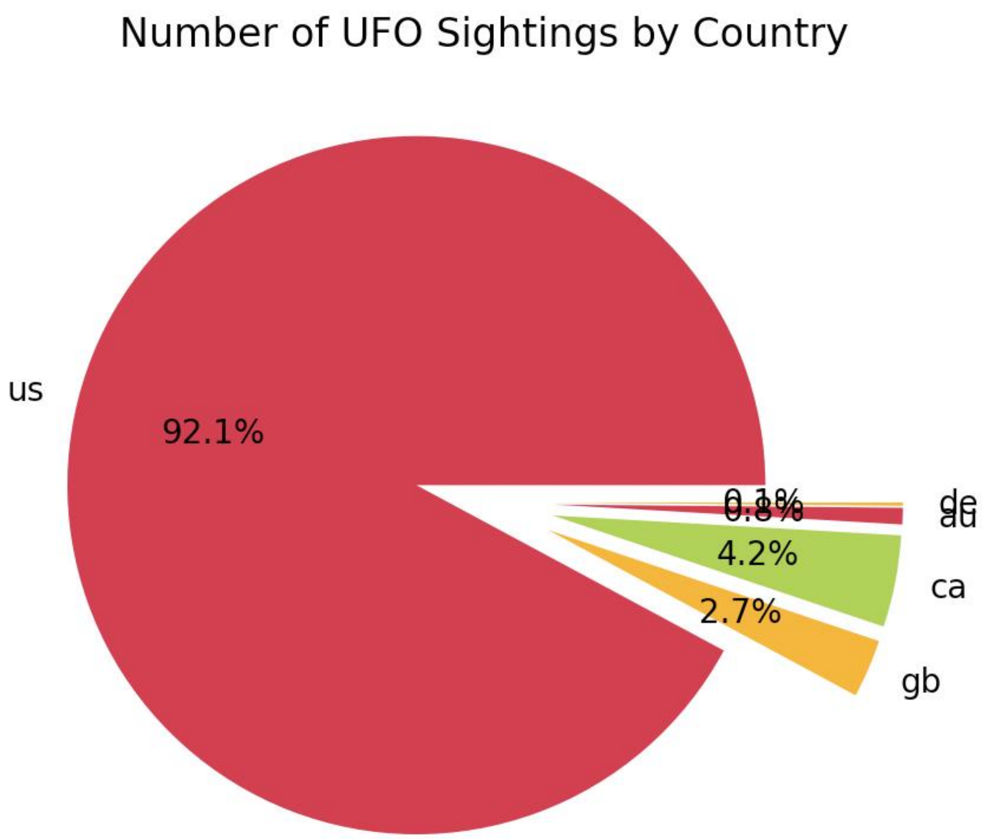
We can add the number of sightings per country with a for loop:

```
countryCounts = []
for entry in allUFOData:
      if entry[3] != “”:
            if entry[3] not in [country[0] for
                        country in countryCounts]:
                  countryCounts.append([entry[3], 1])
            else:
                  for country in countryCounts:
                        if country[0] == entry[3]:
                              country[1] += 1
```

UFO Sightings by Country

```
plot.figure()
plot.title(“Number of UFO Sightings by Country”)
explodeVal = (0.2, 0.2, 0.2, 0.2, 0.2)


fig4 = plot.pie(
      [counts[1] for counts in countryCounts]),
      explode = explodeVal,
      autopct = ‘%1.1%%’,
      colors = [‘crimson’, ‘orange’, ‘yellowgreen’],
      labels = [country[0] for country in
                      countryCounts])

plot.show()
```

UFO Sightings by Country



Duration vs. Time of Day

We can make a more complex scatter graph by taking two measurements – the **duration** and **time of day** for each sighting!

To begin we need to ensure both of these lines are present in our code:

```
from operator import itemgetter

allUFOData.sort(key = itemgetter(1))
```

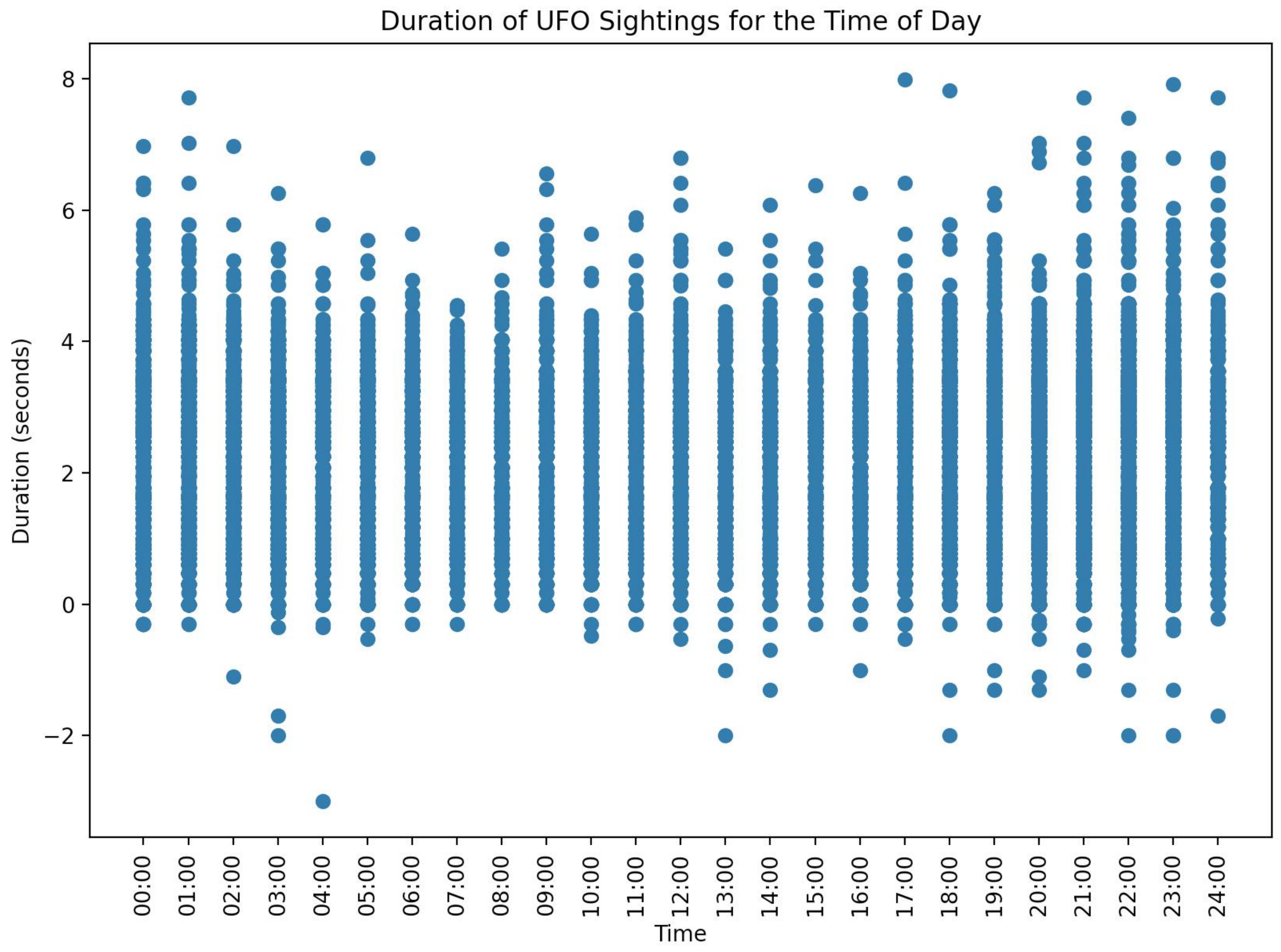
UFO Sightings – Extract Hours

```
timeDuration = []
hourStr = "00"
timeStr = hourStr + ":00"

for entry in allUFOData:
      if entry[1][:2] == hourStr:
            timeDuration.append([timeStr,
                  float(entry[5].replace("\n", ""))])
      else:
            hourStr = entry[1][:2]
            timeStr = hourStr + ":00"
            timeDuration.append([timeStr,
                  float(entry[5].replace("\n", ""))])
```

UFO Sightings – Extract Hours

```
with plot.style.context(‘tableau-colorblind10’):
      fig5 = plot.scatter([time[0] for time in
                                    timeDuration],
            [math.log10(duration[1]) for duration in
                                    timeDuration])
plot.xticks(rotation = 90)
plot.title(“Duration of Sighting – Time of Day”)
plot.xlabel(“Time”)
ploat.ylabel(“Duration (s)”)
```

UFO Sightings by Country



Lab Tasks