Lab 1 – Python Intro

**Part 1 – Create a python file and run it**

Using your editor of choice Create a file called python-demo.py and add the following:

print("Hello World")

You can then run your file via the run / execute command within your IDE or Editor or via the terminal with `python python-demo.py`

**Task** - Within your file print your top 3 favourite films on separate lines

**Part 2 – Using input fields and printing**

With python you can easily get user input and use it for other processes

name = input("Please enter name: ")

print(“Hello ” + name)

With this code it will save a variable called name as the text you enter and then print it adding “Hello “ in front of it.

**Task –** Write code that asks for user input for a coffee order, you should save variables for the following:  
- name (text)  
- drink type (text)  
- whipped cream (Boolean)  
- quantity (number)  
Return a print statement which includes all of the info (‘Name: John Smith, drink: Mocha..’)  
Stretch goal – Also take in up to 3 values for extras, save this as an array

Lab 2 – Selection Lab

Task 1 – Using If Else control flow statements

Within a new python file add the following code:

age = int(input("Please enter age: "))

if age >= 18:

    print("You are in Cat A")

Run the code with an age greater than 18 and see what the output is.   
Develop on this code piece so it prints “You are in Cat B” when the age is equal to 16 and “You are in Cat C” when it is below 16.

Stretch goal – If you did not use If Else statements for the above snippet, convert it to If Else statements.

Task 2 – Use If Else for more complex commands

**Part 1 - Calculator**

Create a program called Calculator.py that lets the user choose what command out of the following Add, Subtract, Multiply, Divide, Raise To Power they wish to do, to two numbers that are entered by the user.

Your file should first print out a menu like the below:

1. Add
2. Subtract
3. Multiply
4. Divide
5. Power

Using an input statement to choose a number between 1 – 5, and then asking the user to input two numbers to plug into the command.

The program should print the sum and result I.e if the user chooses to add and enters 4, 5 it should return 9.

**Part 2 – Exam Grades**

Create a program called ExamGrades.py that takes in a number between 1 and 100 and returns a **Fail** if less than 50, **Pass** if between 50 and 60, **Merit** if between 60 and 70 and **Distinction** if above.

Stretch 1 – Add a check to ensure the grade is above 0 and below 100, if it’s not return an error statement.

Stretch 2 – Ask the user to input a choice between Level 1 and Level 2. If Level 1 is chosen the grade boundaries are the same, if its Level 2 a **Fail** is less than 40, **Pass** is between 40 and 50, **Merit** is between 50 and 65, **Distinction** is above.

**Part 3 - Pythagoras**

Create a program called Pythagoras.py that can return the long angled side of a right angled triangle. Pythagoras’ Theorem states that the square of the long side (C) of a right-angled triangle is the sum of the squares of the two shorter sides (A and B).  
The user should be able to implement the lengths of sides A and B to return the length of C.

Stretch goal – You should use an input statement to allow the user to choose which side to calculate (A, B or C) and run the correct sum.

Lab 3 – Iteration Lab

Task 1 – Using While Loops

**Part 1 - Squares**

Add a new file called Squares.py and add the following to it:

number = 1

while number < 100:

    numberSqr = number \* number

    print(numberSqr)

number += 1

When this code runs it prints the square of each number from 1 -100.   
Add an if statement to end the loop if the square of a number is greater than 2000.

**Part 2 - Investment**

Create a file called Investment.py, this file should Calculates how many years it will take an initial investment of £100 to grow to a target value of £1000 if the interest rate is 10%. This program should use while loops to achieve the result.

Stretch goal – Make the program more usable by allowing the user to input values for: initial investment, target value and interest rate.

**Part 3 – Count Vowels**

Create a file called CountVowels.py that returns the number of vowels in a given word. The user should be able to input a word (as a string) and count the number of vowels (A,E,I,O,U) in said word.   
  
Stretch goal – The user can enter 3 additional letters into the program that it will then check for, and return the total number of found letters. I.e the user enters B, P, T and also enters BURRITO, it should return 5 (B, U, I, T, O).

Task 2 – Using For Loops

**Part 1 – Counting Down**

Add a new file called CountingDown.py and add the following:

for number in range(31, 0, -3):

    print(number)

This file will start from 31 and count down to 0, printing the result in each time.   
Develop this code so that it prints “This is less than 10” when the number is less than 10. The for loop should end after the first time “This is less than 10” has been printed.

**Part 2 – Integer Between two limits**

Add a file called GetInt.py that asks the user to input an integer between set minimum and maximum values (10, 40). The user has 3 chances to guess the correct number, if they guess correct it returns “Correct!”, if its wrong it says “Wrong.. guess again..” and they are given 3 guesses before it returns “Game Over.”

Stretch goal – The user is able to input different values for min and max values as well as the amount of guesses.   
Stretch goal 2 – The program tells the user if the guess is cold or hot depending on how close they are, up to you how you implement this.

**Part 3 – Factorials**

Add a file called Factorials.py that returns the factorial of an entered number. The factorial is the number gained when multiplying all integers between 1 and the chosen number. I.e Factorial of 4: 1 \* 2 \* 3 \* 4 = 24  
Use a For loop or While loop to achieve this

Lab 4 – Lists

Task 1 – Working with large List

In this task you will be working with a set of numbers below:

ages =[12,18,33,84,45,67,12,82,95,16,10,23,43,29,40,34,30,16,44,69,70,74,38,65,36,83,50,11,79,64,78,37,3,8,68,22,4,60,33,82,45,23,5,18,28,99,17,81,14,88,50,19,59,7,44,93,35,72,25,63,11,69,11,76,10,60,30,14,21,82,47,6,21,88,46,78,92,48,36,28,51]

1. Record the length of the list of variables and save this as a variable
2. Display all of the numbers in the list line by line using some form of loop
3. Looping through the list, increase the value of each age by 1
4. Create a new list which only contains the ages in the age range of 16 – 65, display the new list and confirm it only contains 16 – 65 year olds
5. Display the count of 16 – 25 year olds in the new list
6. Sort the ages of the new list (hint use <list>.sort())
7. What proportion of people belong in the 16 – 25 category within the new list

Task 2 – Counting Vowels with a list

We will be developing our count vowels program created in the previous set of labs but using a list instead.   
  
The user should be able to input a string and use this string (which is a list of chars) to count the number of vowels directly.   
Stretch goal – The user can enter 3 extra letters and check if they are in the word.

Lab 5 – Inbuilt Functions

Your task is using the Python inbuilt functions to read, manipulate and display students’ grades from a String.

The data to use is below:

**data="100,30,53,67,85,87,50,45,51,72,64,69,59,17,22,23,44,25,16,67,85,87,50,45,51,72,59,14,50,55,32,23,24,25,37,28,39,30,33,35,40,34,41,43,94,95,59,98,99,44,45,47,48,49,53,61,63,69,75,77,60,83"**

Create a new file called grades.py and go through the following steps:

1. Convert the string into a list of values, this can be done through split()
2. Display the minimum and maximum value of grades
3. If your code displayed 100 as minimum and 99 as maximum work out why it has done this   
     
   (hint – Are the values Strings or numbers?)
4. Display the average of grades to 2 decimal points
5. Import the statistics library using - `import statistics`
6. Use the statistics.mean() function to get the average grade to 2 decimal points
7. Display the median() value of this list
8. Use the string.format() to display the min, max, average, mean and median values

Lab 6 – Custom Functions

Task 1 – Basic Functions

Create a file called DiceRoller.py and add the following:

import random

def rollSix():

    return random.randint(1, 6)

rollSix()

When this function runs it returns a random number between 1 and 6. Create 3 more functions that roll an eight sided dice, 10 sided dice and 4 six sided dice. Run these functions separately and print their results.

You can enter a parameter into a functions’ () so that it passes the value in. This can be done to allow you to specify what size dice without needing to create multiple functions:

def rollDice(num):

    return random.randint(1, num)

rollDice(10)

Write a function which rolls 4 six sided dice, removes the smallest number from the pool (2, 4, 3, 5: remove 2 from the pool and add 4 + 3 + 5). Keep track of this number and generate 6 of these dice values.   
  
Stretch goal – Separate out the dice rolling function so that one function runs another so that each function only does one thing and is simplified.

Task – Create a custom function to work out tax

Create a program called incomeTax.py, this file will contain custom functions to work out the tax on an annual salary. The user will be able to input in a salary and it would return the taxable amount of the salary. This will be done in two parts with the simple version of tax which is total tax (simplified) and complex which is bracketed tax.   
  
Tax breakdowns are below:

No tax paid on £12,570 personal allowance.

£12,571 to £23,000 starter rate of 19%

£23,000 to £40,000 intermediate rate of 30%

£40,001 to £150,000 higher rate of 41%

Above £150,000 top rate of 46%

**Simple version**: Entire Salary will be taxed to the bracket it belongs in

**Complex version**: With the salary it is taxed correctly with it being bracketed, this should be done with a series of loops and conditional statements

Lab 7 – File IO

Task – Write and Read Data from a basic .txt file

Within your working directory create a python file called FileIO.py and a fruit.txt file. Within the fruit.txt file add a string of 5 fruit separated by commas “apples, pears, bananas”.   
Within the python file add the following

fruitFile = open("fruit.txt", "r")

data = fruitFile.readline()

print(data)

fruitFile.close()

When you run the code it will print the contents of the files.   
**Task** - Use inbuilt functions, lists and loops to print out each fruit line by line.

The following code will append a new fruit to the file:

fruitFile = open("fruit.txt", "a")

fruitFile.write(“kiwi”)

fruitFile.close()

Check the file to see the fruit.txt has the new fruit added.

**Task** – Use a loop to enter 1 – 10 on a new file called numbers.txt

Task – Using a provided file, pull the data from it and manipulate

You will be provided a .csv file that contains Car Data showing sales of different companies across different months / years.

From this .csv you should extract the data, convert it to a series of lists and work out the following:

1. Sum of cars sold by Ford in total
2. Sum of card sold in May 2019
3. Average of cars sold in Aug 2019
4. Car manufacturer who sold most cars Jan 2019 – Apr 2019
5. Month where the least amount of cars were sold

Lab 8 – PythonDB

Task - Using a pre configured SQL DB and getting / creating data

Using the GoToMyPC installation open the **Microsoft SQL Server Management Studio** and login with the default credentials and account.

Click on the ‘**New Query**’ button in the upper task bar to write SQL Statements.   
At the top of the task bar in the drop down that says **Master** change this to ‘**QAStore**’ and add the following to the query:

SELECT \* FROM INFORMATION\_SCHEMA.TABLES;

Execute this query by clicking the ‘Execute’ button at the top of the screen to output all of the tables in the results below.

Now you know the tables available you can view the contents of the specific tables with SELECT \* FROM <table name>;

Tasks:

1. View all results from the contact table
2. View all results from sale where company\_no is 3000
3. View all results from sale where order\_value where is grater than 5
4. Add a new company to the company table using INSERT INTO <table name>
5. View your company in the table and update all counties to ‘Gwent’
6. Delete the new record you created
7. Create a new table in this database called items. This table should contain: item\_id, name, price, description. Add 3 items to this table and view the items

Task – Using Python and SQLite to work with a local DB

Create a file called zoo.py and add the following to it:

import sqlite3 as sqlite

conn = sqlite.connect("test-db")

cursor = conn.cursor()

Create an SQL file called your animal type of choice (fish.sql) and inside of this .sql file enter SQL command to create a table with at least 4 fields using a primary key. This .SQL file should also enter at least 3 different entries into the table.

sql\_file = open("<file name.sql>")

sql\_string = sql\_file.read()

cursor.executescript(sql\_string)

print(cursor.execute("SELECT name FROM sqlite\_master WHERE type='table';").fetchall()

The first 3 commands load the .sql file into the SQLite database, the final command is printing all of the tables in the database.   
This final function (cursor.execute(“SQL String”)) can be used to run SQL statement inside of the database and is the foundation of the next few steps.

Create functions that do the following:

1. Adds a pre-set record to the table when the function runs (will need to modify the id)
2. Deletes a record by ID
3. Views all records in the table
4. Views a record by ID
5. Updates records by ID, changes one field to an inputted field
6. Uses input fields to take in data and allow the user to enter a new record

Stretch goal, modify the .sql file so that it contains two tables with a primary and foreign key restraint.

Lab 9 – Classes

Task – Creating classes and constructors within Python

Create a file of your choice of animal.py (Bird.py e.g) and add a constructor to it with the format below:

class animal:

    def \_\_init\_\_(self, trait1, trait2, trait3):

        self.trait1 = trait1

        self.trait2 = trait2

        self.trait3 = trait3

newAnimal = animal("Blue", 123, True)

Your constructor should also contain a function that is true to the animal (bird() -> fly()) that does a simple print statement of this function.

Create 2x objects of this constructor with different values.

Use the getAttr(), hasAttr(), setAttr() and delAttr() functions with the objects attributes to manipulate the values.

Task – Updating old files into Classes

Look through your previous exercises and convert one into a class based object. A good example of this would be Calculator.py, when you run the commands you will have to specify the Calculator object like below:

calcObj = calculator(“V2.45”)

calcObj.addSum(5, 4)

Lab 10 – OOP

Task – Using Inheritance as a tool

**Part 1 – Basic Inheritance**

Create a Parent – Child constructor system using two types of animals. An example of this is **Penguin** is a child of **Bird.**The parent class should contain 3+ fields and 1 function.   
The child class should be formatted like below where it inherits the values:

class child(Parent):

    def \_\_init\_\_(self, trait1, trait2):

        super().\_\_init\_\_(trait3)

        self.trait3 = trait3

Create the child class where you are updating the function inside the parent class AND adding a new function unique to the child class.

**Part 2 – Purpose of Inheritance**

Create a 2nd child class that inherits the values and functions from the parent class to allow convergent objects. The two **child** classes should have **unique** traits, example would be Penguin and Eagle are both types of birds but one can fly and the other can’t.

Task – Using Abstraction

Creating an abstract class

Use the following to create abstract classes:

from abc import ABC, abstractmethod

Create an abstract class of a top level class for your parent – child (example would be animal).   
This abstract class should contain an abstract method that doesn’t return anything.

Modify one of your child classes to use the new Abstract class as a parent and overloads the abstract method.

Task – Using Encapsulation

**Task – ‘Private’ Variables**

Create a user class with name, age, fav\_colour and a private variable for password.   
Private variables are variables with a leading \_underscore.   
Check what are the differences between printing public variables of objects and private variables.

**Task – Encapsulation using Getters and Setters**

**Getters and Setters** are used to Retrieve data and Modify data within an object. Within your user class create containing functions that Get and Set all of the variables that are contained.   
  
**Stretch goal** – Add restriction and validation to the getters and setters, a simple example would be to check the \_password is valid when retrieving or setting data or ensuring the data type is correct when setting data.

Lab 11 – Testing

Task – Simple Testing and using pytest

Using pip to install pytest `pip install pytest` from within the terminal so the app has access to our testing suite.

Add the following code to the file test-demo.py

def func(num):

    return num \* 2

def test\_answer():

    assert func(6) == 10

You can run this test with `pytest test-demo.py` and you’ll see the test fails.   
**Task** – Fix the test so it passes

Create a file called codebase.py and copy the python functions from this repo <shorturl.at/akL01> into the file. Create a separate file test\_codebase.py and write tests for the codebase.py importing the module in and writing tests for each function.   
  
Stretch Goal – Write tests for previous functions and codebases you have worked on during the course.

Lab 12 – Linting

Use Pylint to fix the code in this repo <https://github.com/Reece-elder/QAA-M2-BadCode>

Pylint can be installed with `pip install pylint` and can be ran with `pylint filename.py`.

In order to get the code down you can git clone the repo, fork the repo and clone it down or just copy the .py file into a new python file.

Stretch goal – Use Pylint to lint previous code bases to make them cleaner and have a better build

Lab 13 – Flask

Task – Creating a basic Flask app

Use pip to install flask with `pip install flask` which is a python framework for creating web apps.

Create a file called app.py that contains the following:

from flask import Flask

app = Flask(\_\_name\_\_)

@app.route("/")

def hello\_world():

    return "Hello World!"

if \_\_name\_\_ == "\_\_main\_\_":

    app.run(debug=True, host="0.0.0.0", port=5000)

This file can be ran using `python app.py` and when ran will open a Web Server that can be accessed at localhost:5000 in your web browser.  
   
When this URL is accessed it will return “Hello World” to the browser.

**Task –** With your Flask app add routes that do the following:

* When the user enters /hello it returns “Hello <name>”
* /random it returns a random number between 0 and 10
* /item/<id> that returns a page displaying the id as a variable