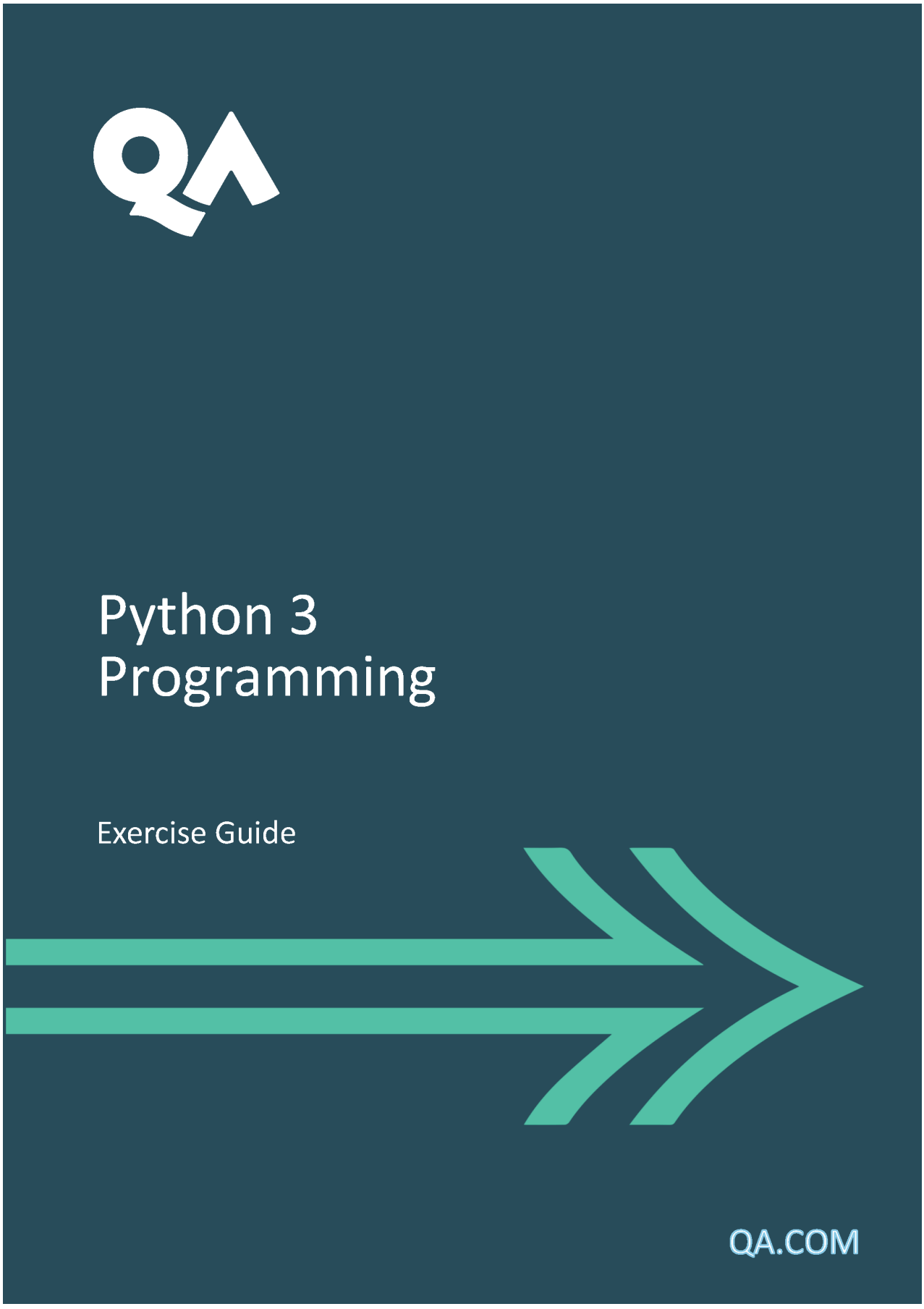
Python 3
Programming
Exercise Guide
QA.COM

Environment Set-Up

Your instructor will direct you to the courseware folder for this course which typically will have the path Courseware\QAPYTH3.

Jupyter Notebooks

1. Goto the Courseware folder and

- Open a console window (command prompt):
- Type :

```
jupyter labs
```

2. That should start the jupyter labs environment and launch your browser
3. You should be able to navigate to your individual notebooks and execute notebooks cells using CTRL ENTER
4. Your instructor will demo how to do this

Pycharm

This is a free to download dev environment for Python by JetBrains. We are using the Community Edition

1. A link to PyCharm will be on the Desktop of your machine and should be on the taskbar as well.
2. Open Pycharm, accept the defaults.
3. Your instructor should walk you through this process

Python IDLE

A link to the Python IDLE Console and Editor should be on your desktop.

Your instructor will walk you through using IDLE.

Exercise 2 - Variables and Type

Objectives

- To understand how to create variables
- To be able to assign value to and retrieve data from variables
- To be able to convert between data types

Activity

1. From the running Jupyter Lab notebook in your browser open notebook

a. 02_Fundamental_Variables\Variables and Data Types.ipynb

2. Follow the instructions for Quicklabs 1 to 3 as directed by your instructor

a. A solution can be found in the solution folder

3. Now take a look at the challenges if you have time. Try as many as you feel comfortable attempting.

Challenges

Challenge 1 - If you would like a further challenge

If you haven't done this already, use Pycharm to open the Labs folder. Ask your instructor for help with this.

The previous Jupyter Notebook asked you to calculate simple interest now we will get you to write a script to calculate compound interest.

1. Open challenge1.py and look at the script
2. All of the data necessary to carry out the following calculation is supplied in some form
3. Using the simple interest equation:

$A = P(1 + \frac{r}{n})^{nt}$

A = final amount

P = initial principal balance

r = interest rate

n = number of times interest applied per time period

t = number of time periods elapsed

4. Calculate the final amount using this formula
5. Outputs some useful results

Challenge 2

Write a program to calculate the gravitational force between the Earth and the Sun using the `gravitational_force` variable.

The formula for gravitational force is as follows:

Are you listen to me and kind of guy do you know what my fucking head but

$F = \frac{GMm}{r^2}$

F is the total gravitation force.

G is the gravitational constant.

M and m are the masses to be compared,

and r is the distance between those masses.

The values you will need to calculate the gravitation force of the Earth and Sun are:

G = 6.67 x 10-11

MSun = 2.0 x 1030

mEarth = 6.0 x 1024

r = 1.5 x 1011

Challenge 3

The height of a projectile (*y*) from a gun (ignoring air resistance) is given as:

$$y = y0 + xtan\theta - \frac{gx2}{2(v0cos\theta)^2}$$

where:

g : Acceleration due to gravity: 9.81 m/s squared

v0 : the initial velocity m/s

θ : (theta) elevation angle in radians

x : the horizontal distance travelled

y0 : height of the barrel (m)

1. Write a Python program to answer the following question:

- At a barrel height of 1m, after a horizontal distance of 0.5m, an elevation of 80 degrees, and an initial velocity of 44 m/s, what is the height of the projectile?

2. To convert degrees (deg) to radians use:

$$theta = deg * \frac{pi}{180}$$

- You will need to import some math methods:

```
from math import pi, tan, cos
```

There will be a further *if time allows* question which expands on this code after the Collections chapter.

Challenge 4

- Create a new program called **F1.py**, it will explore some of the mathematics involved in managing a Formula 1 racing car.
- The task of this program (at first), is to answer a question:
    - "During a race of **45** laps, what is the minimum fuel requirement?"

You will need to know the fuel consumption found during the race qualifying, which is **2.25** kg for each lap.

1. In this exercise, we will make a few more modifications to F1.py. First, we will add an extra fuel load, and then we are going to calculate the lap time based on the weight of fuel, which naturally decreases each lap.
2. In the previous exercise, we worked out the minimum fuel requirement for a 45 lap race and stored this in a variable named fuel_requirement. To fill the tank with the absolute minimum amount of fuel would be foolhardy, and not allow the drivers any margin for manoeuvre. Typically, a car will carry an extra 50% for contingency (multiply the minimum by 1.5). So what fuel will be carried by our fictional F1 car at the start of the race?
3. Modify your F1.py program to calculate this.
4. You might think it odd that fuel is measured in kilograms rather than litres or gallons. This is because the weight of fuel is critical to the way a Formula One car performs.

The qualifying lap time was 80.45 seconds, but that was with only 5kg of fuel: **each 10 kg of fuel increases the lap time by 0.35 seconds**.

- What will be the lap time for the first lap with all the required fuel on board?

Exercise 3 – Control Flow

Objectives

- To Understand how to control the flow of execution
- To use while loops
- To use for loops

Activity

1. From the running Jupyter notebook in your browser open notebook Controlling_Execution_Flow.ipynb
2. Follow the instructions for the Quicklabs as directed by your instructor
3. Now take a look at the challenges if you have time. Try as many as you feel comfortable attempting.

Challenges

Challenge 1

1. Write an income tax calculation engine using the following thresholds (or look the latest ones up for your location (remember Scotalnd and Wales may vary!)

Band	Taxable income	Tax rate
Personal Allowance	Up to £12,500	0%
Basic rate	£12,501 to £50,000	20%
Higher rate	£50,001 to £150,000	40%
Additional rate	over £150,000	45%

2. You may need to take into account of:

- Taxable Employment income
- Savings and Dividend income net of allowances

- Pension income
- Income from property
- Self employed income

Challenge 2

- We have provided you with challenge2.py
- Inside it is a function called generate numbers. (We will cover functions later on in the course). It returns an iterable sequence of numbers. Each number is yielded one, by one back to the caller of the function. It is a generator function (covered later).
- Your challenge(s) should you choose to accept them are to:

Based on the way the function **generate_numbers** works (ie it yields its value one by one from inside a for loop)

1. Complete the **primes** function so that it yields a sequence of prime numbers limited to the count provided by the count parameter. Lookup how to determine if a number is prime.
2. Complete the **fibonacci** to yield a sequence of Fibonacci numbers limited to the count provided by the count parameter.

- A fibonacci number is the sum of it's two previous numbers in the sequence.)
- Sequence begins with 0,1,1 – Third value 1 results from sum of first, 0 + second, 1. 0 + 1 = 1. Next value will be 1+1 = 2 etc

Test your functions using for loops and print the results.

Exercise 4 – Strings

Objectives

· To understand how to work with and process strings

Activity

1. From the running Jupyter notebook in your browser open notebook strings.ipynb
2. Follow the instructions for the Quicklabs as directed by your instructor
3. Now take a look at the challenges if you have time. Try as many as you feel comfortable attempting.

Challenges

Challenge 1

- In this challenge we will write a simple Caesar Cipher. It simple takes a string and adds one to the character code of each character in the message creating a new message.

1. Follow the comment instructions in challenge1.py
2. But that's pretty easy to break so if you want an extra challenge you could try writing a Vignère Cipher. To do this. for example:

- Offset the 1st character by 25
- Offset the 2nd character by 14
- Offset the 3rd character by 17
- Offset the 4th character by 10
- Repeat across the sequence of characters so the 5th character would be offset by 25 etc

Usualy the offsets are determined using a cipher key provider by the person encrypting the message.

3. Write some Python code to do this.

When you have tried out basic encryption why not research it a bit further? You will discover in the real world we don't usually rely on these basic algorithms but they are fun to write!

Challenge 2

1. Open the challenge2.py module.
2. Notice there some code which uses the bitcoin_prices module to load some price data and prints it out in a loop
3. It returns comma seperated data with the following price data

- close,high,low,open,timestamp,volume

4. We want you to use your knowledge of sting handling to extract the data from the commas and perform the following calculations

- Total Volume
- Average close price
- Maximum High
- Minimum Low

5. If you really want to challenge yourself try creating a Simple 20 day Moving Average for the Close price.

- Start with the first price and work forwards
- There won't be an SMA for the first 19 days!

6. Print the Date, Close Price, SMA

Exercise 4 - Collections

Objectives

To understand how to create and manipulate lists and work with dictionaries

Activity

1. From the running Jupyter labs in your browser open notebook 05_Collection\Collections.ipynb
2. Read the tutorials and follow the instructions for the Quicklabs

Challenges

Challenge 1

1. Open morse.py and take a look at it

- It contains a codes dictionary containing alph letters and their morse code equivalents
- Use the dictionary to translate the message string into morse code
- You could use a time.sleep() call to delay displaying each morse code letter
- Remember spaces and punctuation aren't included!

Challenge 2

1. Open quiz.py and follow the ToDo comment
2. You are asked to create a quiz from the dictionary returned from loading a JSON quiz file
3. Use input and some for loops to create an interactive quiz

Challenge 3

1. Write a program that generates a sequence of unique, sorted random numbers.

- You can using the **random** module and the **randint** function.

Requirement	Criteria
Generate a list of random numbers	Should be six numbers in list but this may change · Numbers in list should be unique · Range of numbers between 1 and 55 (inclusive)
Display the list of random numbers	Should be arranged as a vertical list
Count how many iterations it takes to generate a second list of random numbers identical to the first	All members must be identical
Display the number of iterations attempted	How many matches have been attempted?