

# Python 3

# Programming

for GCSE Computing

\*Updated for 2020 OCR Specification

2nd Edition, 2020

Ken McCall

Ken's Python Book is licensed under CC BY-NC-SA 4.0. To view a copy of this license, visit: <https://creativecommons.org/licenses/by-nc-sa/4.0>

# Introduction

---

---

This book is designed to aid GCSE Computing students in building up their programming skills in Python 3. All the necessary parts of the language are covered, with relevant examples to show correct use whenever possible.

The book is designed to be worked through from start to finish if you are new to Python, or as a quick reference guide for the more experienced. Sections can be taught on a lesson by lesson basis, or worked at individually. Each section has basic and extension tasks so students can build up a library of code as they go, and teachers can easily assess progress.

This book does not necessarily show the most efficient ways of solving all problems, merely the methods I have found to be the simplest to teach and learn. It does not cover the design and testing of larger programs that students will be required to create and so you should factor this into the time you have available.

Electronic files to support tracking and assessing students, along with all solutions are available from: [pythonbook@audiobluez.co.uk](mailto:pythonbook@audiobluez.co.uk)

2nd Edition, 2020  
Ken McCall  
First published 2013

Ken's Python Book is licensed under CC BY-NC-SA 4.0. To view a copy of this license, visit: <https://creativecommons.org/licenses/by-nc-sa/4.0>

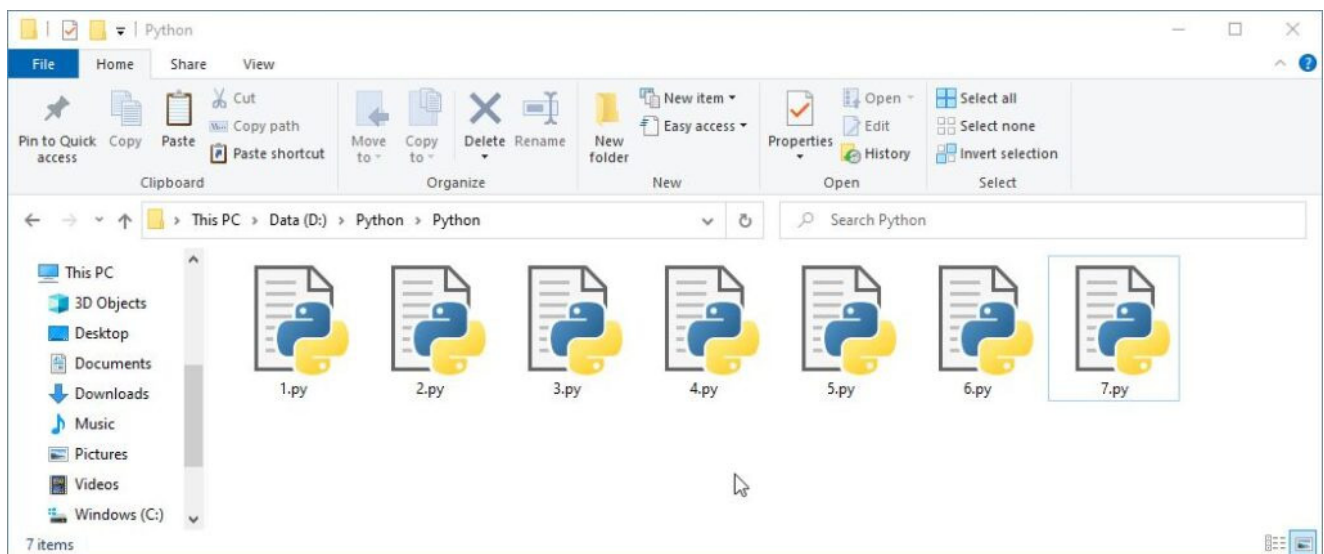
Copies of this book, and a web based version are available at:

[www.kenspythonbook.co.uk](http://www.kenspythonbook.co.uk)

# Housekeeping

## Create a folder to store your work

- ▶ Always create a folder with an obvious name like Python to save all your work in. Naming your Python files with the task number in the filename means by the end of this book you'll have a decent collection of code snippets you can refer back to later when working on projects.

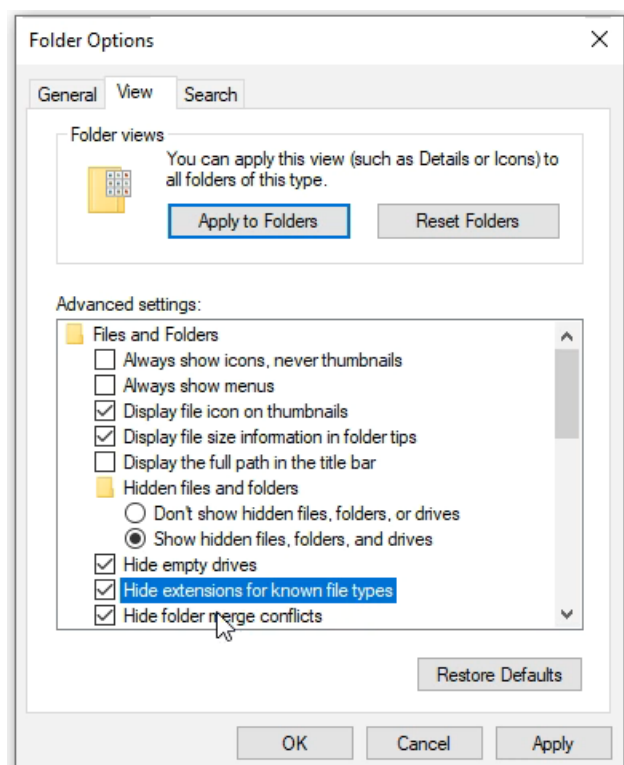


## Show File Extensions in Windows Explorer

- ▶ Especially when you get to working with text files, you'll need control over the extensions of the files you are creating.
- ▶ Make sure you can see the full filenames in Windows Explorer by following the short tutorial at the website address below:

[kenspythonbook.co.uk/housekeeping](http://kenspythonbook.co.uk/housekeeping)

- ▶ Uncheck the Hide Extensions option for full control over your filenames



# Contents

---

▶ 1. Accuracy	7
▶ 2. Output	8
▶ 3. Variables Part 1	9
▶ 4. Variables Part 2	10
▶ 5. Input	11
▶ 6. Test 1 - Ask And Tell	12
▶ 7. Variable Conversion Part 1	13
▶ 8. Variable Conversion Pt 2	14
▶ 9. Numbers	15
▶ 10. Working With Numbers	16
▶ 11. Test 2 - Volume And Area	17
▶ 12. Comments And Blank Lines	18
▶ 13. Decimal Places	19
▶ 14. Random Numbers	20
▶ 15. Selection With IF	21
▶ 16. IF...ELSE	22
▶ 17. IF..ELIF..ELSE	23
▶ 18. Checking Numbers	24
▶ 19. Checking Data Types	25
▶ 20. Logical Operators	26
▶ 21. Test 3 - Calculator	27
▶ 22. Checking Strings	28
▶ 23. Operators And Strings	29
▶ 24. Other String Things Part 1	30
▶ 25. Other String Things Part 2	31
▶ 26. Count Controlled Iteration	32
▶ 27. Condition Controlled Iteration	33
▶ 28. Test 4 - Iteration	34

# Contents

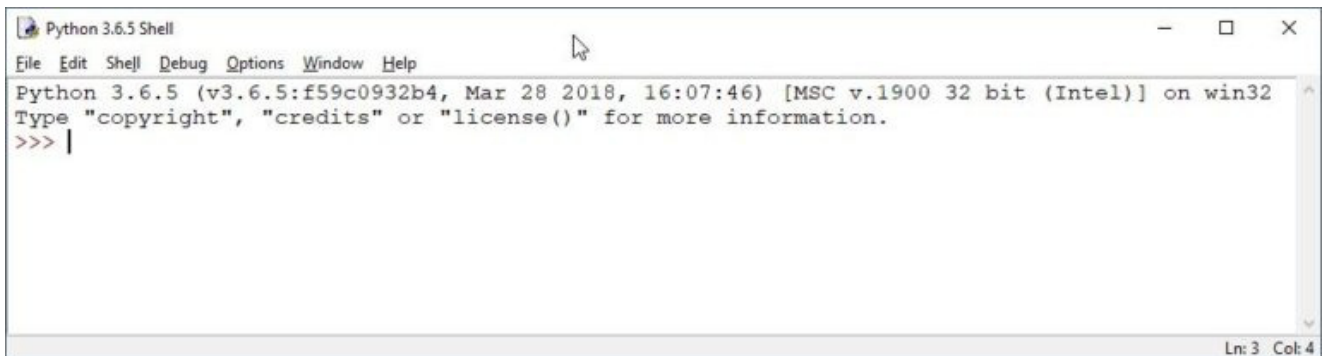
---

---

▶ 29. Subroutines	35
▶ 30. Functions	36
▶ 31. Creating, Editing & Deleting Lists	37
▶ 32. Cycling Through & Checking Lists	38
▶ 33. Searching and editing list items	39
▶ 34. Quick Sorting/Searching Lists	40
▶ 35. 2D Lists (Lists Within Lists)	41
▶ 35. 2D Lists (Lists Within Lists)	42
▶ 36. Searching/Sorting 2D Lists	43
▶ 36. Searching/Sorting 2D Lists	44
▶ 37. Advanced String Handling Part 1	45
▶ 38. Advanced String Handling Part 2	47
▶ 39. Advanced String Handling Part 3	49
▶ 40. Reading And Writing Files	50
▶ 41. Read/Write A Single Value	51
▶ 42. Reading Multiple Lines	52
▶ 43. Writing Multiple Lines	54
▶ 44. Reading Delimited Files	55
▶ 45. Writing Delimited Files	57
▶ 46. Reading A File Into A List	58
▶ 47. Writing A List To A File	60
▶ 48. Using Pickle	61

# Starting Python

- ▶ Load up the Python IDLE shell
- ▶ Hit CTRL-N to open a new window
- ▶ That's it. You are ready to go! Hit F5 to run your program



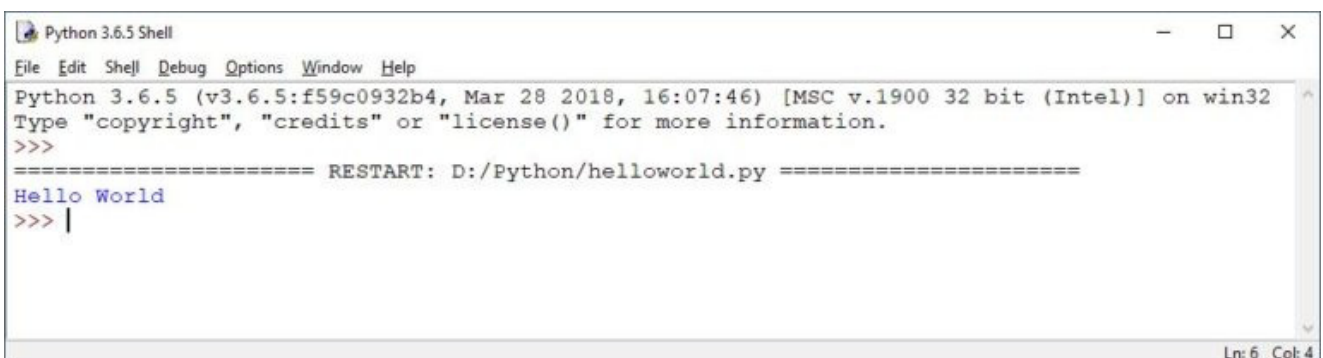
A screenshot of the Python 3.6.5 Shell window. The title bar reads "Python 3.6.5 Shell". The menu bar includes "File", "Edit", "Shell", "Debug", "Options", "Window", and "Help". The main text area shows the Python version and build information: "Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [MSC v.1900 32 bit (Intel)] on win32". Below this, it says "Type 'copyright', 'credits' or 'license()' for more information." and the prompt ">>> |" is visible. The status bar at the bottom right shows "Ln: 3 Col: 4".

▶ The Python Shell



A screenshot of the Program Window titled "\*Untitled\*". The menu bar includes "File", "Edit", "Format", "Run", "Options", "Window", and "Help". The main text area contains a single line of Python code: `print("Hello World")`. The status bar at the bottom right shows "Ln: 1 Col: 20".

▶ The Program Window



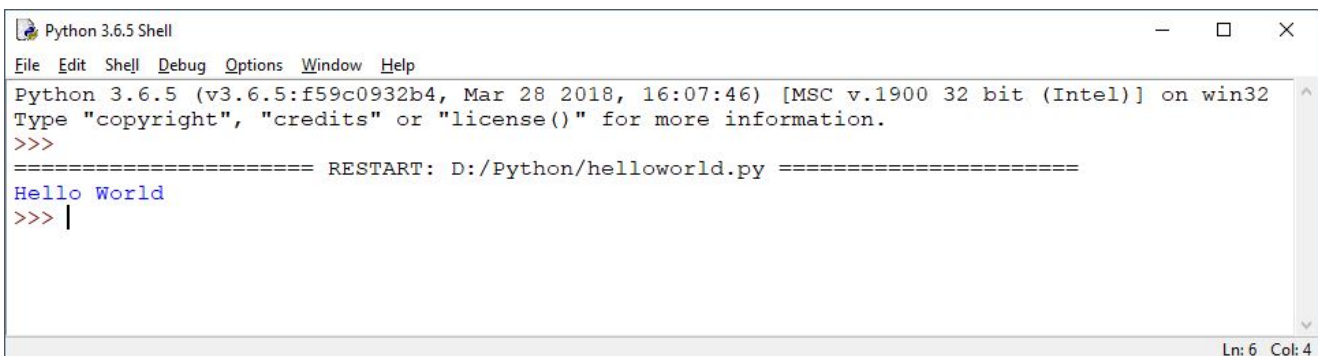
A screenshot of the Python 3.6.5 Shell window showing the output of the program. The title bar reads "Python 3.6.5 Shell". The menu bar includes "File", "Edit", "Shell", "Debug", "Options", "Window", and "Help". The main text area shows the same version and build information as the previous screenshot. Below that, it says "Type 'copyright', 'credits' or 'license()' for more information." and the prompt ">>> |" is visible. The output of the program is displayed: "==== RESTART: D:/Python/helloworld.py =====", followed by "Hello World" on a new line. The status bar at the bottom right shows "Ln: 6 Col: 4".

▶ Successful Output

# The IDLE Interpreter

- ▶ A very powerful beast. Works magic with numbers. It's kind of like a calculator on steroids. Work out what the following operations do by typing them into the IDLE interpreter shell and hitting ENTER. Try to work out what each line of code is doing.

```
2+5
67/34
30*4
4<10
3==6
22-7
23>=5
3!=5
39==39
"hello" + "world"
"hello" * 3
list(range(0,100,2))
len("hello there")
```



The screenshot shows a window titled "Python 3.6.5 Shell". The menu bar includes "File", "Edit", "Shell", "Debug", "Options", "Window", and "Help". The main text area displays the following content:

```
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: D:/Python/helloworld.py =====
Hello World
>>> |
```

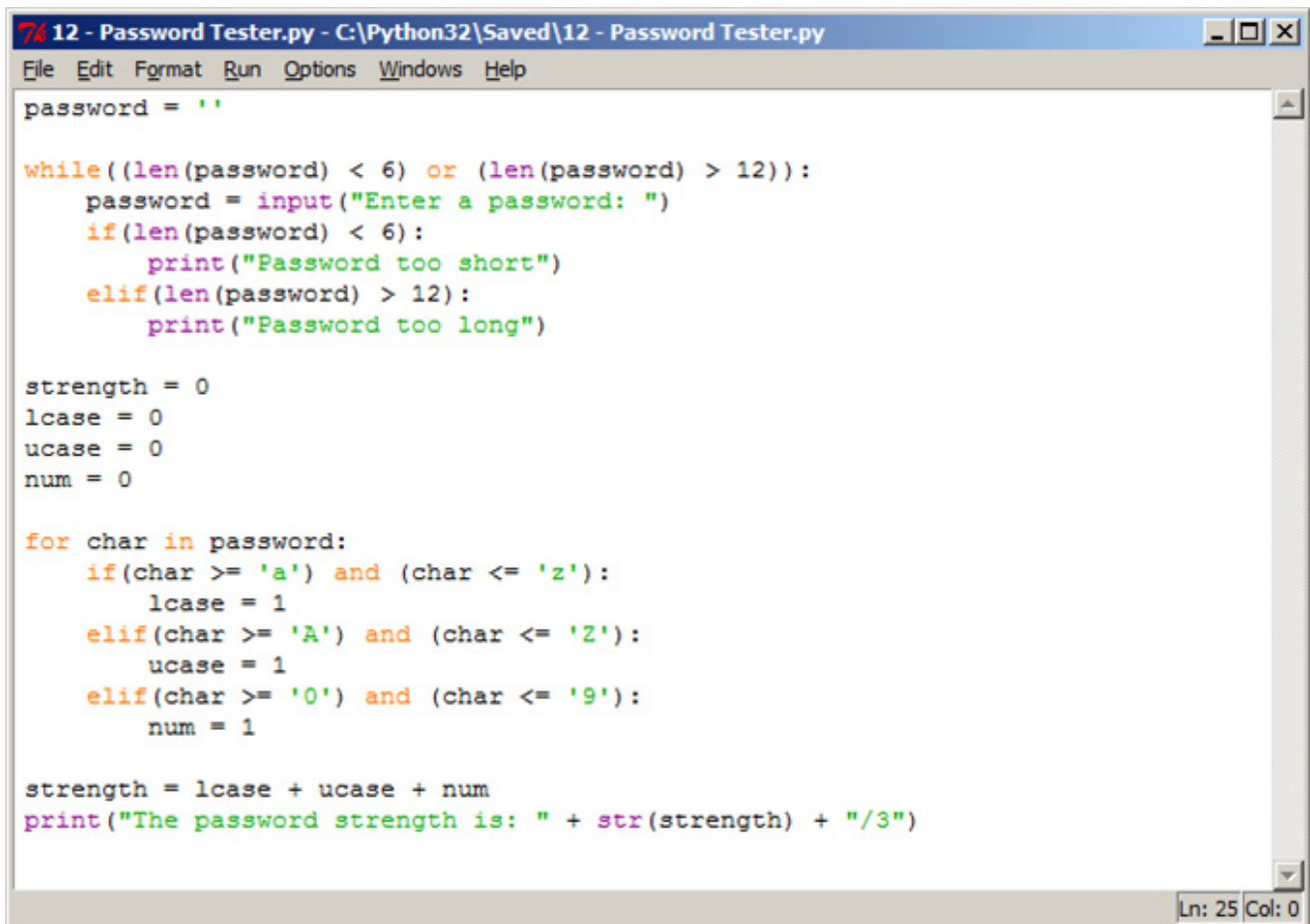
The status bar at the bottom right indicates "Ln: 6 Col: 4".

## Tasks

1. Type the commands into the IDLE interpreter, and make a note of what each line does
2. Save & label your code

# 1. Accuracy

- ▶ If you can't write code accurately, you'll have difficulty with programming
- ▶ Correct spelling and accurate use of other important symbols (brackets, colons, speech marks) is crucial
- ▶ Case also matters. Instructions are always in lowercase (highlighted orange or purple in the code below)
- ▶ Double and single quotes are interchangeable most of the time. Choose one or the other and stick to it!

A screenshot of a Python IDE window titled "12 - Password Tester.py". The window contains the following Python code:

```
password = ''

while ((len(password) < 6) or (len(password) > 12)):
    password = input("Enter a password: ")
    if (len(password) < 6):
        print("Password too short")
    elif (len(password) > 12):
        print("Password too long")

strength = 0
lcase = 0
ucase = 0
num = 0

for char in password:
    if (char >= 'a') and (char <= 'z'):
        lcase = 1
    elif (char >= 'A') and (char <= 'Z'):
        ucase = 1
    elif (char >= '0') and (char <= '9'):
        num = 1

strength = lcase + ucase + num
print("The password strength is: " + str(strength) + "/3")
```

The code is color-coded: keywords like 'while', 'if', 'elif', 'for', and 'print' are in orange; strings and variables are in purple; and operators and punctuation are in green. The window's status bar at the bottom right shows "Ln: 25 Col: 0".

## Tasks

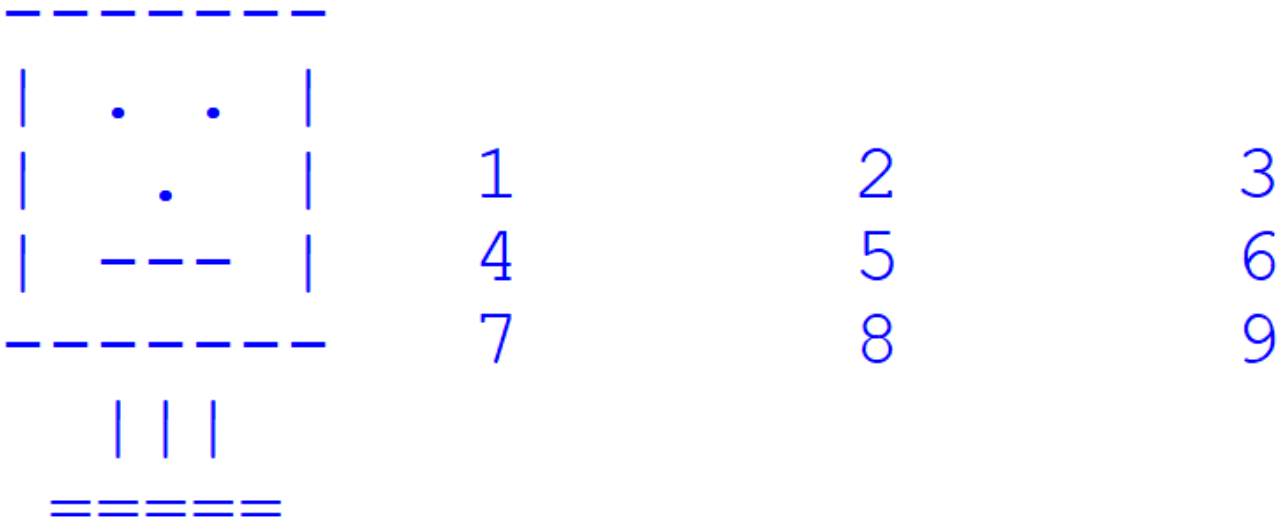
1. Type the program above into a new Python window
2. Hit F5 to run it and check it works
3. Print and label your code, explaining any parts you understand



## 2. Output

- ▶ The print() function allows you to output something to the screen
- ▶ Text must be enclosed in ""
- ▶ \n and \t are escape characters. Make sure you understand what they do

```
print("Hello World")
print("Hello\nWorld")
print("H\tE\tL\tP")
print(3+4)
```



### Tasks

1. Create programs to...
  - a. print your name three times, separated by a space
  - b. draw Robbie the robot
  - c. print the number table
2. Save and label your programs

### Extension

1. Type the commands into the IDLE interpreter, and make a note of what each line does
2. Save & label your code

## 3. Variables Part 1

---

---

- ▶ Variables are names we attach spaces in the computer memory where we can store numbers, characters and strings
- ▶ Computer memory is like a huge filing cabinet
- ▶ Each time we put something into the cabinet, we label it so we can find it again later
- ▶ I might call it x, i or total
- ▶ Make sure the variable has a suitable name with no spaces
- ▶ Use camelcase, no spaces or reserved words like print

```
x = 20
y = 40.5
z = x + y

messageA = "Hello"
messageB = "World"
messageC = messageA + messageB
```

### Tasks

1. Assign some variables using the code above
2. Print out all the variables using the print() function
3. Remember, you don't need speech marks "" to print variables!
4. Save and label your code

### Extension

1. Try setting and combining different variables. What happens when you try to add text and a number?

## 4. Variables Part 2

---

---

- ▶ You can reassign variables at any point in your program
- ▶ You can perform arithmetic on numbers, and add string variables together

### Tasks

1. Copy the code below. What is happening to the message variable?

```
message = "Hello World"
print(message)
message = "Goodbye Cruel World"
print(message)
```

1. Reassign the message variable again and print it out
2. Change the messages printed in the program

```
print(message + ". I'm going soon")
print(message + 10)
print("Hello " + message)
print(message + message)
```

1. Add the code above. One of the lines produces an error.
2. Disable the correct line using the # character at the start of the line
3. Make sure the program runs without error
4. Save and label your code

### Extension

1. Can you multiply a word? What happens if you try?

# 5. Input

---

---

- ▶ Once you understand variables, you can take input from the keyboard and store it
- ▶ All input from the keyboard is stored as a string
- ▶ The example below asks the user to Enter something and stores the input in a variable called thing

```
thing = input("Enter something: ")
```

## Tasks

1. Write a program that asks for a name, and prints out the name entered
2. Modify the code to ask for both first name and surname, and print them out on a single line
3. Add an additional line to print the first name 5 times
4. Save, print and label your code

## Extension

1. Type the commands into the IDLE interpreter, and make a note of what each line does
2. Save & label your code

## 6. Test 1 - Ask And Tell

---

---

### Main Task

1. Write a program that will
  - a. Prompt and ask for name and favourite colour.
  - b. Return a message like the example shown below (substituting in the colour entered)

```
"Hello name. My favourite colour is also colour."
```

### Extension

1. Extend your program so it asks for first name and surname separately, and prints the users full name, as well as their favourite colour

### Hints

- ▶ You will need to use the input function to read in data from the user
- ▶ You will need suitable variables to store the input
- ▶ Use a single print statement to output the message

# 7. Variable Conversion Part 1

---

---

- ▶ Variables contain either string, integer or float data
- ▶ All user input is stored as a string by default, meaning an extra step is required to convert the input to the correct data type
- ▶ For example, the program below will not output the expected result (doubling the age entered)

```
age = input("What's Your Age? ")
double_age = age + age
print(double_age)
```

- ▶ This is because you shouldn't add a number to a string of characters
- ▶ To solve this problem you convert the age to an integer as shown below

```
age = input("What's Your Age? ")
age = int(age) #Convert to Integer
double_age = age + age
print(double_age)
```

- ▶ Finally, a common technique is to convert numbers back to a string before printing them out, or displaying them as part of a message

```
print("Your doubled age is " + str(double_age))
```

## Tasks

1. Type in the first example and observe the output. Why is it incorrect?
2. Type in the second example and observe the output. What has changed?
3. Type in the final line to print out the doubled age as part of a sentence
4. Save and label your code

## Extension

1. Can you modify the program to ask 2-3 questions, and print them out as part of a sentence?

## 8. Variable Conversion Pt 2

---

- ▶ Numbers can be integer (no decimal point) or float (decimal point)
- ▶ The example below declares two variables, one of each type and prints them as integers, floating point numbers and as strings
- ▶ if no type is specified, the variable will print as it was input

```
a = 3
b = 22.54

print(a)
print(int(a))
print(float(a))
print(str(a))

print(b)
print(int(b))
print(float(b))
print(str(b))
```

### Tasks

1. Type in the code and test that it works
2. Save and label your code

### Extension

1. When converting data types, the examples above are only temporary as they are enclosed within print statements. To permanently change the data type, reassign the variable as shown below

```
a = float(a)
```

## 9. Numbers

---

---

- ▶ Variables detect what data is in them if you assign them values.
- ▶ Note strings are declared within speech marks ""

```
x = 10
y = 10.4
z = "hello"
```

- ▶ If you input from the keyboard, the computer will always treat input as a string. You can save some time by casting (converting) the data type of variable when you ask for the input

```
age = int(input("What's your age? "))
height = float(input("What's your height? "))
```

### Tasks

1. Type in the code and test that it works
2. Save and label your code

### Hints

1. Remember to use str() to convert back when printing if you are adding strings and numbers. For example:

```
print("Your age is: " + str(age))
```

### Extension

1. Use these techniques to ask the user several questions (both string and number), and print out a short story using the entered values



# 10. Working With Numbers

---

---

- ▶ Arithmetic operations are easy in Python
- ▶ Common operators to use with numbers are as follows:

Symbol	Example	Operation
()	(a + b) / c	Parenthesis
*	a * b	Multiplication
/	a / b	Division
%	a % b	Modulus
+	a + b	Addition
-	a - b	Subtraction
=	a = b	Assignment

```
a = input("Enter a whole number: ")
a = int(a)
square = a * a
print(square)
```

## Tasks

1. Write a program that asks for the width and height of a rectangle, and outputs the area
2. Write a program that asks for the radius of a circle, and outputs the area
3. Assume pi ( $\pi$ ) is 3.14159
4. The formula to calculate the area of a circle is  $\pi r^2$
5. Save, print and label your code

## Extension

1. Can you modify the programs above to calculate the volume of the three dimensional versions of these shapes? A cuboid and a cylinder.

# 11. Test 2 - Volume And Area

---

- ▶ The formula for the volume of a cylinder is  $\pi r^2 h$
- ▶ The formula for the surface area is  $(2\pi r^2) + (2\pi r h)$

## Main Task

1. Create a program that will:
  - a. Ask for the radius and height of a cylinder
  - b. Calculate the volume and surface area
  - c. Output the values separately
2. Save and label your code

## Extension

1. Create a similar program that will:
  - a. Ask for the length, width and height of a cuboid
  - b. Calculate the volume and surface area
  - c. Output the values separately
2. Save and label your code

## Hints

- ▶ Assume the user will enter numbers with decimal points
- ▶ Create an answer variable to store the result before printing
- ▶  $\pi$  ( $\pi$ ) = 3.14159265359
- ▶  $r^2$  is the same as  $r*r$

## 12. Comments And Blank Lines

---

---

- ▶ It is good practice to layout your code so it can be easily understood by others
- ▶ This includes using comments and blank lines to space out your code and label it to explain what it does
- ▶ Comment lines are ignored by the interpreter. They are used to write information useful to someone reading the code
- ▶ Look how easy it is to understand!

```
#get amount of pounds & pence to convert
pounds = float(input("Enter amount to convert: "))

#convert to euros
euros = pounds * 1.2406

#print to two decimal places
print("%.2f" % euros)

#print normally
print(euros)
```

### Tasks

1. Go through some of the examples you have completed so far and add comments and blank lines, where necessary, to make your code easier to read

# 13. Decimal Places

---

---

- ▶ Sometimes, you need to print out numbers with a set number of decimal places
- ▶ This method is useful when working with currency values or when you are dividing numbers that may create answers with many decimal places

```
#get amount of pounds & pence to convert
pounds = float(input("Enter amount to convert: "))

#convert to euros
euros = pounds * 1.2406

#print to two decimal places
print("%.2f" % euros)

#print normally
print(euros)
```

## Tasks

1. Enter the code above and test it works
2. Modify the code to print to 2 decimal places
3. Modify the code to print € symbol before the converted amount
4. Save, print and label your code

## Extension

1. There is a function that can round numbers called round(). An example is shown below. Incorporate it into your program

```
x = 3.566532
print(round(x, 4))
```

# 14. Random Numbers

---

---

- ▶ Sometimes you need to create random numbers to make 'lucky dip' choices from lists or ranges of numbers
- ▶ To create a random integer (whole number) you can use the code below:

```
#Put this at the top of your file (only once)
import random

#Generates a random number between 1 and 100
number = random.randint(1,100)

#Print the number
print(number)
```

## Tasks

1. Create a program to simulate a lottery draw. It should generate 6 numbers between 1-49 and print them appropriately
2. Run the program multiple times. Why will this program not be suitable for drawing lottery numbers?
3. Save and label your code

## Extension

1. Can you figure out a way of picking six unique numbers for the lottery draw?

# 15. Selection With IF

---

---

- ▶ IF is the first conditional statement you will use
- ▶ It allows you to execute different code depending on the circumstances
- ▶ Look how the code is indented to identify it is part of the IF statement

```
#Declare variables
letter = "a"

#Check if letter is 'a'. If so, print message
if letter == "a":
    print("The letter is a")
    print("Indented lines are included!")

#Print this message anyway because there is no indent
print("This prints no matter what happens")
```

## Tasks

1. Type in the code above and make sure it works
2. Modify the program to accept a character typed in by the user
3. Modify the program to check for the letter 'b'
4. Save, print and label your code

## Extension

1. If statements work with numbers too. Create an IF statement to check a number.

## 16. IF...ELSE

---

- ▶ The ELSE statement is added at the bottom of the IF statement to do something different if the condition is not met
- ▶ The example will execute one of the print statements, not both

```
#Declare variables
letter = "a"

#IF ELSE conditional statement
if letter == "a":
    print("the letter is a")
else:
    print("The letter is not a")
```

### Tasks

1. Modify your IF Selection program to use the ELSE statement
2. Save and label your code

### Extension

1. How could you use this technique to identify if a given number is greater, or less than zero?

## 17. IF..ELIF..ELSE

---

---

- ▶ You can check a variable for many different conditions using ELIF

```
#Declare variable
letter = "a"

#IF ELIF ELSE conditional statement
if letter == "a":
    print("The letter is a")
elif letter == "b":
    print("The letter is b")
else:
    print("The letter is not a or b")
```

### Tasks

1. Modify your IF ELSE program to use the ELIF statement to check for the letters a-f
2. Save and label your code



# 18. Checking Numbers

---

---

- ▶ Checking if a number is equal is useful, but we need to be able to do more!
- ▶ Relational operators allow us to perform a range of checks on variables
- ▶ Remember when we are comparing, use two equals rather than one!

Symbol	Example	Operation
==	if name == "John":	equal to
<	if char < "z":	less than
<=	if char <= "z":	less than or equal to
>	if char > "a":	greater than
>=	if char >= "a":	greater than or equal
!=	if char != "a":	not equal to

```
#Read in number as integer
x = int(input("Enter a number: "))

#Check x
if x < 0:
    print("x is negative")
elif x > 0:
    print("x is positive")
elif x == 0:
    print("x is 0")
else:
    print("x is not a number")
```

## Tasks

1. Type in the code above and make sure it works
2. Modify the program to print a message if the number is over 50
3. Modify the program to print a message when the number is over 100
4. Save and label your code

## Extension

1. Investigate what happens if you type in letters instead of numbers. How might you prevent these issues?

# 19. Checking Data Types

---

---

- ▶ It is very important to be able to check the type of data entered
- ▶ If you try to convert data to a type that won't 'fit', your program will crash
- ▶ The code below will crash if a non-numeric input is supplied

```
#This will crash if you enter a string
x = input("Enter a number: ")
x = int(x)
```

- ▶ The example below uses the Try/Except statement to catch the error
- ▶ The program attempts to execute the code in the try section
- ▶ If an error occurs, the code in the Except section is executed
- ▶ This works equally well at checking float() conversions too

```
#This will catch the error, print a message, and set x = 0
x = input("Enter a number: ")

try:
    x = int(x)
except:
    x = 0
    print("You didn't enter a number")
```

## Tasks

1. Type in the code above and ensure it works
2. Modify the program to output "You entered a number" if the input is valid
3. Save and label your code

## Extension

1. You can use this technique to detect an attempt to divide by zero. Use this code to create a small program that will ask for two numbers and detect any attempt to divide by zero

## 20. Logical Operators

---

- ▶ Logical operators allow you to combine checks on variables
- ▶ The two operators are OR and AND
- ▶ OR will be true if either condition is met
- ▶ AND will be true when both conditions are met
- ▶ In Python, they are always written in lowercase

```
x = 4

if (x < 1) or (x > 10):
    print("x is not between 1 and 100")

if (x > 1) and (x <= 5):
    print("x is between 1 and 5")
```

### Tasks

1. Type in the code and test that it works
2. Modify the program to check if a number entered is between 1-10, 11-20, 21-30, 31-40 or 41-50
3. Save and label your code

### Extension

1. Modify the program to accept user input, and handle unexpected input appropriately (i.e. the user does not enter a valid number)

# 21. Test 3 - Calculator

---

---

## Main Task

1. Create a basic calculator program that will take two numbers, and perform one of the following functions depending on what the user chooses:
  - a. add
  - b. subtract
  - c. multiply
  - d. divide
2. You should assume that all user input will be valid whole numbers. You should ask the user which function they wish to perform. The output should be formatted and laid out appropriately.
3. Have the program display an error message and exit if the user attempts to divide by zero
4. Save and label your code

## Extension

1. Modify the program so it prints to 2 decimal places
2. Modify the program so it checks the input numbers are valid numbers
3. Save and label your code

## Hints

- ▶ It may help to sketch out on paper the order in which you need to perform parts of the program
- ▶ You'll need to cast (convert) inputs to integers
- ▶ Think about what input your program requires
- ▶ IF/ELIF/ELSE can be used to select different arithmetic operations
- ▶ All the required knowledge required for this task is contained in the previous tasks on the website

## 22. Checking Strings

---

---

- ▶ We can check strings using the same methods as with numbers
- ▶ A character is simply a string with a length of 1
- ▶ The in operator is a quick way of checking if one string or character is contained within another – useful to check user input against an allowed list of characters or symbols
- ▶ Some operators have interesting results, and work when you least expect them

```
#Define a string & read in a character
string = "abcdefg"
char = input("Enter a character: ")

#Check if char is in string
if char in string:
    print("The character is in the string!")
else:
    print("The character was not found!")

#More general comparison using IF
if char > "m":
    print("Character is after m in the alphabet")
```

### Tasks

1. Type in the code above and make sure it works
2. Modify the program to check if the character entered is in the alphabet (there are two ways to do this using the examples shown)
3. Save and label your code

### Extension

1. The use of the NOT operator can be used as follows. Experiment.

```
if char not in string:
    print("The character is not in the string!")
```

## 23. Operators And Strings

- ▶ Characters can be checked and compared in much the same way as numbers, using the same symbols
- ▶ Less than (<) and greater than (>) work if you think about the alphabet as going from low (a) to high (z)
- ▶ Remember that lowercase and uppercase letters are recognised as different characters by the computer

Symbol	Example	Operation
==	if name == "John":	equal to
<	if char < "z":	lower in the alphabet
<=	if char <= "z":	less than or equal in the alphabet
>	if char > "a":	greater in the alphabet
>=	if char >= "a":	greater than or equal to in alphabet
!=	if char != "a":	not equal to
in	if char in string:	is character in string?

### ASCII

- ▶ Less than (<) and greater than (>) works because the computer assigns all letters a preset numerical value, documented in the ASCII character set

```
#Use ord() to find the ASCII value of a character
print(ord("A"))

#Use chr() to find the character associated with an ASCII number
print(chr(65))
```

### Tasks

1. Type in the code above and check it works
2. Modify the program to accept user input
3. Save and label your code

### Extension

1. You can combine these techniques with an IF statement to check for any allowed alphanumeric character (look at the table and find the lowest, and highest numbers that identify the range of characters you want to accept)

# 24. Other String Things Part 1

---

- ▶ There are other useful functions for checking and manipulating characters and strings

```
#Defines and prints original string
str = "AbCdEfG"
print(str)

#Checks to use in IF statements
print(str.isalpha())
print(str.islower())
print(str.isupper())
print(str.isdigit())

#Ways to convert strings
print(str.upper())
print(str.lower())
print(str.swapcase())
print(str.capitalize())

#Length of string
print(len(str))
```

## Reversing A String

- ▶ Sometimes you might need to reverse a string. This may be to make a working with binary strings easier, or to help create random passwords.

```
#This code reads the string variable,
#reverses it into the r_str variable
r_str = string[::-1]
```

## Tasks

1. Type in the code and make sure that it works
2. Add code to reverse the string and print it out
3. Save and label your code

## Extension

1. You can combine these techniques with an IF statement to ask the user to input a letter, and print a message if the input is 1 in length, and is an uppercase letter.

## 25. Other String Things Part 2

---

---

- ▶ All the different ways of checking strings and characters on the last page return either TRUE (1) or FALSE (0). This means that they can be used with IF statements
- ▶ If you played around with the input string, you should have realised that `islower()` and `isupper()` only return true when the entire string is a particular case

```
#Ask for a character
character = input("Enter a character: ")

#Check for a password length
if len(character) != 1:
    print("Only one character should be entered")

#Check if password is lowercase
if character.islower() == 1:
    print("Character is lowercase")
```

### Tasks

1. Type in the code above and test it works
2. Modify the program to print out a message to say if the character is uppercase or lowercase
3. Modify the program to print a message if the character is a number
4. Save and label your program

### Hints

- ▶ You can use these techniques to perform rigorous checks on user input to protect your program
- ▶ Checking user input is called validation and it helps make sure your program runs as expected



## 26. Count Controlled Iteration

---

- ▶ One of the things that makes a computer useful is the ability to repeat an operation many times over at high speeds. This is called iteration or looping
- ▶ The FOR loop allows us to carry out a set of instructions a specific number of times
- ▶ The code within the loop must be indented
- ▶ The range arguments set the number of loops to perform: range(start,finish)
- ▶ The counter variable i displays the number of times the loop has executed
- ▶ Remember that the computer always starts counting at 0!

```
#Basic for loop to cycle a number of times
for i in range(0,5):
    print("hello")

#Loop that prints the counter
for i in range(0,5):
    print(i)
```

### Tasks

1. Type in the code and make sure that it works
2. Modify the program so it prints out a list of numbers: 1-20
3. Modify the program so it takes in a word and number, and prints the word the number of times specified
4. Save and label your code

### Extension

1. The code below adds an extra parameter to the range() function that controls the step, i.e. how many it jumps by each time it counts. The example below counts up in twos from zero to one hundred. Can you work out how to get the loop to count down from 100?

```
#Basic for loop to cycle a number of times
for i in range(0,100,2):
    print(i)
```

## 27. Condition Controlled Iteration

---

---

- ▶ Remember, indenting your code indicates it is inside a loop
- ▶ Always set your variable properly so the loop runs at least once
- ▶ Don't forget the colon to start the conditional statement

```
#Set x to 0
x = 0

#Loop while x is less than 5
while x < 5:
    print("looping")
    #Add 1 to x each time the loop runs
    x = x + 1
```

### Tasks

1. Type in the code and make sure that it works
2. Change the code so it loops 20 times and prints out x each time
3. Modify the program so it prints out the squares of the numbers 1-10 (hint: multiply the counter by itself)
4. Save and label your code

### Extension

1. Create a loop that will ask for a password and continue to do so until 'apple' is entered. A hint is shown below, but what other code do you need to make it work (use the example above to help)?

```
while password != "apple":
```

## 28. Test 4 - Iteration

---

---

### Main Task

1. Create a program that:
  - a. Randomly generates a number between 1 -100
  - b. Asks the user to guess the number until they get it right
  - c. Says whether the guess is too high or too low
  - d. Says when they have guessed correctly, and quits

### Extension

1. Modify the program so it records the number of guesses taken

### Hints

- ▶ It may help to sketch out on paper the order in which you need to perform parts of the program
- ▶ A good way of doing this is pseudocode, an example of which is shown below
- ▶ Write pretty much what you like, but each line should translate directly into a line in your program. You should keep indents to show the bits that loop

```
import the random library
randomly generate a number between 1-100
set attempts to 0
set guess to 0
while the guess is not equal to the random number
    ask for a guess to be input
    add one to the number of attempts
    if guess is less than random number tell user
    if guess is more than random number tell user
print a well done message and display the guessed number
print the number of attempts
```

## 29. Subroutines

---

- ▶ Subroutines (sometimes known as procedures) allow you to split your program into reusable chunks that you can call from any other part of your program
- ▶ Subroutines always appear before the main program
- ▶ They can work on their own or you can pass them values
- ▶ Subroutines do not return values to the main program

```
#Subroutines first
def mySub():
    for i in range(0,3):
        print("hello")

def mySub2(word,times):
    for i in range(0,times):
        print(word)

#Main program starts below subroutines

#Call with no parameters
mySub()

#Call with parameters
mySub2('blah',5)
```

### Tasks

1. Type the code and make sure it works
2. Create a subroutine that prints the cube of a number passed to it
3. Save and label your code

### Extension

1. You can create subroutines to do many things. Create subroutines to perform basic calculations (add, subtract, multiply, divide) that you can pass values to

## 30. Functions

---

- ▶ The subroutines on the other page are simply procedures that run the code within then when called. The other type is a function that returns value(s) rather than printing to the screen

```
#Functions first
def double(number):
    doubled = number * 2
    return doubled

#Execute function and print returned value
print(double(10))

#Call function, pass a value (10) and save return value in
doubled variable
doubled = double(10)
print(doubled)

#Use in IF statements
x = 25
if double(x) > 100:
    print("It's a big one!")
```

### Tasks

1. Type in the code and make sure it works.
2. Create a function that takes the radius of a circle, and returns the area of a circle  $\pi r^2$
3. Save and label your code

### Extension

1. Functions can return more than one value, and can do so at any point using the return command. Create the code to call this function, and modify the code to accept input from the user

```
def pos_or_neg(x):
    if x > 0:
        return "Positive"
    elif x < 0:
        return "Negative"
    else:
        return "0"

#Call the function here
#??
```

# 31. Creating, Editing & Deleting Lists

---

- ▶ Lists are essential when using Python to store large amounts of information, especially if we don't know how much in advance
- ▶ A list has a name, just like a variable but there may be many lists within it too (known as two-dimensional lists (that's for later!))
- ▶ Each list has an index, starting at 0
- ▶ The example below is known as a one-dimensional list

```
#Create a list containing 4 names
#Leave square brackets empty to create an empty list
names = ['Joe', 'Mike', 'Sarah', 'Charlie']

#Lists can also contain numbers
numbers = [23, 45, 6, 9, 10]

#Will print "Sarah"
print(names[2])

#Add to the end of the list
names.append("Alf")

#Remove a name from the list
names.remove("Mike")

#Remove by index number if known, no number pops last the list
names.pop(2)

#Find the length of the list (returns a numerical value)
list_length = len(names)

#Empty the list entirely
names.clear()
```

## Tasks

1. Type in the code and make sure that it works
2. Add print statements at relevant points to show what effect each line has on the list
3. Save and label your code

## Extension

1. A while loop is frequently used to allow a user to keep adding new names to the list, until a specific character (such as 'Q') is detected, when it will stop and print out the list. Can you implement this technique?

## 32. Cycling Through & Checking Lists

---

---

- ▶ FOR and WHILE loops are essential to make full use of lists with the minimum amount of code
- ▶ The len() function can tell you how big your list is
- ▶ The in operator will be your best friend soon. It makes life very easy

```
#List containing 4 names + print (good for testing)
names = ["Joe", "Mike", "Sarah", "Charlie"]

#Get length and print
length = len(names)
print(length)

#Using the 'in' operator to cycle through each name and print.
#The IF statement checks for "Joe" and finds index
for name in names:
    print(name)
    if name == "Joe":
        print("Hey Joe, love Jimi!")
        position = names.index("Joe")
        print("Found at index: " + str(position))
```

### Tasks

1. Type in the code and make sure that it works
2. Modify the program to allow the user to input a name to search for
3. Modify to use a list of 10 names, and print custom messages for two other names
4. Save and label your code

### Extension

1. The index number is frequently used to identify the exact position of an item in the list, usually so you can delete it using the code below, the position variable containing the index of the item to remove. Modify the program to ask if the name should be deleted if it is found in the list

```
names.pop(position)
```

## 33. Searching and editing list items

---

- ▶ The index number of the list item allows you to edit and delete particular items
- ▶ This example uses the in operator to quickly check the whole list
- ▶ If the search term exists, it then looks up the index, and uses this to directly edit the name "Sarah", changing it to "Sara"

```
#List containing 4 names
names = ['Joe','Mike','Sarah','Charlie']

#Checking a list using 'in' and IF
name = 'Sarah'
if name in names:

    #Print found name message
    print(name + " found at location: ")

    #Grab index number of current name and print
    index = names.index(name)
    print(index)

    #Use the index to change the item
    names[index] = "Sara"

    #Use the index to delete the item
    names.pop(index)
```

### Tasks

1. Type in the code and make sure it works
2. Modify the program to ask the user for a name, search for it and allow the user to update it if found
3. Save and label your code

### Extension

1. This 'search' technique is essential knowledge because you must be able to search in order to delete, or edit items in a list that already exists. Can you modify the program to offer an option of edit (or) delete if the name is found?



## 34. Quick Sorting/Searching Lists

---

---

- ▶ A couple more tricks can be performed with lists to make them really useful

```
#List containing 4 names, and print
names = ["Joe", "Mike", "Sarah", "Charlie"]
print(names)

#Sort and reprint the list Ascending (A-Z)
names.sort()
print(names)

#Sort and reprint the list Descending (Z-A)
names.sort(reverse=True)
print(names)

#A way to search a specific record and get the index
#without using for loop. This is no good for lists
#containing duplicate values
index = names.index("Mike")
print(index)
```

### Tasks

1. Type in the code and make sure it works
2. Create a new list with 10 numbers in it, and print them sorted into order
3. Modify the program to accept user input to add more items to the list
4. Save, print and label your code

### Extension

1. When you have a list with duplicate values, the quick search technique above will not work. When faced with duplicate values, you must use a FOR loop to cycle through the list and check each item individually, and count, or display the occurrences. Can you do this? Refer back to the earlier list tasks to help.

## 35. 2D Lists (Lists Within Lists)

---

---

- ▶ This may sound complicated, but lists within lists are the best way to save 'records' i.e. multiple bits of information about one thing, such as the name, address and telephone number of a user
- ▶ This type of list is known as a two-dimensional list
- ▶ Lists are sometimes referred to as 'arrays'

```
#Create a new list, each list item containing two values
users = [
    ["Mike","Smith",24],
    ["John","Thompson",56],
    ["Mary","Field",19]
]

#Add additional record
users.append(["Katie","Johnson",15])

#Use elements in loop ([0] is name, [1] is surname, [2] is age)
for user in users:
    print(user[0],user[1],user[2])

#Display a specific item (this example will display '56'
print(users[1][2])

#Access individual parts of each item (change a record)
#Indexes work down, then across
users[0][0] = "Michael"
users[0][1] = "Thomas"

#Messy, but quick method of printing an entire list
print(users)
```

## 35. 2D Lists (Lists Within Lists)

---

---

Index	0	1	2
0	Mike	Smith	24
1	John	Thompson	56
2	Mary	Field	19

### Tasks

1. Type in the code and make sure it works
2. Add a new field to the list items to hold the e-mail address of each person, and show you can print and edit it
3. Save and label your code

### Extension

1. Creating code to neatly print out a list is important. Use a for loop and appropriate escape characters (\n) or (\t) to display the list as a table. There are several ways to do this, the most efficient using two FOR loops, one inside the other

## 36. Searching/Sorting 2D Lists

---

- ▶ Searching through two-dimensional lists is best done with a FOR loop
- ▶ Each time, you can use an IF statement to check a particular part of the list item
- ▶ Sorting a 2D list is easy as long as the field to sort by is the first item (first name in the example below)

```
#Create a new list, each list item containing three items
users = [
["Mike","Smith",24],
["John","Thompson",56],
["Mary","Field",19],
["Katie","Johnson",15]
]

#Name to search for
searchSurname = input("Enter surname: ")

#Loop. If the name is found, the location will be printed
for user in users:

    #Compare the searchSurname to the name in the list
    if user[1] == searchSurname:

        #Get & print the index location of the matching name
        foundat = users.index(user)
        print("Found at index: ",foundat)

        #Ask if the user wishes to delete the record
        q = input("Press (y) to delete this record: ")

        #If the user chooses 'y'
        if q == "y":

            #Delete the record using the index
            users.pop(foundat)

#Sort and print the list once the loop has finished
users.sort()
print(users)
```

## 36. Searching/Sorting 2D Lists

---

---

### Tasks

1. Type in the code above and make sure that it works
2. Add 3 more people into the original list
3. Modify the program to search for first name
4. Save and label your code

### Extension

1. Investigate how this program reacts if there are duplicate names in the list. What happens when you search and there are two people named 'Smith' for example? You can also modify this code to allow the editing of a name if it is found

## 37. Advanced String Handling Part 1

---

---

- ▶ Using loops and comparisons, you can do some pretty nifty things with strings of characters
- ▶ Performing comparisons is much more powerful when you use loops to inspect each individual character
- ▶ Note the in operator works on strings as well as lists

```
#Declare a string
string = "John Henry was a desperate little man."

#Loop through each letter in the string, and print with a space
after each
for letter in string:
    print(letter)

#Quick length calculation
string_length = len(string)
print("String Length:",string_length)

#Set counters to 0
vowels = 0
others = 0

#Loop string and count vowels
for letter in string:

    #if a vowel...
    if letter in "aeiou":
        vowels += 1

    #if not a vowel
    else:
        others += 1

#Print the total counts once complete
print("Vowels: ",vowels)
print("Others: ",others)
```

# 37. Advanced String Handling Part 1

---

## Tasks

1. Type in the code and test it works
2. Modify the program to count uppercase and lowercase letters
3. Modify the program to count punctuation and spaces
4. Save and label your code

## Extension

1. The `break` command allows you to exit a for loop at any time. You can use this to modify your program to print an error and quit if a number is detected in the input string.

## 38. Advanced String Handling Part 2

---

- ▶ Multiple items of information are sometimes stored on each line in a file that your program might load up. They might be separated by commas or spaces and you can easily split the lines up and re-join them once you have finished
- ▶ Dates and product codes are frequently manipulated using these techniques

```
#date
date = "12/04/1981"

#Simple split function into single variables
day,month,year = date.split("/")

#Split the date into a list at (/) slash characters (alternate)
split_date = []
split_date = date.split("/")

#Grab items from split_date list
day = split_date[0]
month = split_date[1]
year = split_date[2]

#Rejoin date parts with a different delimiter (:) colon
date = day + ":" + month + ":" + year
print(date)
```

### Tasks

1. Type in the code and test it works
2. Modify the program to count uppercase and lowercase letters
3. Modify the program to count punctuation and spaces
4. Save and label your code



## 38. Advanced String Handling Part 2

---

---

### Extension

1. The `split()` function will not work if the number of splits does not match the number of variables. For example:

```
date = "12/04"  
day,month,year = date.split("/") #Won't work
```

2. Good practice would be to use a for loop/if statement construct to check the number of slashes before splitting. How might you implement this?

## 39. Advanced String Handling Part 3

---

---

- ▶ You can chop up strings and check individual parts when the input may be multiple words or numbers separated by spaces or particular characters such as full names or dates
- ▶ In the last example you split strings into an array but you can access individual bits of them by number to save time
- ▶ Remember to use if statements to validate input!

```
#Date variable
date = input("Enter the date (dd/mm/yyyy): ")

#Access the string directly
day = date[0:2]
month = date[3:5]
year = date [8:12]

#You can't change individual bits of the string though
#date[0:2] = "15"

#You might recreate the original instead after updates
day = "15"
date = day + "/" + month + "/" + year

#Print new date string
print(date)
```

### Tasks

1. Type in the code and test it works
2. Save, print and label your code

## 40. Reading And Writing Files

---

---

- ▶ Reading and writing files is fairly simple if you know what you want to load and save
- ▶ There are simple techniques for reading and writing individual items of data to and from a text file, a line at a time or all at once
- ▶ Reading multiple lines is usually done with a for loop and the contents are loaded into a list within the program. Using a list means your program can handle files containing different amounts of data.
- ▶ Depending on what you are using files for, it pays to have planned exactly what you are loading and saving from your file. Questions to ask yourself:
- ▶ What am I actually storing on each line of the file?
- ▶ Is it an unorganised list or are the items in a particular order?
- ▶ Are there a set number of lines that will be used?
- ▶ Do you need to read/write each line individually?
- ▶ Do you need to add or delete individual items in the file?
- ▶ Do you need to store multiple items of information on each line? If so, how will they be separated?

```
#Open a file (r = read, w = write, a = append)
file = open("users.txt","r")

#Do stuff here

#Close the file
file.close()
```

# 41. Read/Write A Single Value

- ▶ The simplest thing you might need to do with files is to load or save a single item of data
- ▶ The example below uses a text file called savedfile.txt to read/write a number
- ▶ The program will produce an error if it does not exist so create it first (in the same directory where your program is saved)

```
#Open file for reading, read the contents
#of the file and close
file = open("savedfile.txt","r")
value = file.read()
file.close()

#Print the value from the file
print(value)

#Open file for writing, Write 37 to
#the file and close
file = open("savedfile.txt","w")
file.write("37")
file.close()
```



## Tasks

1. Type in the code to test it works
2. Modify the code to accept user input of a new number to save
3. Try saving and loading strings, does it work?
4. Save and label your code

## Extension

1. This technique was how early website counters worked. Each time the code ran, it loads the number from the file, adds 1 and saves it back to the file. Can you implement this technique?

## 42. Reading Multiple Lines

---

---

- ▶ This example uses a file savedfile.txt with a list of names and reads in each line at a time
- ▶ Each time readline() is called, the file advances by one line
- ▶ This is useful if you know exactly where you saved specific items in the file, and you know how many lines there are in total
- ▶ However, it is no use if you don't know how many lines are in the file

```
#Open file for reading
file = open("savedfile.txt","r")

#Read one line after another
line1 = file.readline()
line2 = file.readline()
line3 = file.readline()
line4 = file.readline()

#Close file
file.close()

#Each line when it is read in has a hidden \n at the end that
produces
#extra line breaks between each print. Remove it with strip()
line1 = line1.strip("\n")
line2 = line2.strip("\n")
line3 = line3.strip("\n")
line4 = line4.strip("\n")

#Print the lines from the file that have been read in
print(line1)
print(line2)
print(line3)
print(line4)
```

## 42. Reading Multiple Lines

---

---



```
*savedfile - Notepad
File Edit Format View Help
Daniel
Mike
Cathy
Georgia
Ln 4, Col 8 100% Windows (CRLF) UTF-8
```

### Tasks

1. Type in the code and make sure it works
2. Modify the program to read the lines into a list and print the contents
3. Save and label your code

### Extension

1. One technique to store more than one item per line, is to put a delimiter between them, usually a comma (,) and you can use the `split()` function to separate the parts of the line when you read it in. Add data into the text file and try to get this technique to work

## 43. Writing Multiple Lines

---

---

- ▶ When writing to files, you can either overwrite the information in the file (w), or add to it as you go along (a). This is set as shown below. Choose one or the other, not both
- ▶ There is a function called writeline() that you might assume is the opposite of readline(). Using this is slightly different and we will simulate the same effect by adding a line break \n each time we write a line to the file

```
#Open file for appending (add to)
file = open("savedfile.txt", "a")

#Open file for writing (overwrite)
file = open("savedfile.txt", "w")

#Write one line at a time, note the addition of a line break \n
on the ends
file.write("Jemima\n")
file.write("Asif\n")
file.write("Susan\n")

#Close file
file.close()
```

### Tasks

1. Type in the code and make sure it works - append will add names to the text file every time you run the code. Write will save over the previous names in the file
2. Modify the program to accept user input to add 5 names to the file. There are several ways to do this, the most efficient using a for loop that runs 5 times
3. Save and label your code

### Extension

1. A while loop can be used to keep asking for names, and writing them to the file until a specific letter or character is typed in, such as 'X'. How could you implement this feature? Can you make it work?

## 44. Reading Delimited Files

- ▶ Sometimes, you need to read in files from other programs containing information such as student records or address details.
- ▶ We use files known as comma separated values files
- ▶ They are simply text files with a different extension (.csv)
- ▶ Each line will have multiple items on it, each usually separated by a comma. It might look like the example below
- ▶ When working with files, make sure they are NOT open in notepad whilst Python is running
- ▶ There should be NO spaces between line items



```
addresses.csv - Notepad
File Edit Format View Help
Ken McCall,65 Lincoln Blvd,Washington,07989 345664
Joe Smith,21 Jefferson Ave,Washington,07664 354748
Ally Major,99 Kennedy St,Washington,08775 467362
```

```
#Open file for reading
file = open("addresses.csv","r")

#Read into list and split at commas
for line in file:

    #Strip the \n from the end of the line
    line = line.strip("\n")

    #Split the line into separate variables
    name,address,state,phone = line.split(",")

    #Print the variables
    print(name,address,state,phone)

#Close file, contents now in list
file.close()
```



# 44. Reading Delimited Files

---

## Tasks

1. Create a suitable .csv file with details of 5 people in it
2. Type in the code above and make sure it works
3. Save and label your code

## Extension

1. This code is very basic and will break very easily if the file does not exist. A function called `exists()` can be used to check for a file before trying to open it. Do some research and try to implement a feature that detects if the file exists

## 45. Writing Delimited Files

- ▶ Writing delimited files is easy. You simply combine the items for each line, separated by a comma, with a `\n` on the end and then write it to the file



```
addresses.csv - Notepad
File Edit Format View Help
Ken McCall,65 Lincoln Blvd,Washington,07989 345664
Joe Smith,21 Jefferson Ave,Washington,07664 354748
Ally Major,99 Kennedy St,Washington,08775 467362
```

```
#Open file for append (add to)
file = open('addresses.csv', 'a')

name = "Jay Jones"
address = "56 3rd St"
city = "Delaware"
phone = "08556 474562"

#Write a line separated by commas, \n on the end
file.write(name + "," + address + "," + city + "," + phone + "\n")

#Close file
file.close()
```

### Tasks

1. Type in the code above and make sure it works
2. Modify the program to accept user input
3. Save and label your code

### Extension

1. You can implement a for or while loop in your program that will allow you to enter details for multiple people, either until a specific character is pressed or a set number of details have been entered. Can you make this work?

## 46. Reading A File Into A List

---

---

- ▶ A really important thing you will need to be able to do is read and write text files with variable numbers of lines
- ▶ Because we don't know how many lines there will be, we tend to read text files into a list, make any changes, and then write the whole list back over the file

```
#Define blank list
file_contents = []

#Open the file
file = open("addresses.csv","r")

#Loop through each line
for line in file:

    #Strip the \n from the line
    line = line.strip("\n")

    #Split the line into parts
    name,address,city,phone = line.split(",")

    #Add items to file_contents list
    file_contents.append([name,address,city,phone])

#Close the file
file.close()

#Messy print
print(file_contents)
```

### Tasks

1. Type in the code and make sure it works
2. Modify the code to allow you to choose a filename
3. Modify the code to print the list in a neater fashion
4. Save and label your code

## 46. Reading A File Into A List

---

---

### Extension

1. You can make this code more efficient by combining the splitting and appending into one line (shown below). Implement this improvement:

```
file_contents.append(line.split(","))
```

## 47. Writing A List To A File

---

---

- ▶ A really important thing you will need to be able to do is read and write text files with variable numbers of lines
- ▶ Because we don't know how many lines there will be, we tend to read text files into a list, make any changes, and then write the whole list back over the file

```
#List held in program
address_list = [
    ['Ken McCall', '65 Lincoln Blvd', 'Washington', '07989 345664'],
    ['Joe Smith', '21 Jefferson Ave', 'Washington', '07664 354748'],
    ['Ally Major', '99 Kennedy St', 'Washington', '08775 467362'],
    ['Jay Jones', '57 3rd St', 'Delaware', '08556 474562']
]

#Open the file
file = open("addresses.csv","w")

#Loop through each line of the list
for line in address_list:

    #Build line to write to the file, each separated by a comma,
    #and a line break \n at the end
    file.write(line[0] + "," + line[1] + "," + line[2] + "," +
line[3] + "\n")

#Close the file
file.close()
```

### Tasks

1. Type in the code and make sure it works
2. Add two more people into the list (there are several ways to do this)
3. Modify the code to allow you to choose a filename
4. Save and label your code

### Extension

1. Python can be buggy if it tries to write to a file that is open in another program such as notepad. Try it and see what happens

## 48. Using Pickle

---

- ▶ If you use lists to store the data from your program, there is a neat function called `pickle()` that can be used to save the entire list rather than using loops
- ▶ The example below saves a list to a file and then reloads it
- ▶ The data is not stored as text though, so you can't edit it by hand
- ▶ You need to declare the pickle library at the start of your program

```
#Import pickle library
import pickle

#List held in program
address_list = [
    ['Ken McCall', '65 Lincoln Blvd', 'Washington', '07989 345664'],
    ['Joe Smith', '21 Jefferson Ave', 'Washington', '07664 354748'],
    ['Ally Major', '99 Kennedy St', 'Washington', '08775 467362'],
    ['Jay Jones', '57 3rd St', 'Delaware', '08556 474562']
]

#Open file for write (binary), save list to file and close
file = open('savedfile.dat','wb')
pickle.dump(address_list,file)
file.close()

#Clear the address list
address_list.clear()

#Open file for reading (binary), read back into list and close
file = open('savedfile.dat','rb')
address_list = pickle.load(file)
file.close()

#Print list contents
print(address_list)
```

## 48. Using Pickle

---

### Tasks

1. Type in the code and make sure it works
2. Open the .dat file in notepad. What do you see?
3. Save and label your code

### Extension

1. You cannot create pickle data files using notepad, so you need to build code to generate a list that you then save to the file. From that point on you need to read into your list at the start of the program, and save it back out at the end. Removing the list declaration at the top, and swapping round the read and write sections in the program above will make sure your program does this properly