

String Concatenation

We can combine multiple strings together to form one string. This is called Concatenation.

Use a + operator to concatenate (combine) strings together. This only works with strings.

```
first_name = "Homer"  
last_name = "Simpson"  
full_name = first_name + " " + last_name  
print(full_name)
```

Homer Simpson

This creates a space.

Length of String

We can also check the length of a string by using len()

Code

```
string = "This string contains forty-two characters."  
print(len(string))
```

Output

42

Using Input Statements

Most programs accept data from the user, process it and output a result.

After declaring a variable, you can use the **input()** function to allow the user to input data.

```
name = input()
```

Within the brackets, you can add a statement to prompt the user of what to enter.

```
name = input("What is your name?")
```

The response to this will now be assigned to the variable, **name**. For example, if the user entered "king alfred" the value king alfred will be known as name.

Outputting Data

Most programs accept data from the user, process it in some way, and output a result

```
print("How many hours a night do you sleep?")
```

```
hoursPerNight = input()
```

```
hoursPerWeek = hoursPerNight * 7
```

```
print(hoursPerWeek)
```

Using the **print()** function will output the result and/or data to the user.

Arithmetic Operators (in programming)

Addition	+
Subtract	-
Division	*
Multiply	/
Exponent (power of)	^
MOD	%
DIV	//

Using MOD

Using MOD divides a number and returns the remainder.

```
13 MOD 2 = 1
```

This is because 2 goes in to 13 6 times with 1 remaining.

Using DIV

Using MOD divides a number and returns the remainder.

```
13 DIV 2 = 6
```

This is because 2 goes in 13 6 times.

Sequence

Sequence are statements executed one by one in the order they are written:

```
mark1 = 78
mark2 = 67
total = mark1 + mark2
average = total / 2
print(average)
```

Selection

An IF statement is a selection statement. The next statement to be executed depends on whether the condition being tested is True or False.

```
if average >= 80 then
print("Distinction")
else
print("Pass")
endif
```

Comparison operators	Meaning	Pseudocode example	Result
==	Equal to	5 == 5	True
!=	Not equal to	5 != 5	False
>	Greater than	5 > 5	False
>=	Greater than or equal to	5 >= 5	True
<	Less than	5 < 5	False
<=	Less than or equal to	5 <= 5	True

Iteration

Iteration mean repetition of a section of code.

There are three types of iteration statement in most languages such as Visual Basic, Java, and C#

```
for ... next  
while ... endwhile  
do ... until
```

For Loops

Use this when you want to execute the loop a specified number of times

```
total = 0  
for i = 1 to 7  
    dailyTemperature = input()  
    total = total + dailyTemperature  
next i  
averageWeeksTemp = total / 7  
print(averageWeeksTemp)
```

Do until loops

Use this when you want to execute the loop until a certain condition is True. The loop will always execute once
The condition is tested at the end of the loop

```
password = ""  
do  
    password = input("Invalid password - try again")  
until password == "jd8UpP2+d"  
print("Correct password")
```

While Loops

Use this when you want to execute the loop while a certain condition is true.

```
password = input("Please enter password: ")  
while password != "jd8UpP2+d"  
    password = input("Invalid password - try again")  
endwhile  
print("Correct password")
```

Sequence

Sequence are statements executed one by one in the order they are written:

```
mark1 = 78
mark2 = 67
total = mark1 + mark2
average = total / 2
print(average)
```

Selection

An IF statement is a selection statement. The next statement to be executed depends on whether the condition being tested is True or False.

```
if average >= 80 then
print("Distinction")
else
print("Pass")
endif
```

Comparison operators	Meaning	Pseudocode example	Result
==	Equal to	5 == 5	True
!=	Not equal to	5 != 5	False
>	Greater than	5 > 5	False
>=	Greater than or equal to	5 >= 5	True
<	Less than	5 < 5	False
<=	Less than or equal to	5 <= 5	True

Arrays

An array is a data structure that allows you to hold many items of data which is referenced by one identifier. All items of data in the array must be of the same data type.

For example:

```
names = ["Adam", "Bob", "Charlie", "Dan", "Ethan"]
```

Note that arrays use square brackets.

Accessing Elements in an Array

```
names = ["Adam", "Bob", "Charlie", "Dan", "Ethan"]  
         0         1         2         3         4
```

In order to access elements inside an Array, we need to specify the index position.

```
print(names[3])
```

→ Dan

↑
Index position inside square brackets.

2d Arrays

Two dimensional array is an array within an array. In this type of array the position of an data element is referred by two indices instead of one.

```
log_in_details = [ ["User1", "Pass1"],  
                  ["User2", "Pass2"],  
                  ["User3", "Pass3"],  
                  ["User4", "Pass4"] ]
```

```
print(log_in_details[0][1])  
print(log_in_details[2][1])  
print(log_in_details[3][0])
```



First number is the row
Second number is the
collum

	0	1
0	"User 1"	"Pass 1"
1	"User 2"	"Pass 2"
2	"User 3"	"Pass 3"
3	"User 4"	"Pass 4"

Length of Array

```
names = ["Adam", "Bob", "Charlie", "Dan", "Ethan"]
```

0 1 2 3 4

You can find the length of an Array by using len()

```
names = ["Adam", "Bob", "Charlie", "Dan", "Ethan"]  
print(len(names))
```




5

Changing Elements in an Array

You can also change the value of an element by using its index position, like this.

```
new_name = "Fred"  
names[4] = new_name
```



```
['Adam', 'Bob', 'Charlie', 'Dan', 'Fred']
```

0 1 2 3 4

Adding to Array

When adding a new elements to an array, use `.append()`.

Syntax = `arrayname.append(element to be added)`

```
names = ["Adam", "Bob", "Charlie", "Dan", "Ethan"]  
  
new_name = "Fred"  
names.append(new_name)
```

`['Adam', 'Bob', 'Charlie', 'Dan', 'Ethan', 'Fred']`

Removing from Array

`.remove()` allows you to remove a specified element from the Array.

```
names = ["Adam", "Bob", "Charlie"]  
names.remove("Adam")
```

→ `['Bob', 'Charlie']`

`.pop()` allows you to remove a elements from an Array by referring to the index position.

```
names = ["Adam", "Bob", "Charlie"]  
names.pop(1)
```

→ `['Adam', 'Charlie']`

File Handling

File handling is the process of storing data in external files. Particularly in python, we can **open, write and close a file in our program.**

Creating a File

A file can be created by using this code below.

```
Names = newFile("names.txt")
```

Setting up a File

File handler to store the file in a variable.

Full file name as a string. This also includes the extension.

```
file = open("test.txt", "r")
```

Function to open file.

Flag to determine how the file is to be used.

File Handling Flags

"w" write	Writes data to a file, but overwrites data already stored on the file. This will create a new file if the file doesn't exist.
"a" append	Writes data to a file, but adds and/or changes data already in the file. This will create a new file if the file doesn't exist.
"r" read	Read contents of the file. File must be created before reading.

Writing to a File

```
file = open ("file_test.txt", "w")
file.write("London \n")
file.write("Paris \n")
file.close()
```

Use .write() to write data to a file.

"\n" is a whitespace character and adds a line break in your file. Just like if you were to press the enter key.

Reading a file

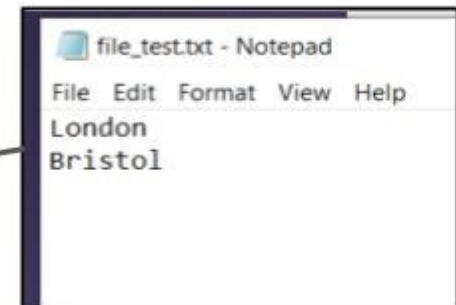
```
file = open ("file_test.txt", "r")
data = file.readlines()
file.close()

print(data)
```

```
['London\n', 'Bristol']
```

The output with \n to represent each line break.

The file to read from



SQL (Structured Query Language)

SQL stands for Structured Query Language

It is a language which allows you to create, query and add data to databases

SQL is used within most Database Management Systems, including MySQL, SQL Server and MS Access

SQL queries can be used within high level programming languages such as Python, Visual Basic or C#

SQL Syntax

The SQL syntax for querying a database is:

SELECT ... (list the fields to be displayed)

FROM ... (specify the table name)

WHERE ... (list the search criteria)

		Fields	Table name: Pupil Register			
Records	Student ID	Forename	Surname	Code	Course	Fee
	21562	Lewis	James	DG0011	Tennis	30.00
	36767	Michelle	Richards	CR0001	Kayaking	100.00
	38645	Frank	Jones	DG3002	Swimming	20.00

SELECT- The field/s you are selecting.

FROM- The name of the table.

WHERE- The condition of what to look for.

If you want to select all of the fields use

SELECT *

Example - Write a SQL statement that returns all pupils taking a course less than 100

```
SELECT Forename, Surname
FROM Pupil Register
WHERE Fee < 100
```

String Manipulation Character Position

Strings have their own index position, just like an array.

Index positions are placed inside square brackets.

Remember index position starts at 0.

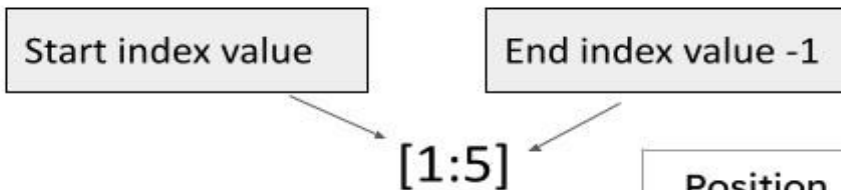
```
device = "Computer"
print(device[5]) #t
print(device[2]) #m
print(device[0]) #C
```

Position	0	1	2	3	4	5	6	7
Character	C	o	m	p	u	t	e	r

Slicing

We can also select parts of a string within a set range. We call this slicing.

Use a colon between the first and last index value (the last value is not returned).



```
device = "Computer"
print(device[3:5]) #pu
print(device[1:2]) #o
print(device[0:6]) #Comput
```

Position	0	1	2	3	4	5	6	7
Character	C	o	m	p	u	t	e	r

String Concatenation

We can combine multiple strings together to form one string. This is called Concatenation.

Use a + operator to concatenate (combine) strings together. This only works with strings.

```
first_name = "Homer"  
last_name = "Simpson"  
full_name = first_name + " " + last_name  
print(full_name)
```

Homer Simpson

This creates a space.

Length of String

We can also check the length of a string by using len()

Code

```
string = "This string contains forty-two characters."  
print(len(string))
```

Output

42