

String handling

1. Introduction to strings

One of the basic data types in programming is the '**string**'.

A **string variable** stores text rather than a number. The string is made up of a set of characters that can also include spaces, symbols and numbers.

Examples of strings

```
"a"  
"this is a string"  
"0145955"  
"this is a string with numbers 1223"  
"*$&ff^"((*ff&&*$$$"
```

Most programming languages have a number of handy functions to manipulate strings. These will be discussed in this section.

String handling

2. String assignment and storage.

Assignment

The first step in using strings is to get the program to recognise which variables are meant to be strings. This is done by putting the characters you want in the string in between a pair of speech marks, like this:

```
MyLargeString = "Hello this is a string"
```

Strings can have numbers inside of them, but the number is now handled as just another set of characters.

You can't carry out any calculations on numbers in a string without first converting it into a different data type, like an integer or floating point number. For example a telephone number can be stored in this way as no maths is likely to be carried out on it.

```
MyTelephoneNumber = "01926111111"
```

Strings can be input by the user and stored in a string variable.

For example the pseudocode :-

```
YourName = input('Please enter your first name')
```

This displays a prompt on the screen asking the user to enter a string. When they do, it is stored as the YourName variable.

Storage

Each character in a string is stored as a single byte. If there is more than one character in the string, the group is given a name and handled in the program as a single item.

For example

```
MyLargeString = "Hello this is a string"
```

MyLargeString is handled as one variable. It takes 22 bytes of memory to store all of the characters in this string (remember that blank spaces take up as much space as any other character).

String handling

3. String copy and concatenate

In addition to storing strings, programs can be written to do various things to the text inside them. You can add, remove, edit, or search the text using various commands. We will discuss several of these commands in the next few pages.

Copy

This function copies one string into another, over-writing any previous content. The exact command for doing this varies from language to language. For example in 'C' the command is strcpy(destination, source).

There is also an easier way to copy one string into another. Just use the assignment '=' operator, like this :-

```
MyString = SecondString
```

This pseudocode copies the content of SecondString into the MyString variable, replacing whatever was in MyString.

Concatenation

Concatenation means to link things together. You can do this with strings using the '+' operator.

Like this :-

```
Line 1: FirstName = input("Enter your first name")
Line 2: Surname = input("Enter your surname")
Line 3: Message = "Your name is " + FirstName + " " +
Surname
Line 4: print Message
```

The first two lines are inputs for the user to enter their first name and their surname.

Line 3 uses the concatenate operator to form a message and store it in Message. Notice the " " in the middle of it to insert a space between the first and surname.

The Message string is then displayed by Line 4.

String handling

4. String length and string traversing

String length

One of the basic properties of a string is its length i.e. the number of characters it contains. It can be useful to know the total number of characters, perhaps you might need to know the 4th and 6th character. Being able to identify the length of the string makes this possible.

Most programming languages make it simple to keep track of string length. In Python for example the function `LEN(stringname)` can be used.

In OCR pseudocode you should add `.length` to the string statement.

For example if you wanted to know how many characters (including symbols and spaces) are in this string:

```
MyString = "Hello"
```

You would use the `.length` extension as follows:

```
MyString = "Hello"  
print MyString.length
```

Since "Hello" is five characters long, this command would print out "5".

String traversal

The software is able to keep track of the position of every character within a string. This lets you locate or make changes to individual characters rather than changing the entire string. You just need to refer to the position of the character you want to look at or change.

This is called "string traversal".

For example

```
MyString = "Hello"  
FirstLetter = MyString(0)
```

FirstLetter now contains the string "H", since H is the zeroth character in the MyString string.

String handling

5. String search

Many languages provide functions to search a string for a specific character or a set of characters within that string (substring), perhaps a word or even a few words.

For example,

```
LongString = "It was the best of times"  
result = SearchString(LongString, "best")
```

SearchString searches the LongString for the substring 'best'. If it is found, it returns the position of the first character of the substring.

Python

In Python string searching is done using find(), like so:

```
MainString.find(findme, begin=0, end=len(MainString))
```

- The string to be searched is MainString
- The substring we are looking for is findme
- The begin sets the starting position of the search - usually zero
- And the end position by default is the length of the string to be searched.

'C' and PHP language

In 'C', and in the server language PHP, the function is called strstr()

String handling

6. Other common functions

Without going into any details, here are a list of more common string functions

- Convert string into all upper case

- Convert string into all lower case
- Split a string at a given position into two strings
- Trim a string so there are no start or trailing white spaces
- Remove all white space from a string
- Extend a string by duplicating itself (5 become 5555 etc)

File handling

1. Introduction to Files

A file is an object that is used to store data. Files can contain all sorts of data, from images to spreadsheets to executable programs.

Files are stored on secondary storage media, for example, hard disk, external hard drive, CD, flash memory stick etc.

Every file is given a name when it is created, and usually, an *extension*, telling the computer and the user what type of data the file contains.

For example

```
MyText.txt    (contains text data)
MyPic.jpg     (contains image data)
MyMusic.mp3   (contains music or sound data)
MyMovie.mpg   (contains video and audio data)
```

Files do not have to have extensions, technically you can still open them. However if you don't know the correct file format and try to open it with an incompatible application, then all you will see is garbage.

There are many other standard file types. Standard file types (such as mp3) can be used by hundreds of different applications because the file data format has been made public.

However it is also perfectly possible to create a custom file type with your own software program and give it a unique extension such as

```
MySpecialFile.mine
```

The software that created the MySpecialFile.mine will have formatted the data just the way it needs.

In the following pages, we will discuss how files can be opened, closed, read, and written to by a programmer.

File handling

2. Open File

A 'file open' command in a programming language effectively means 'prepare the file, ready for use'.

In Python, the command is:

```
Myfile = open ('filename', 'mode')
```

The open command has two arguments. The first is the name of the file, and the other is the 'mode' in which the file is to be opened.

File handles

The open command returns a 'file handle' - a temporary label used by the program itself to identify that file.

File handles are useful because file names can get very long and cumbersome. Rather than having to type out something like "Personal account sheets from 2005-2015 - Andy and Bob.xls" every time, you can assign a short file handle like 'Myfile' to a variable and use that throughout the program.

File modes

The second argument in that Python command is the 'mode'. You don't always want to give programs permission to do whatever they want with files. The mode tells the program what it is allowed to do with a file. These are the modes:

File open modes	
Mode	Description
'r'	Read Only. Allows data in the file to be read. You can only use Read Only mode on files that already exist - it can't create new files.

File open modes

Mode	Description
'w'	Write mode. If the file exists, all of the data currently inside of it is discarded and new data is written over the top of it. If the file does not exist, Write mode will create an empty file.
'a'	Append. Much like write, this allows data to be written into an existing file or into a new file. But where write mode writes over the top of whatever was there, append mode adds its changes to the end of the file. Existing data is kept intact.
'r+' or 'w+'	Allow both read from and write to the file. W+ mode will also blank existing files, while R+ does not.
'a+'	Open or create a file for reading and allow data to be written at the end of the file.

File handling

3. Close File

Once the program has finished using the file, it has to be closed, otherwise it can't be used by other applications.

In Python, you close files using the `close()` method

Look at this example of Python code:

```
filehandle = open('mytextfile.txt', 'r')  
filehandle.close()
```

First it opens a file called 'mytextfile.txt' in read mode.

As discussed on the previous page, it gives it the file handle "filehandle". Then it immediately closes the file.

In addition to allowing other programs to access the file, closing files is important because otherwise the file content may remain in RAM. Closing it frees up that memory to be used for something else.



File handling

4. Write to a file

The example Python code on the previous page simply opened and closed a file, without doing anything to it. This isn't very useful. You probably want your program to be able to do something like write data into the file between opening and closing it.

In order to write information into a file, the file has to be opened in the correct mode. Remember that read-only mode does not allow the program to write to the file.



Each language has its own commands for writing to a file. Take a look at the Python code below:

```
filehandle = open('mytextfile.txt', 'w')
filehandle.write('this is a test')
```

The first line of code shows a file called mytextfile.txt being opened in 'write' mode. If the file doesn't exist, the open command will create it.

If mytextfile.txt already exists, then all the data within it is discarded, ready for the program to write new data into it.

The second line of code is what writes new data into the file, using the file handle and the write() method.

If a number of lines needs to be inserted, then the command is just repeated. You will need to tell the program to add new lines using the newline character (\n), or it will just put everything onto the same line.

```
filehandle.write('Monday\n')
filehandle.write('Tuesday')
```

This results in a text file containing two lines of text.

Using append

The 'write' mode effectively creates an empty file by discarding all of the existing data within that file.

If the data needs to be kept, then don't use 'write'. Instead, open the file in 'append' mode. This will keep the existing data intact, and any new data is added to the end of the file.

File handling

5. Using a loop to write to a file

If there are lots of data to be added to the file, then a WHILE or a FOR loop can be used instead of just repeating the write command over and over.



Pseudocode example:

```
01 filehandle = open(MyTextFile.txt, 'a')
02 FOR i = 0 TO 199
03     filehandle.write("Some extra news \n")
04 NEXT i
05 filehandle.close()
```

Lets go through this example line by line

Line 1 - Opens a file called "MyTextFile.txt" in append mode. It is assigned the file handle "filehandle".

Line 2 - A FOR loop is set up to start at 0 and end at 199.

Line 3 - A new line is added to the end of the file. The line will contain the text "Some extra news".

Line 4 - Loops back to repeat the action until the FOR loop reaches its end at i = 199

Line 5 - Closes the file

This program will create a file with 200 lines of repeating text.

If you tried to do this without some kind of loop, your program would have to have one line for every line you added, which isn't very efficient. In fact, you would probably find it faster to just write the file by hand. But by using a loop, you can write large amounts of text quickly and easily.

File handling

6. Read File

If you want to read data from a file, you have to open the file in a mode that gives your program permission to read from it. "r" or "r+" modes would work.

When a program is told to read all data from a file in one go, it converts the entire contents of that file into a single string or array variable. The program can then search that string or variable to find the position of the data it wants. An alternative is to first of all find a specific position in the file, then read some bytes from there - this is called 'file seek'.

Each programming language has its own commands for reading a file.

In Python the `read()` method is used to read in the entire file into one string variable. Like this:

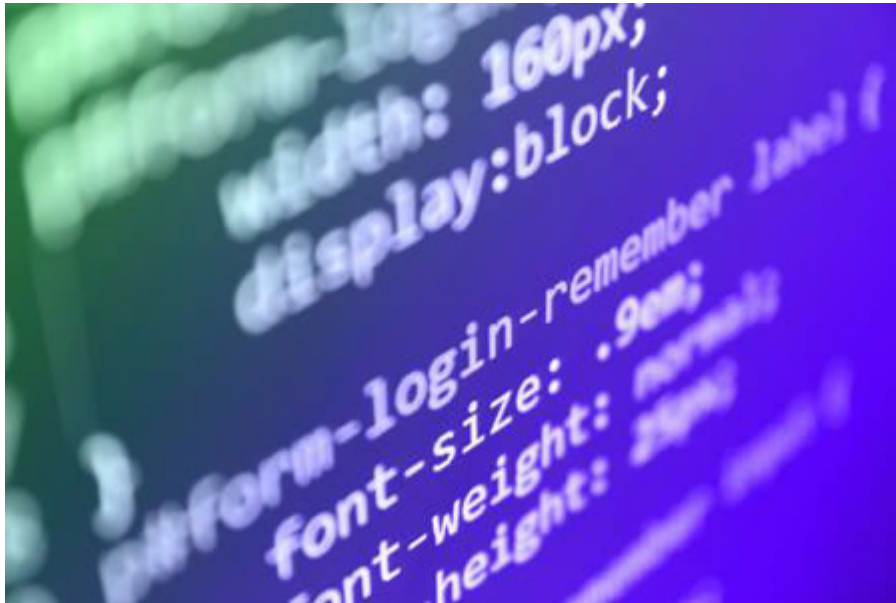
```
filehandle = open('mytext.txt', 'r')
MyString = filehandle.read()
```

This reads the entire file into the variable string `MyString`. There has to be enough computer memory available to do so, or an error will occur.

You can use the same Python method to just read *part of* a file, by telling the program how many characters from the file you would like it to enter. Like this:

```
fh = open('mytext.txt', 'r')
MyString = fh.read(1000)
```

This will enter the first 1000 characters from the file rather than the entire file.



Data format

The data within a file is rarely just a random collection of bytes - instead, the data has a structure to it. For example, the image data in a JPG file has a standard, published, format.

In order to read the data correctly, the software code has to take account of the format.

File handling

7. Using a loop to read a file

An iteration loop (FOR or WHILE) can be used to read data from a file. Either the entire data can be read in or perhaps only some of it.

Let's consider a text file that contains lines of text, each of which ends with newline. So the data inside the file looks like this:

```
This is the first line\nThis is the second line\nThis is the third line
```

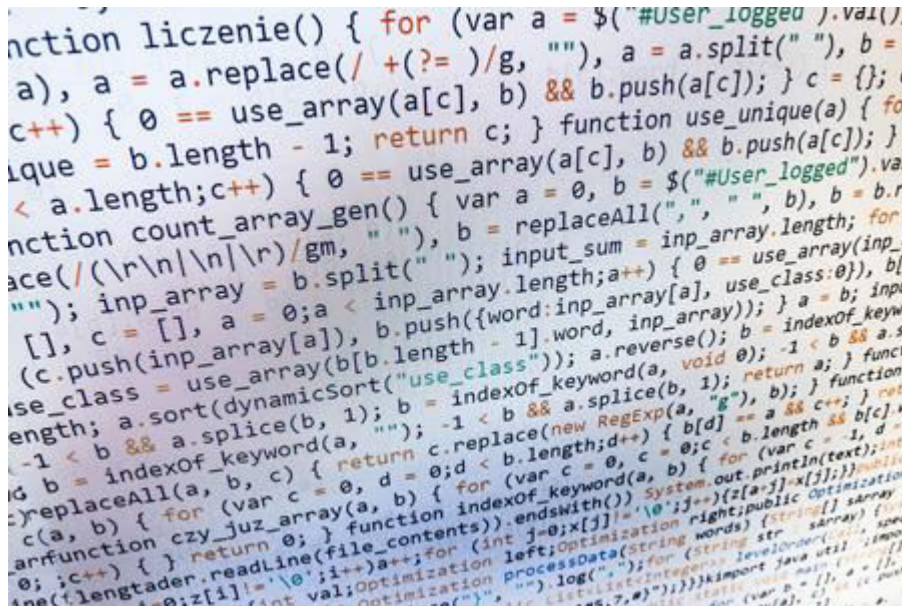
This time we will use Python commands to read and display each line.

```
01 filehandle = open(MyTextFile.txt, 'r')\n\n02 for line in filehandle\n\n03     print(line)
```

Let's go through each line

- Line 1 - open a file called MyTextFile.txt in read only mode. It is assigned the file handle "filehandle"
- Line 2 - The FOR command loops over every line in the file until the end of the file is reached
- Line 3 - As each line is read in, it is printed

Python is unusual in that it does not use the NEXT command in a for loop.



To do the same thing as pseudocode, it can be set out as follows

```
01 fh = open("MyTextFile.txt", read-only)
02 WHILE data_item != End_of_file
03     data_item = fh.read()
04     print data_item
05 END WHILE
```

Let's go through each line

- Line 1 - open the file in read-only mode and return a file handle fh
- Line 2 - Start a WHILE loop that will end once the end-of-file is detected.
- Line 3 - read in a line
- Line 4 - display or print the line just read in
- Line 5 - loop around again

File handling

8. File handling summary

- A file is an object to store data
- Files are stored in many different types of storage media, including hard disk and dvd
- A file can be opened in a number of different modes - read-only, write and append
- Data can be read from the file byte-by-byte
- If the data is formatted, then the code can make use of this to read in the data correctly
- Data can be written into a file
- It depends on the file open mode as to whether the original data it contained is discarded or added to at the end of the file
- Every open file needs to be closed properly by the software



Structured Query Language (SQL)

1. Introduction

Data, such as a list of names, can be stored in a straightforward text file. But it has some major limitations in terms of searching and sorting the data.

An alternative to storing data directly in a file is to store it in a **database**.

A database is designed specifically to handle, sort, search and manipulate organised data.

A **database** is a collection of data or information which is held together in an organised or logical way.

Because of the high organisation of a database, data can be retrieved, sorted and updated very efficiently.



Adding and retrieving information in a database is done using a special-purpose programming language called SQL.

The basic actions carried out on a database are

- Insert
- Update
- Delete
- Retrieve
- Sort

The rest of this section will describe the parts that make up a database and how SQL is used to manipulate its data.

Structured Query Language (SQL)

2. Parts of a database

A database is made up of one or more tables.

Like this:

First Name	Last Name	Address	City	Age
Mickey	Mouse	123 Fantasy Way	Anaheim	75
Bat	Man	321 Cavern Avenue	Gotham	56
Wonder	Woman	987 Truth Way	Paradise	43
Donald	Duck	555 Quack Rd	Mallard	68
Arnold	Mouse	333 Star Street	Anaheim	34

Each table should be given a relevant name. The table above is named 'Characters' because it contains well known cartoon or comic characters.

First Name	Last Name	Address	City	Age
Mickey	Mouse	123 Fantasy Way	Anaheim	75
Bat	Man	321 Cavern Avenue	Gotham	56
Wonder	Woman	987 Truth Way	Paradise	43
Donald	Duck	555 Quack Rd	Mallard	68

Each column in a table is called a **field** and contains information about a particular item of data. Each field should be given a relevant name, for instance the field called 'city' contains a city name. A field is always set to a certain data type such as

- numeric types: integer, float, real
- boolean
- character
- string
- Date and time
- binary
- Other e.g. blob

Each row in a table is called a **record**. Here is a single record from the table above

Wonder	Woman	987 Truth Way	Paradise	43
--------	-------	---------------	----------	----

A record is made up of collection of fields related to a single item.

In short:

- Each piece of information is entered into a 'field'

- Related fields are organised into records
- Related records are placed into tables
- Related tables are collected into a database

Structured Query Language (SQL)

3. SQL

Short for 'Structured Query Language'.

SQL was created for use with an SQL database. It consists of a set of commands that can carry out the following tasks

- Search records
- Update records
- Insert records
- Delete records
- Find related records

The table below shows the SQL statements you need to understand

SQL commands

SELECT	This retrieves a record or a set of records. Other commands can then be carried out on that record set without affecting the whole database.
FROM	The identifies the table where the data comes from
WHERE	This sets out some condition with which to choose only certain records

These will de discussed over the next few pages.

Structured Query Language (SQL)

4. Using SELECT

This is a command to retrieve records from a database table. It doesn't do anything to the record itself. But it is very useful because by selecting a record, other commands can be carried out on just the selected records without affecting the rest of the database.

Let's work with this table:

First Name	Last Name	Address	City	Age
Mickey	Mouse	123 Fantasy Way	Anaheim	75
Bat	Man	321 Cavern Avenue	Gotham	56
Wonder	Woman	987 Truth Way	Paradise	43
Donald	Duck	555 Quack Rd	Mallard	68
Arnold	Mouse	333 Star Street	Anaheim	34

If we wanted to retrieve every record from this table, the SELECT command is:-

```
SELECT * FROM Characters
```

The asterisk * is called a 'wildcard' and it means 'all records' or 'all fields'. The FROM is followed by the name of the table.

If we wanted to fetch all fields from within a specific record then the wildcard is useful for that as well, for example if we wanted all fields from the Donald Duck record the SELECT statement is

```
SELECT * FROM Characters WHERE First Name = "Donald"
```

The SELECT command is also used to fetch specific fields from within a record set. For example

```
SELECT Address, City FROM Characters WHERE First Name = "Donald"
```

Structured Query Language (SQL)

5. Using SELECT WHERE

This is a command to select only certain records from a database table.

Table name: Characters

First Name	Last Name	Address	City	Age
Mickey	Mouse	123 Fantasy Way	Anaheim	75
Bat	Man	321 Cavern Avenue	Gotham	56
Wonder	Woman	987 Truth Way	Paradise	43
Donald	Duck	555 Quack Rd	Mallard	68
Arnold	Mouse	333 Star Street	Anaheim	34

The general format of the SELECT ... WHERE command is :-

SELECT fields FROM table WHERE Criteria

Let's use an example to explain this.

	Fields (columns)		Table		Criteria
SELECT	FirstName, LastName	FROM	Characters	WHERE	City = 'Anaheim'

Or, without the labels:

SELECT FirstName, LastName FROM Characters WHERE City = 'Anaheim'

FirstName and LastName are fields (columns) within the table Characters.

This SQL command will return the first and last names of anyone in the table whose city is Anaheim. In this case, there are two matches:

- Mickey Mouse
- Arnold Mouse

The standard set of arithmetic comparators can be used in the criteria (=, >, <., >=, <=, !=). For example Age < 60 would get all records with an age of less than 60.

Structured Query Language (SQL)

6. Summary

- Databases are able to store highly organised data as one or more data tables
- Each piece of information is entered into a 'field'
- Related fields are collected into records
- Related records are collected into tables

- Related tables are collected into a database
- Databases can be searched and modified using a specialised programming language called SQL
- Common SQL commands for searching include SELECT, WHERE
- SELECT retrieves records
- WHERE limits retrieval to those records that match a set of criteria
- The standard comparator operators can be used so set up the criteria

Subroutines and functions

1. Sub-procedures and functions

Most computer programs are fairly complicated in the work they need to do. And so it is very important that the code is written in a well structured manner.

One of the key tools for producing neat, efficient, structured code is to make use of sub-procedures and functions.

This section will explain what these are and how to use them.

Spaghetti code

Code that is very poorly structured and jumps all over the place in a tangled manner is called 'spaghetti code' and is usually a nightmare to update and maintain. So avoid spaghetti code by following good coding practice as will be described in this section.

Subroutines and functions

2. Spotting repeating code

Look at the block of code below

```

FirstName = "Joe"
SecondName = "Smith"
FullName = FirstName + SecondName
Print FullName

```

The first two lines declare the first and second name of a person held as variables. The third line then concatenates the names and the fourth line prints out the full name. All well and good. It is a very efficient piece of coding.

Now let's consider that three people need to have their names printed out. The pseudocode below shows how this may be written

```
FirstName = "Joe"
SecondName = "Smith"
FullName = FirstName + SecondName
Print FullName
```

```
FirstName = "Louise"
SecondName = "Brown"
FullName = FirstName + SecondName
Print FullName
```

```
FirstName = "Mandy"
SecondName = "Jones"
FullName = FirstName + SecondName
Print FullName
```

This will work perfectly well. But do you notice how often the same two lines code are repeated i.e. the FullName and Print statements?

This is inefficient and it also makes it more likely that a programmer will introduce a mistake into the code.

Carrying out the same task in different parts of the program is so common, that a special way of avoiding it has been developed. It is called a subprocedure or subroutine as explained on the next page.

Subroutines and functions

3. Building up a sub-procedure

The previous page has this block of code;

```
FirstName = "Joe"
SecondName = "Smith"
FullName = FirstName + SecondName
Print FullName
```

```
FirstName = "Lousie"
```

```

SecondName = "Brown"
FullName = FirstName + SecondName
Print FullName

FirstName = "Mandy"
SecondName = "Jones"
FullName = FirstName + SecondName
Print FullName

```

The code in red is just doing the same thing, over and over. In order to create a sub-procedure, take this block of code and enclose it in a procedure statement, like this

```

procedure
  FullName = FirstName + SecondName
  Print FullName
end procedure

```

This block of code needs to be given a sensible name in order to use it - let's call it `print_name` as shown below

```

procedure print_name
  FullName = FirstName + SecondName
  Print FullName
end procedure

```

We are still not quite there yet, because the code is using two variables `FirstName` and `SecondName` but these need to be defined before they can be used. So let's supply that information to the procedure. Like this

```

procedure print_name (FirstName, SecondName)
  FullName = FirstName + SecondName
  Print FullName
end procedure

```

We can describe a subroutine as:

A **subroutine** is a block of code intended to be used a number of times within a program. The code is packaged as a single, named, unit.

The next page will show how this new procedure is used.

Subroutines and functions

4. Using a sub-procedure

The procedure we have developed on the previous page is shown below

```

procedure print_name (FirstName, SecondName)
  FullName = FirstName + SecondName
  Print FullName
end procedure

```

The code below shows how it can now be used

```
FirstName = "Joe"
SecondName = "Smith"
print_name(FirstName, SecondName)

FirstName = "Louise"
SecondName = "Brown"
print_name(FirstName, SecondName)

FirstName = "Mandy"
SecondName = "Jones"
print_name(FirstName, SecondName)
```

The example procedure is only two lines long, so the reduction in lines of code has not reduced that much. However if it was a complicated sub-procedure made up of dozens of lines, then it become far more efficient to use a procedure rather than repeating code.

The action of accessing a subprocedure is called '**calling**' a subprocedure. When it is called, the code within it runs and then the program returns to the next line of code in the main program.

Another advantage of using a procedure is reliability. The block of code only needs to be tested in that one place in the program. Once it is proven to work, the programmer can be confident that it will work no matter where it is called from. This also saves a lot of programming time.

Another common name for a sub-procedure is **subroutine**.

Advantages of using a sub-procedure

- More efficient program
- Improved reliability
- Saves total programming time

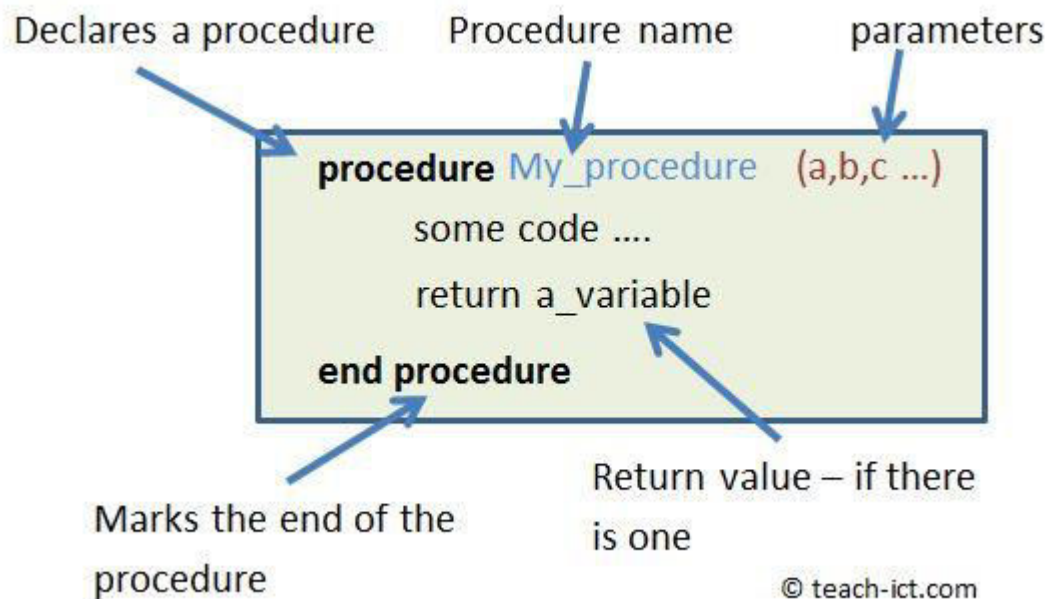
Subroutines and functions

5. Function returns a value

The example we used was supplied with two values, known as the **parameters** of the subprocedure. There can be zero, one or any number of parameters passed into a subprocedure. For example

```
hello_world()      - a subprocedure with no parameters
hello_world (Name) - a subprocedure with one parameter required
```

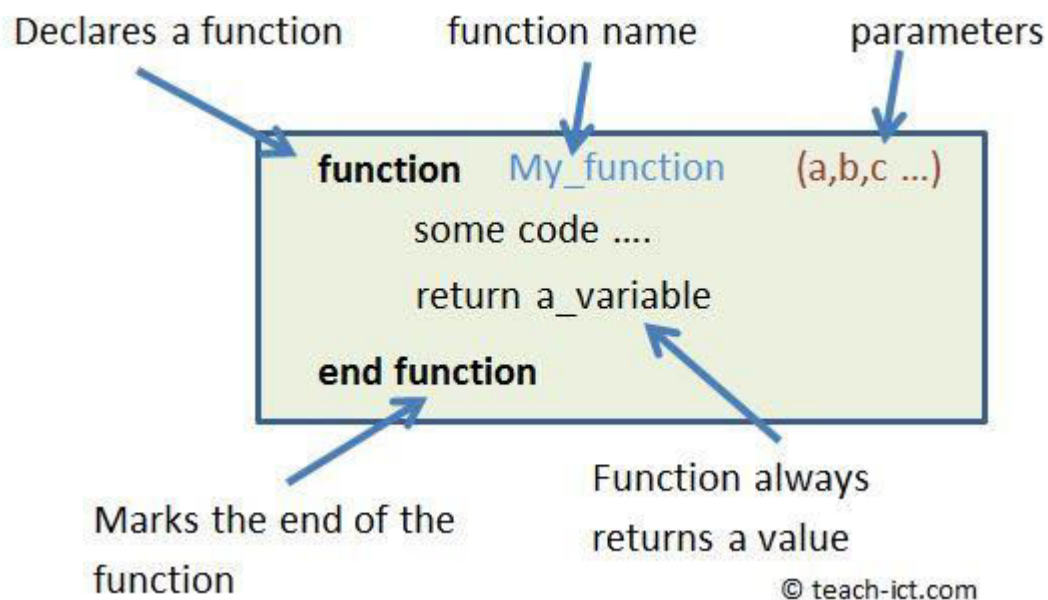
The picture below shows the general parts of a subprocedure



Function and Subprocedure difference

The difference between a function and a subprocedure is that a function always returns a single result whereas a subprocedure may return none at all or many values.

The picture below shows the general parts of a function



Let's think about a function that calculates the average of three numbers.

The function is given three numbers as parameters and returns the answer.
For example

```
function average (number1, number2, number3)

    average_number = (number1 + number2 + number3)/3

    return average_number

end function
```

The calling program stores the returned value of the function in a variable, so it can be used at some other point in the program

```
myaverage = average(10,30,40)
```

Functions are very useful for making specific calculations that can then be used wherever the programmer needs to do that calculation.

Subroutines and functions

6. Libraries

Functions and subprocedures are able to carry out a specific task and they are called from the main program.

What this means is that it is perfectly possible to put all of these functions and subroutines into a single package called a **code library**, which can then be used by other programmers.

There are many code libraries available for sale, which a programmer can purchase.

They do so because

- It saves programming time
- A commercial library is reliable
- The functions and procedures have been thoroughly tested
- They have been written by experts in their speciality
- They are easy to use

For example, let's say you are writing a computer game. And one of the things you want to do is to fully shade and render a character from a wire-frame model. The coding for this is very complicated and needs an expert to write it. Unless, as a programmer, you really want to spend months becoming an expert in rendering, the best solution is to buy in a graphics code library. Then add it to your program and call the required function.

For example, imagine a graphics library called `Wonderful_Graphics_Library`. In that library there is a function called `render_character` that does the job you want it to do. The pseudocode below shows how it may be used

```
include Wonderful_Graphics_Library  
  
my_hero = render_character(wireframe_model)
```

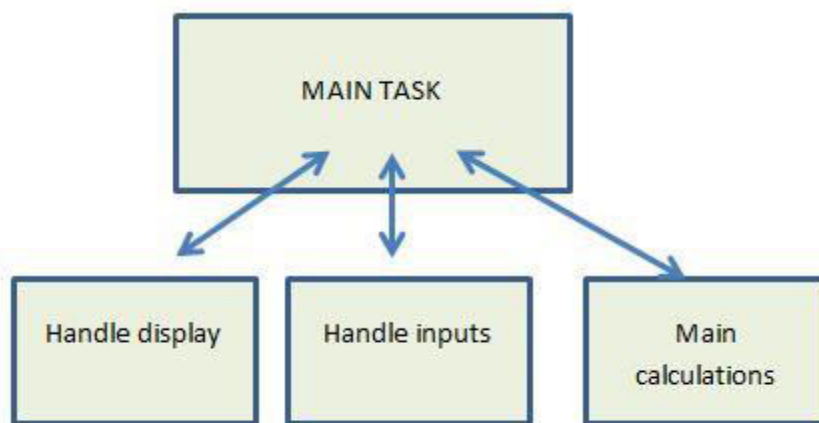
Subroutines and functions

7. Organising code

One of the skills in computer programming is to be able to break down a problem / task into a number of smaller tasks. This is called 'computational thinking' and **we have a section on it here**

Once the sub-tasks have been identified, a very good way of converting those sub-tasks into real code is to assign a subprocedure for each one of them.

For example, imagine that a large job has been divided into a number of smaller tasks are shown below



© teach-ict.com

Perhaps the 'Handle display' part of the task could be given over to a subprocedure. Another could deal with the 'handle inputs' task and yet another one coded up for doing the main calculations.

Advantages of using subprocedures

When code is structured in this way it offers some real advantages

- It is simpler for others to read the code and understand what it is doing

- It is simpler for yourself to understand your own code if you have been away from it for months or years
- It is easier to test the individual parts of the program
- It is easier to allocate parts to different members of the team
- It is easier to maintain and update the code in the future
- It is easier to re-use some of the code for other projects

Subroutines and functions

8. Summary

- A poorly structured program is called 'spaghetti code' and as a programmer it should not be written that way.
- Functions and subprocedures help form well structured, efficient code
- Using subprocedures avoids repeating the same code in many places in the program
- A function returns a single value
- The data passed into a subprocedure are called parameters
- Another name for subprocedures is subroutine
- A complicated task can usually be broken down into a simpler set of sub-tasks. Subprocedures can be used to code these smaller tasks

Random numbers

1. Introduction

Random numbers are widely used in computer programming and engineering. So it is important that you know how to create them in the computer language of your choice.

Here are some examples that use random numbers

- The amount of damage that a character hits you in a computer game
- Picking a lottery ticket (we shall use this in this section)
- Creating a random set of values for testing code

- 'Monte Carlo' testing. Where a simulation is run with a random starting conditions each time - weather forecasters use this to see how accurate their forecast is likely to be as the more consistent the result, then the more likely the simulation is accurate.
- Encryption - creating massive random keys to scramble data
- Creating variety and surprise in game programming

We are going to use Python 3 to show how random numbers are produced. A number of Python demo files are included in page 10 so you can run the code on your own computer.

In your exam, the OCR exam reference language format will use this format to form a random number:

```
random( ..., ....)
```

for example, to return a random number in the set 1 to 6, the command is

```
random (1,6)
```

Random numbers

2. Python

Python is a very popular, free, programming language. It comes with a number of useful modules for various tasks. And one of those modules offers the creation of random numbers.

Therefore we shall use Python to demonstrate how random numbers are handled in a real programming situation.

The code we demonstrate over the following pages is written using Python 3. If you want to follow along, please ensure that you have it installed on your computer. It can be found at: <https://www.python.org/>

```
# Python 3: List comprehensions
>>> fruits = ['Banana', 'Apple', 'Lime']
>>> loud_fruits = [fruit.upper() for fruit in
fruits]
>>> print(loud_fruits)
['BANANA', 'APPLE', 'LIME']

# List and the enumerate function
>>> list(enumerate(fruits))
[(0, 'Banana'), (1, 'Apple'), (2, 'Lime')]
```

We have also chosen to use a free Python IDE called 'Thonny'.

You can download the installation at <https://thonny.org>. The installation comes with Python 3 built in, so no need to download from python.org if this is used.

Random numbers

3. Import Random

A Python module contains a set of functions and procedures.

A large collection of modules are available as part of the standard Python installation. We will be using the 'random' module, which as its name suggests, is used for creating random numbers.

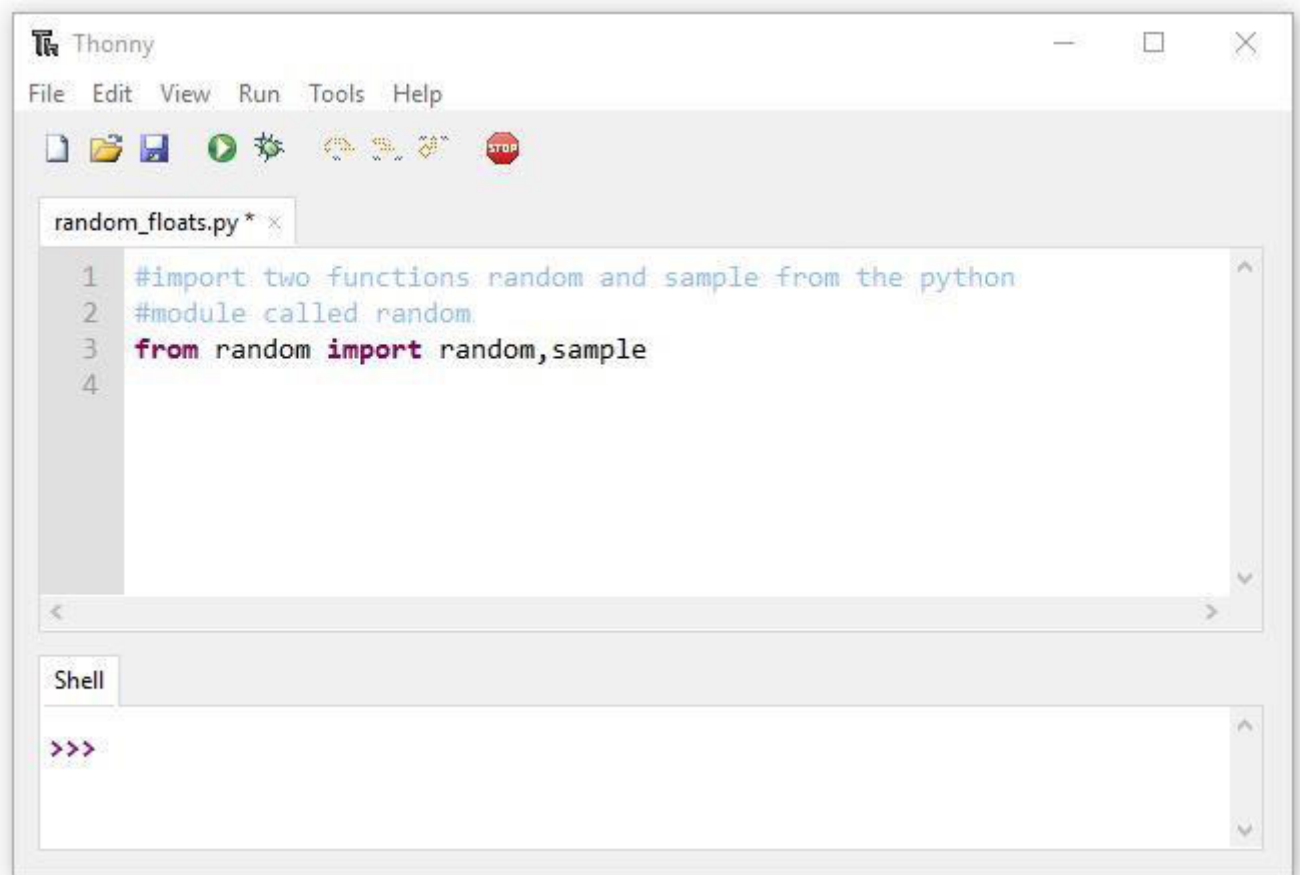
You can import the entire random module by typing:

```
import random
```

However, we will be using just two of the functions from this module, `random()` and `sample()`. Type the following:

```
from random import random, sample
```

The file is shown loaded into our IDE below (lines starting with # are comments)



Random numbers

4. Using random()

The `random()` function returns a random floating point "real" number between 0.0 and less than 1.0

We declare a variable called 'num' and assign the returned random number to it. In Python, this is done using the assignment operator = :

```
num = random()
```

Then this number is printed out, along with a description.

In pseudocode, this would use the keyword OUTPUT. The Python equivalent is "print":

```
print('Random floats 0.0 to 1.0: ', num)
```

The file is run in the IDE and the Shell shows the resulting floating point number. Every time the script runs, a different random number will be produced.



The screenshot shows the Thonny Python IDE interface. The top menu bar includes File, Edit, View, Run, Tools, and Help. Below the menu is a toolbar with icons for file operations and execution. The main editor window displays a Python script named `random_floats.py` with the following code:

```
1 #import two functions random and sample from the python
2 #module called random
3 from random import random,sample
4 # The random() function returns a float between 0.0 and 1.0
5 num = random()
6 print('Random floats 0.0 to 1.0: ',num)
7
```

Below the editor is a Shell window showing the execution of the script:

```
>>> %Run random_floats.py
Random floats 0.0 to 1.0: 0.18175777786563774
>>>
```

Random numbers

5. Random Integer

We created a random floating point number in the previous step. Now we will be more specific. Rather than just any random number, let's create a random **integer** between 0 and 9. Remember, integers are whole numbers, without decimal points.

Create a new file called `random_integers.py` and include the import command as before.

`random()` generates a random number between 0.0 and 1.0. So the first step to getting an integer between 1 and less than 10 is to multiply our generated random number by 10.

```
random()*10
```

At this point, it's still a floating point number. It might have some data after the decimal point. So we need to convert it to an integer. You can read more about converting, or "casting" one data type to another in our data types

section. In python, you can convert numbers into integers using the `int()` function.

```
int( random()*10 )
```

Since `int()` rounds down, we now have a random integer between 0 and 9. If we want it to be between 1 and 10 instead, we need to add 1 to the result:

```
int( random()*10 ) + 1
```

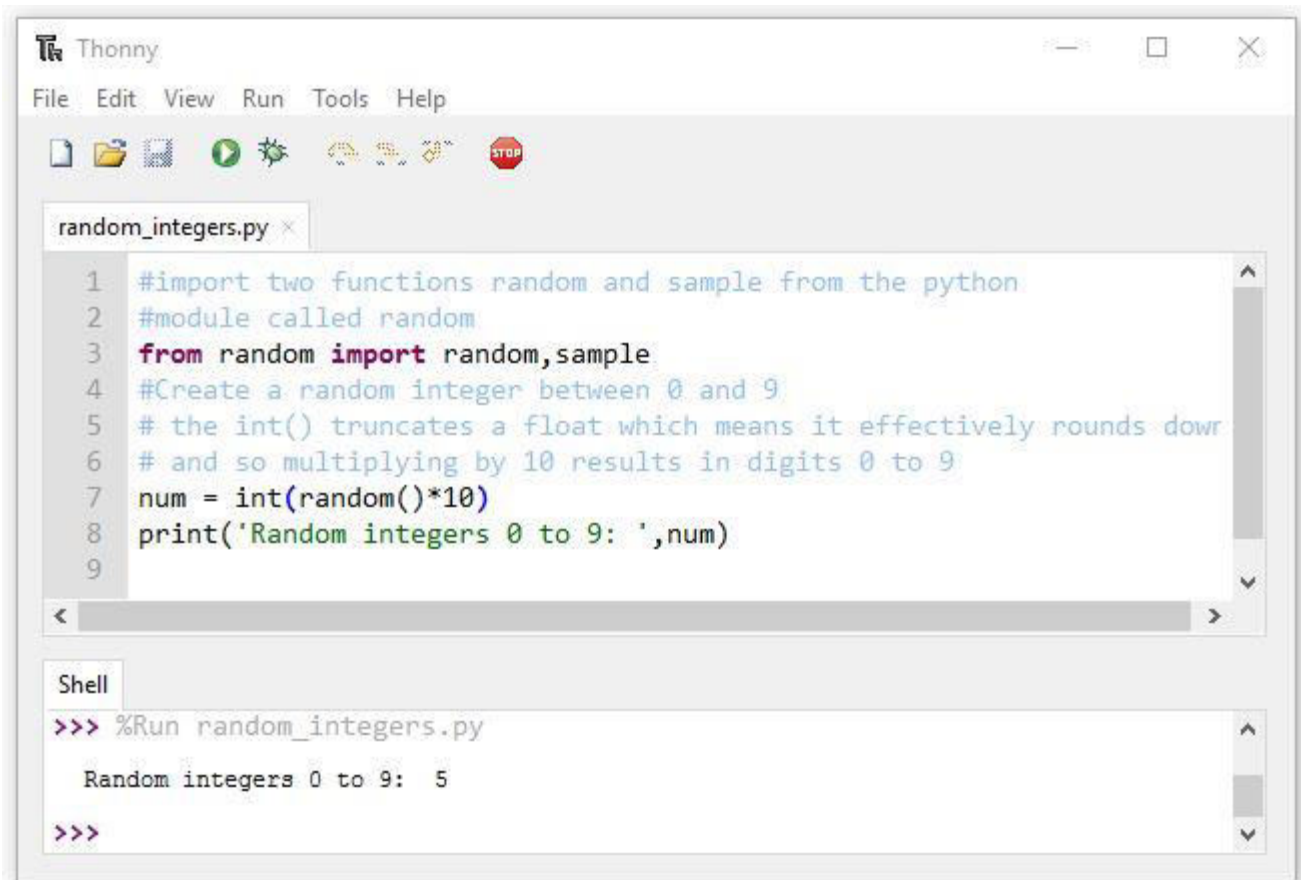
Now we have a random integer between 1 and ten. Time to assign it to a variable, so that the program can keep track of it. Let's use "num" again.

```
num = int(random()*10)
```

And the 'print' command displays it:

```
print (num)
```

The resulting script is run in the IDE and the Shell now shows a random integer value (5) being produced. This will be different each time the script is run.



The screenshot shows the Thonny Python IDE interface. The main editor window displays a Python script named `random_integers.py` with the following code:

```
1 #import two functions random and sample from the python
2 #module called random
3 from random import random,sample
4 #Create a random integer between 0 and 9
5 # the int() truncates a float which means it effectively rounds down
6 # and so multiplying by 10 results in digits 0 to 9
7 num = int(random()*10)
8 print('Random integers 0 to 9: ',num)
9
```

Below the editor is the Shell window, which shows the execution of the script:

```
>>> %Run random_integers.py
Random integers 0 to 9:  5
>>>
```


Random numbers

6. Random range of numbers

Task 3a

The previous tasks were concerned with producing a single random number. Our goal in this section is to generate a lottery ticket, which needs six numbers. So we need to write a program that generates six random numbers in one go.

Create a new python file called random_list.py

Task 3b

When you want to perform the same task multiple times, always think about using loops. It saves a lot of time. You can read more about loops [here](#). And since we want to store six related numbers, we'll keep them in an array. You can read more about arrays [here](#).

The loop needs a counter so that it can iterate the correct number of times and no more, we shall give the counter a single letter name:

```
i = 0
```

Next, the python empty list needs to be declared.

```
i = 0  
myarr = []
```

An initial message is displayed before the loop begins:

```
i = 0  
myarr = []  
print('Six random integers 1 to 10: ')
```

which helps the user understand the purpose of the output.

We're going to use a WHILE loop, though other types of loop would work just as well. It will need to loop 6 times, so the next line will be:

```
i = 0  
myarr = []  
print('Six random integers 1 to 10: ')  
while i<6:
```

We worked out the code we'd need in order to produce a random integer between 1 and 10 on the previous page, so it can just be repeated here:

```
i = 0
```

```
myarr = []
print('Six random integers 1 to 10: ')
while i<6:
    num=int(random()*10)+1
```

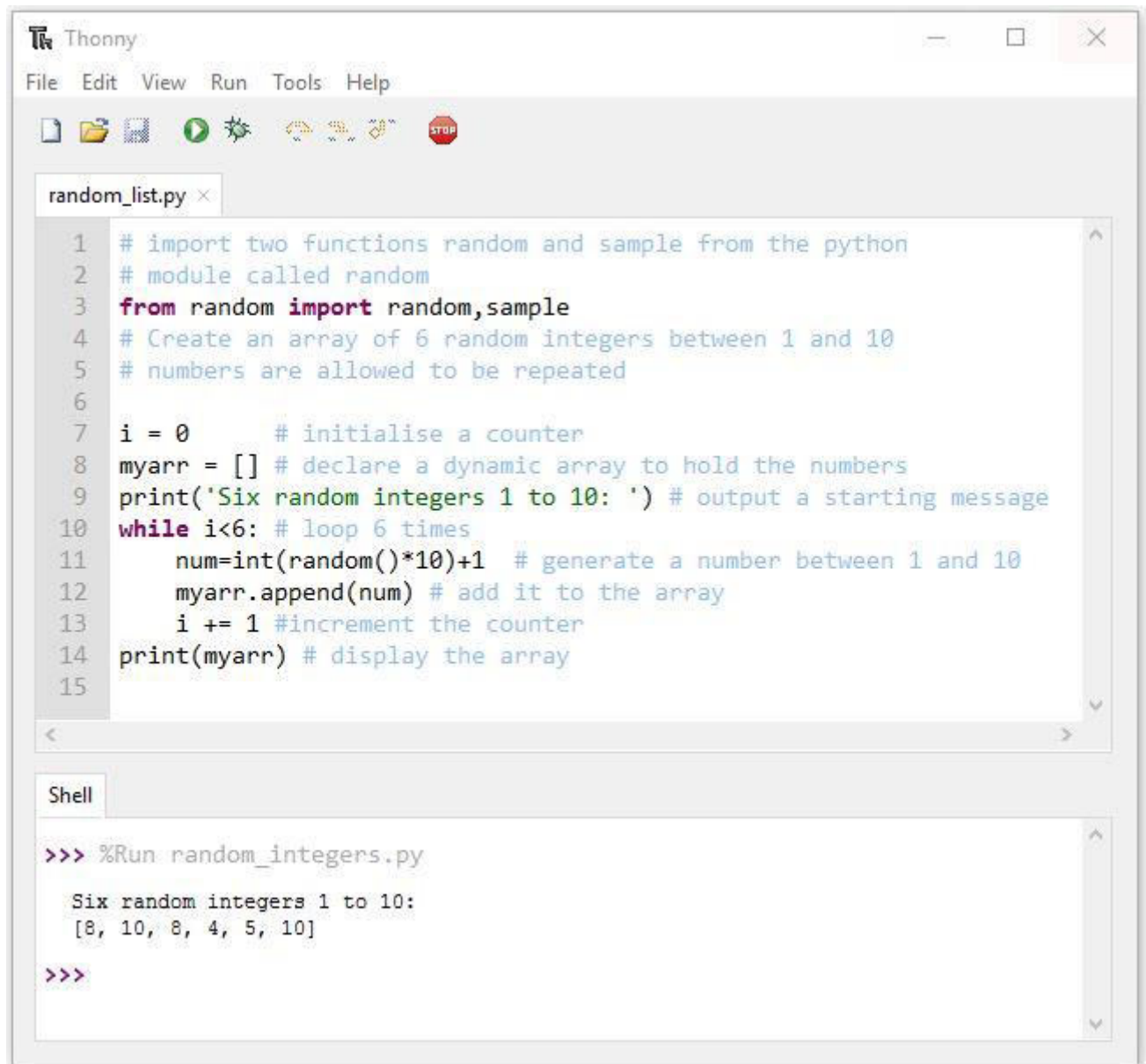
We want to store each random integer in the **myarr** array. This is done by 'appending' it to what's already there. Append has its own built-in command in python:

```
i = 0
myarr = []
print('Six random integers 1 to 10: ')
while i<6:
    num=int(random()*10)+1
    myarr.append(num)
```

Next the counter needs to be incremented each time the loop goes around.

```
i = 0
myarr = []
print('Six random integers 1 to 10: ')
while i<6:
    num=int(random()*10)+1
    myarr.append(num)
    i = i + 1
```

Then once the loop is complete, the result is displayed using "print". The complete code is shown below, and the Shell below that shows the output of the program.



The screenshot shows the Thonny Python IDE interface. The top menu bar includes File, Edit, View, Run, Tools, and Help. Below the menu is a toolbar with icons for file operations and execution. The main editor window displays a Python script named `random_list.py`. The script imports the `random` module and uses a `while` loop to generate six random integers between 1 and 10, storing them in a list named `myarr`. The output of the script is shown in the Shell window at the bottom.

```
1 # import two functions random and sample from the python
2 # module called random
3 from random import random,sample
4 # Create an array of 6 random integers between 1 and 10
5 # numbers are allowed to be repeated
6
7 i = 0      # initialise a counter
8 myarr = [] # declare a dynamic array to hold the numbers
9 print('Six random integers 1 to 10: ') # output a starting message
10 while i<6: # loop 6 times
11     num=int(random()*10)+1 # generate a number between 1 and 10
12     myarr.append(num) # add it to the array
13     i += 1 #increment the counter
14 print(myarr) # display the array
15
```

Shell

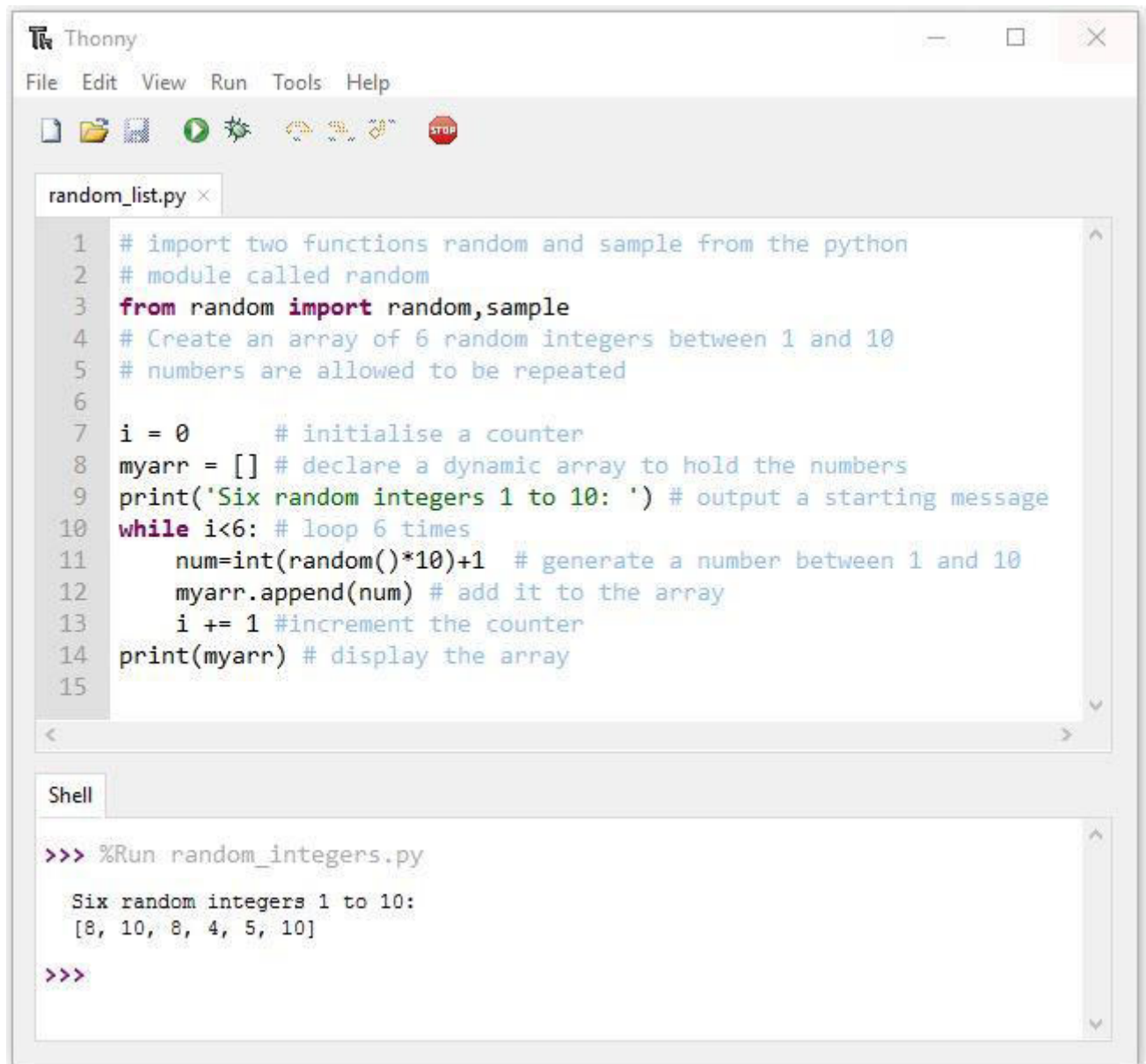
```
>>> %Run random_integers.py
Six random integers 1 to 10:
[8, 10, 8, 4, 5, 10]
>>>
```

Random numbers

7. Random range of unique numbers

Task 3c

You may have noticed that the output of the program on the previous screen included duplicate random numbers - "10" appeared twice, and so did "8".



The screenshot shows the Thonny Python IDE interface. The main editor window displays a Python script named `random_list.py`. The script imports the `random` module and the `sample` function. It then creates a list `myarr` and uses a `while` loop to generate six random integers between 1 and 10, appending them to the list. The output of the script is displayed in the Shell window below the editor.

```
1 # import two functions random and sample from the python
2 # module called random
3 from random import random,sample
4 # Create an array of 6 random integers between 1 and 10
5 # numbers are allowed to be repeated
6
7 i = 0      # initialise a counter
8 myarr = [] # declare a dynamic array to hold the numbers
9 print('Six random integers 1 to 10: ') # output a starting message
10 while i<6: # loop 6 times
11     num=int(random()*10)+1 # generate a number between 1 and 10
12     myarr.append(num) # add it to the array
13     i += 1 #increment the counter
14 print(myarr) # display the array
15
```

Shell

```
>>> %Run random_integers.py
Six random integers 1 to 10:
[8, 10, 8, 4, 5, 10]
>>>
```

That wouldn't happen on an actual lottery ticket, so let's fix that!

Make a new file called `random_unique_list.py` and include the import command as before.

For this we will be using the the second imported function, called `sample()`.

Task 3d

`Sample()` requires two arguments. The first is the complete list of allowed numbers (called the 'population') and the second is how many numbers you want to pick from the list (the sample). In our case, since we're picking six random numbers between 1 and 10, we want the population to be 1-10, and our sample to be 6.

So how do you enter "1-10" as an argument in a function?

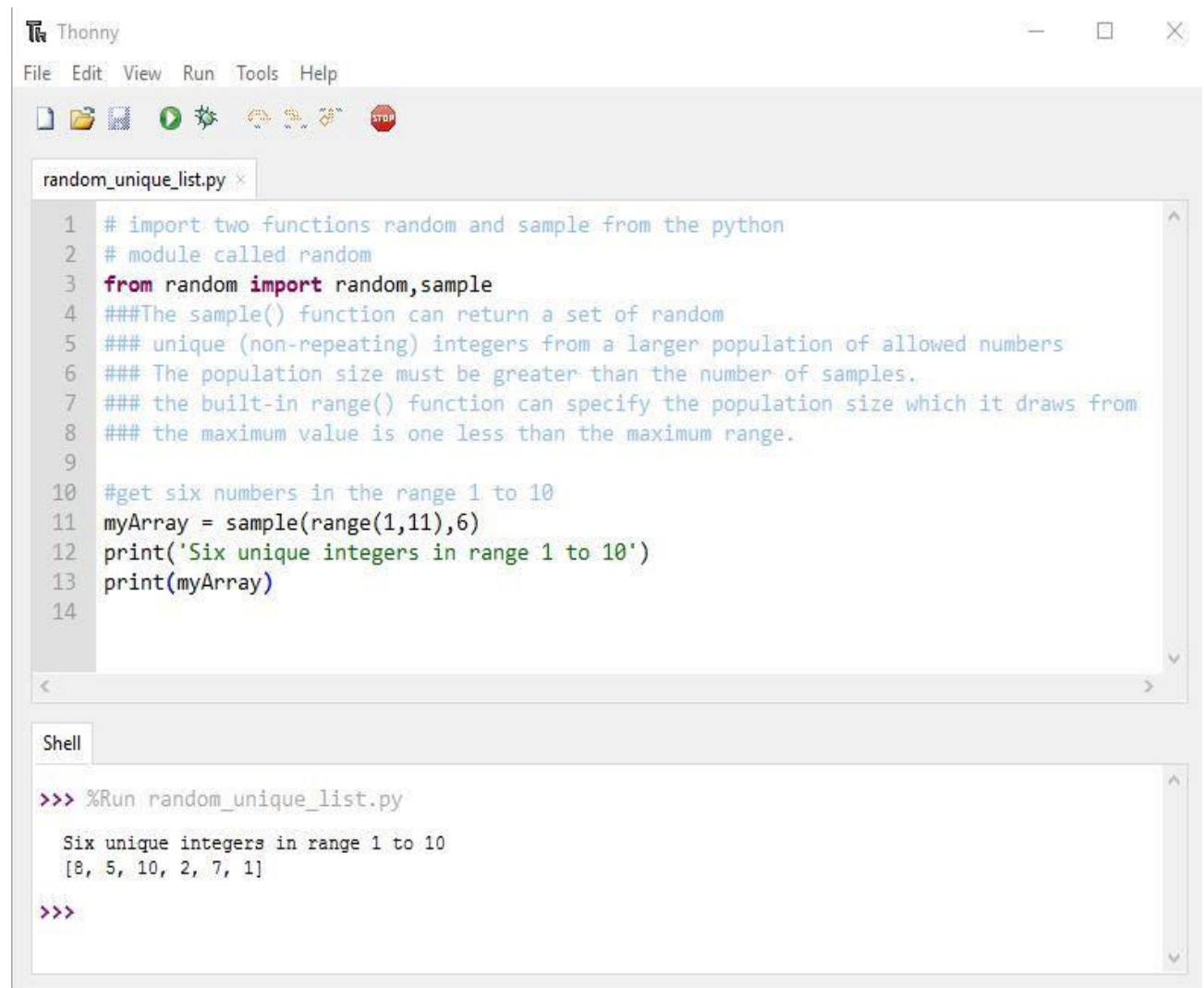
The answer is to use **range()**. Range takes two numbers as arguments, and lists every number between them. Remember, though, that many languages including Python begin counting from 0 rather than 1. So to python, 10 means 9. This means our range function will be:

```
range(1,11)
```

Now we slot this into the sample() function. We already know we want to sample six numbers, so let's add that too at the same time.

```
myArray = sample(range(1,11),6)
```

This line of code stores six random numbers in the variable called "myArray". After adding some commands to print out text, our program, and the result of running it, looks like this:



The screenshot shows the Thonny Python IDE interface. The top menu bar includes File, Edit, View, Run, Tools, and Help. Below the menu is a toolbar with icons for file operations and running code. The main editor window displays a Python script named 'random_unique_list.py' with the following code:

```
1 # import two functions random and sample from the python
2 # module called random
3 from random import random,sample
4 ###The sample() function can return a set of random
5 ### unique (non-repeating) integers from a larger population of allowed numbers
6 ### The population size must be greater than the number of samples.
7 ### the built-in range() function can specify the population size which it draws from
8 ### the maximum value is one less than the maximum range.
9
10 #get six numbers in the range 1 to 10
11 myArray = sample(range(1,11),6)
12 print('Six unique integers in range 1 to 10')
13 print(myArray)
14
```

Below the editor is a 'Shell' window showing the execution of the script:

```
>>> %Run random_unique_list.py
Six unique integers in range 1 to 10
[8, 5, 10, 2, 7, 1]
>>>
```

Each time the program is run, we will get a different six numbers generated.

Random numbers

8. Lottery numbers example

Task 3e

The last example showed how to create six unique numbers. We're almost there in creating a lottery ticket.

But in the UK, the numbers on lottery tickets aren't between 1 and 10. They can go up to 49. We'll need to fix that.

For a bonus task, we'll also adjust the script so that it keeps generating new lists until it finds one that includes our lucky number, 21.

Task 3f

Create a file called `random_lottery.py` and include the import command as before.

As in the previous example, we know we are going to use a loop, however we will not know ahead of time how many times it needs to try to find a ticket that includes the lucky number.

21 has a 1 in 49 chance of being selected so it will not loop many times (maybe none, if it was picked the first time). We shall count how many attempts were needed with a counter:

```
i = 1
```

As before, we need a python list to store the numbers. Since we're picking numbers between 1 and 49 this time, we need to adjust the range.

```
myArray = sample(range(1,50),6)
```

We need to check if 21 is NOT present in the list. So the NOT operator is used, and also the python 'in' operator is used. This checks to see if something we're looking for (in this case, the number 21) is present in the list :

```
while 21 not in myArray:
```

If the lucky number is not in the list, then another attempt is made to create a ticket and then the number in the counter is increased by one. The complete loop is :

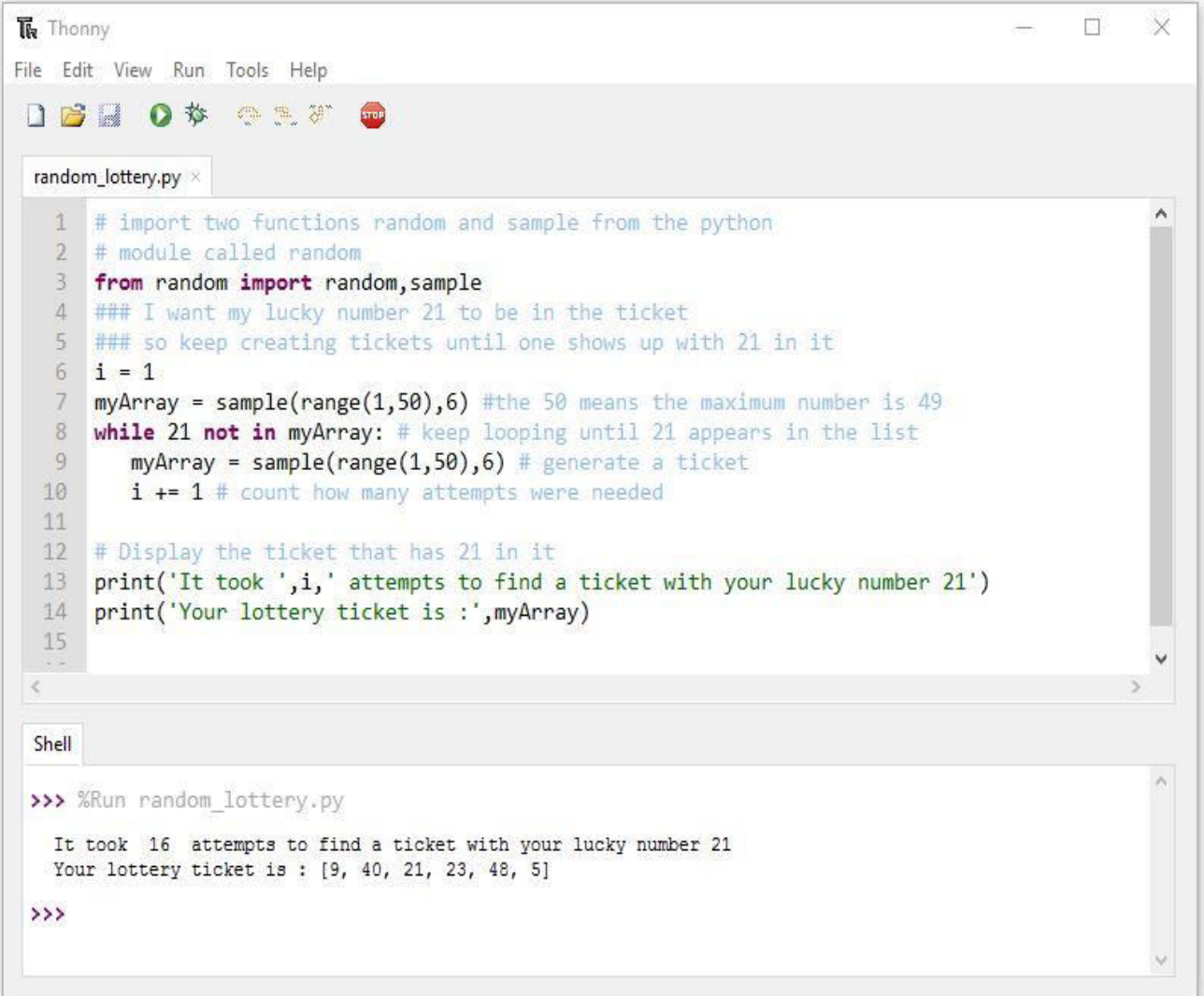
```
while 21 not in myArray:
    myArray = sample(range(1,50),6)
    i = i + 1
```

This will keep on generating ticket sequences until 21 is present, then it exits.

Finally, the resulting ticket is displayed :

```
print('It took ',i,' attempts to find a ticket with your lucky number  
21')  
print('Your lottery ticket is :',myArray)
```

The complete code is shown below, and the Shell shows the output list with non-repeating numbers. This will be different list each time the script is run.



The screenshot shows the Thonny Python IDE interface. The main editor window displays a Python script named `random_lottery.py`. The script imports the `random` module and uses the `sample` function to generate a list of 6 unique numbers from the range 1 to 50. It then enters a `while` loop that continues until the number 21 is found in the generated list. Once found, it prints the number of attempts and the resulting lottery ticket.

```
1 # import two functions random and sample from the python  
2 # module called random  
3 from random import random,sample  
4 ### I want my lucky number 21 to be in the ticket  
5 ### so keep creating tickets until one shows up with 21 in it  
6 i = 1  
7 myArray = sample(range(1,50),6) #the 50 means the maximum number is 49  
8 while 21 not in myArray: # keep looping until 21 appears in the list  
9     myArray = sample(range(1,50),6) # generate a ticket  
10    i += 1 # count how many attempts were needed  
11  
12 # Display the ticket that has 21 in it  
13 print('It took ',i,' attempts to find a ticket with your lucky number 21')  
14 print('Your lottery ticket is :',myArray)  
15
```

The Shell window at the bottom shows the execution of the script:

```
>>> %Run random_lottery.py  
  
It took 16 attempts to find a ticket with your lucky number 21  
Your lottery ticket is : [9, 40, 21, 23, 48, 5]  
  
>>>
```

Random numbers

9. Lottery numbers with user input

Task 3g

Our previous program ensured that our lucky number, 21, always showed up on the ticket. But 21 isn't the lucky number for everyone. We should amend the program to let the user input their own lucky number.

Since we're asking for input from the user, we also need to add in some validation. Otherwise, who knows what the user might input. The program should only allow inputs of numbers between 1 and 49. If an invalid input is entered, then it should give a warning ask the user to input something else.

Copy the 'random_lottery.py' file and rename the copy 'random_lottery_lucky_number.py'. This file has almost all the code we need. We just need to alter it slightly.

Just below the import command, we need to insert some extra code. Since we need to store the user's lucky number, we'll need a variable to keep it in. Let's use "pick", and set it to 0 until the user picks something.

```
pick = 0
```

A WHILE loop will be used to check that the entered value is within 1 to 49

```
while pick < 1 or pick > 49:
```

Notice the use of the < and > operators to check the number as well as the OR operator. Next, the loop is entered (because at this point 'pick' is 0 :

```
picked = input('Enter a number in the range 1 to 49: ')
```

All inputs in python are treated as strings, and we want a number. So we'll need to change the input into an integer using the built-in int() function

```
pick = int(picked)
```

Next, the integer in pick has to be within 1 to 49. This is done with an IF statement

```
if pick < 1 or pick > 49:  
    print(pick, ' is not a valid choice, choose 1 to 49')
```

If the input is not valid, then the code above displays a warning message and what is a valid input. Then it loops back to the WHILE to check on pick, the complete segment is:

```
while pick < 1 or pick > 49:  
    picked = input('Enter a number in the range 1 to 49: ')  
    pick = int(picked)  
    if pick < 1 or pick > 49:  
        print(pick, ' is not a valid choice, choose 1 to 49')
```

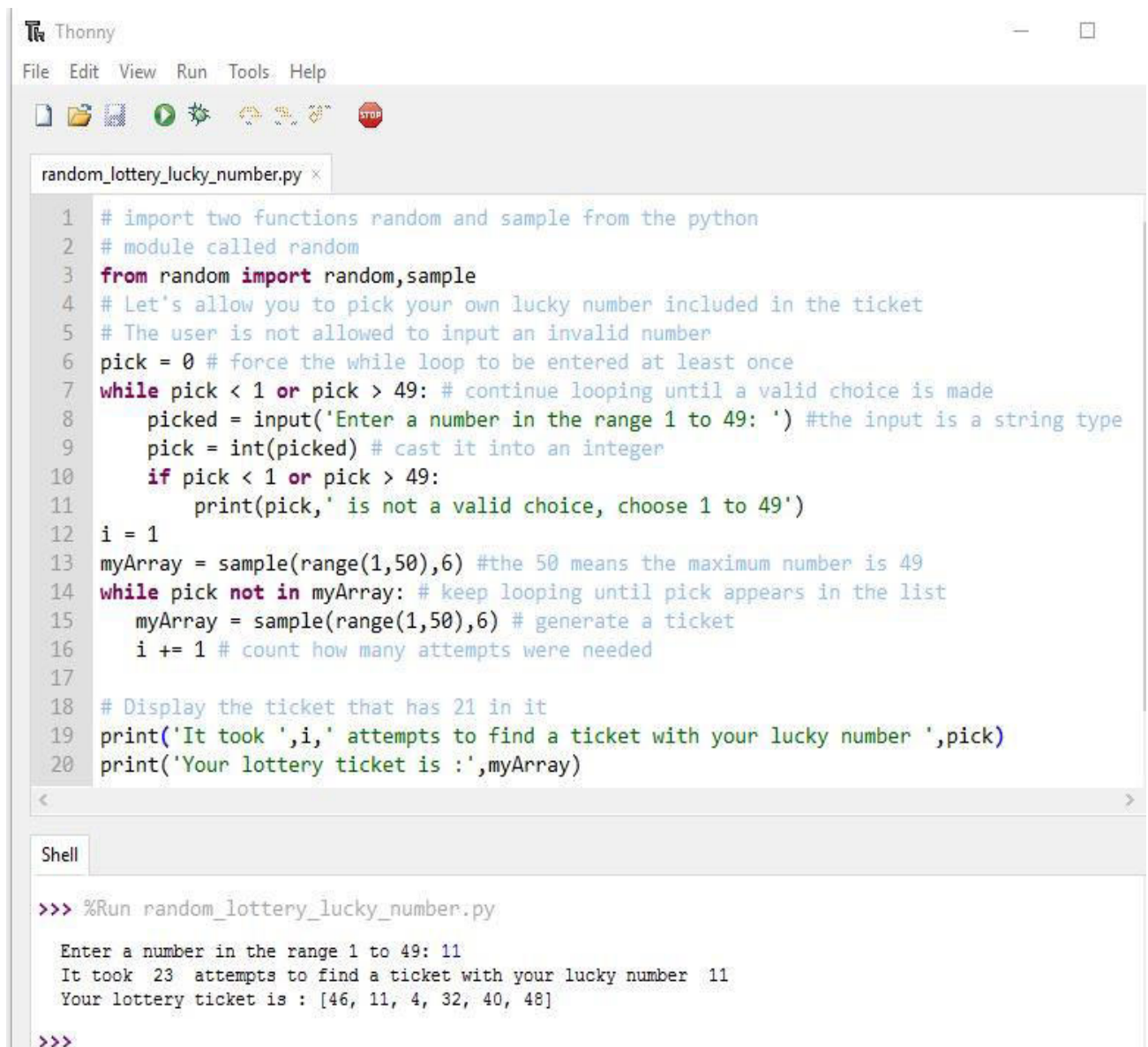
Once a valid lucky number is entered, the code below is entered, this almost the same as lucky number '21' version, but now instead of '21' the variable 'pick' is used.

```
i = 1  
myArray = sample(range(1,50),6)  
while pick not in myArray:  
    myArray = sample(range(1,50),6)  
    i += 1
```



```
print('It took ',i,' attempts to find a ticket with your lucky number  
,pick)  
print('Your lottery ticket is :',myArray)
```

The complete code is shown below, and the Shell shows the output list when 11 was chosen as the lucky number, which you can see is present in the ticket.



The screenshot shows the Thonny Python IDE interface. The top menu bar includes File, Edit, View, Run, Tools, and Help. Below the menu is a toolbar with icons for file operations and execution. The main editor window displays a Python script named `random_lottery_lucky_number.py`. The script implements a lottery simulation where a user picks a number between 1 and 49, and a random ticket is generated until the user's number is included. The script uses `random.sample` to generate the ticket. The Shell window at the bottom shows the execution of the script, where the user entered 11 as their lucky number, and the program outputted the ticket [46, 11, 4, 32, 40, 48] after 23 attempts.

```
1 # import two functions random and sample from the python
2 # module called random
3 from random import random,sample
4 # Let's allow you to pick your own lucky number included in the ticket
5 # The user is not allowed to input an invalid number
6 pick = 0 # force the while loop to be entered at least once
7 while pick < 1 or pick > 49: # continue looping until a valid choice is made
8     picked = input('Enter a number in the range 1 to 49: ') #the input is a string type
9     pick = int(picked) # cast it into an integer
10    if pick < 1 or pick > 49:
11        print(pick,' is not a valid choice, choose 1 to 49')
12    i = 1
13    myArray = sample(range(1,50),6) #the 50 means the maximum number is 49
14    while pick not in myArray: # keep looping until pick appears in the list
15        myArray = sample(range(1,50),6) # generate a ticket
16        i += 1 # count how many attempts were needed
17
18 # Display the ticket that has 21 in it
19 print('It took ',i,' attempts to find a ticket with your lucky number ',pick)
20 print('Your lottery ticket is :',myArray)
```

Shell

```
>>> %Run random_lottery_lucky_number.py
Enter a number in the range 1 to 49: 11
It took 23 attempts to find a ticket with your lucky number 11
Your lottery ticket is : [46, 11, 4, 32, 40, 48]
>>>
```