Python Programming Handbook

This is an informal introduction to Python programming. It introduces the beginner to some fundamental programming concepts such as: input/output, variables and conditional statements.

Overview
One point about programming must be clarified immediately: **Anyone can learn to program computers.** However, you must be willing to **spend some time studying and practising**.

There are two aspects to programming that must be mastered. One concerns **problem solving** and the other concerns the **programming language, in this case Python,** that is to be used.

You must learn how to solve problems. This is the core of programming. But you also must learn how to express your problem solution in Python, so that it can be carried out on a computer. These are two separate skills. You must try not to confuse them. It is difficult, however, to explain one without reference to the other.

Problem Solving Skills
Computer programming is about problem solving. Every computer program solves some particular problem, even programs to play games. It is impossible to write a computer program unless you understand the problem you are being asked to solve. In programming, **you solve** the problem, not the computer. A computer program describes to the computer **what** it must do and **how** it is to be done. You give the computer instructions in the form of a program, telling it what to do and how to do it. **The set of instructions required to solve a problem is a computer program**

The term **algorithm** is used to describe the **set of steps that solve a problem**. An algorithm may be expressed in English or in a programming language. A **computer program must be expressed in a programming language.** In programming, we first develop an algorithm for the problem at hand, and then we ***translate*** this algorithm to a Python program, so that it can be executed (carried out, ran) on a computer.

Sometimes we make mistakes in telling the computer what to do. We overlook part of the problem or do not understand what to do ourselves. In these cases, the computer program will not produce the 'right answer'. It is, however, still solving a problem. It is simply not the problem we wanted to solve. For this reason, it is important to thoroughly check that your programs do indeed solve the problem you intended.

Lesson 1: Output

In this lesson we learn how to perform output – to display messages on your screen.

All program code will be displayed using the `Courier font` in this text.

Output is the term used to describe information that the computer displays (**writes**) on your screen or stores (writes) on a disk drive.

The commonest form of output is that of displaying a message on your screen. In Python, we use the `print` statement to display output on the screen.

The following `print` statement will display the message:

```
 My name is Beth. This is my first program
```

on the screen.

```
     print( 'My name is Beth. This is my first program' )
```

This is a single Python program **statement**. A statement is a command to the computer to carry out an action. So our first Python program is composed of this single statement. To execute this program, it is stored in a file which we call `print1.py.` which contains one line:

```
    print( 'My name is Beth. This is my first program' )
```

To execute the program we use the command python3 which runs the program:

```
% python3 print1.py
My name is Beth. This is my first program
```

You may also run Python programs using an IDE which will be discussed later.

We call the message that `print` displayed a **string**.

A string is a list of characters in quotes. You may put any characters that you wish in a string. Strings can be represented by characters inside either single or double quotes:

```
'My name is Beth.'
"My name is Beth."
```

You can display strings using a `print`:

```
    print( 'Hello ! Goodbye!' )
    print( 'rubbish 123 rubbish xyz @£$%^&*' )
```

Statements

When people communicate in any language, they use **sentences**. When you write down a sentence in English, you have a full stop at the end. This tells us where the sentence ends. You call the full stop a sentence **terminator** i.e. it indicates the end of a sentence.

Similarly, when you write programs you also use sentences to communicate with the computer. In programs, sentences are called **statements**. They also have a terminator. Python statements use the **end of line** to terminate a statement. The 2 statements below are terminated by the **newline character**.

```
print( 'Hello ! Goodbye!' )
print( 'rubbish 123 rubbish xyz @£$%^&*' )
```

After you entered **)** on the first line, you pressed the Return key on the keyboard to start a new line – we call this the **newline character**. Similarly after you entered **)** on the second line, you also pressed Return thus starting a new line.

If you make a mistake entering a statement, a computer will not understand the statement and it will display an error message. This often caused by a **spelling error** or a **missing bracket** or **quotation** mark. For example, in a programming, just as in English, you must always have the correct number of quotation marks and brackets.

Left brackets such as (, { and [are called **opening** brackets.

The quotation marks at the start of a phrase: **"** (double) and **'** (single) are called **opening** quotes.

Right brackets such as) , } and] are called **closing** brackets and quotation marks at the end of a phrase are called **closing** quotes.

*A simple rule is that **for every opening** bracket or quotation mark you must have a **corresponding closing** bracket or quotation mark.*

Example 1.1: Matching brackets and quotes:

```
'Hello'
( age  > 10 )
'x'
list[10]
```

Breaking this rule causes a **syntax error**. The following are examples of syntax errors:

```
age > 10)
'x
list[10
prin( 'Hello ! Goodbye!' )
```

The last error is due to misspelling `print`

What to do when an error occurs
When a syntax error occurs, you must work out what mistake(s) you have made. This means checking the statements of your program and seeing where the syntax is incorrect.

You then **edit** your program to correct the mistake. When you have corrected your program, it can be executed.

To execute a program or to run a program means that the computer carries out the statements making up the program.

A Python program is made up of a group of one or more statements. These statements allow us to control the computer.

Using them, we can **display information on the screen**, **read information from the keyboard** and **process information** in a variety of ways.

As we proceed we will describe the different kinds of statements which are:

Input/Output (I/O) statements (e.g. display information on screen) Lessons 1 and 2

Variable manipulation statements (e.g. to do arithmetic) Lesson 3

Conditional statements (e.g. to make decisions) Lessons 4 and 5

We will also look at more complex variables called **arrays** and **lists** in Lesson 6

Lesson 1 Exercises – try these in your own time

1. What is the output of the following print statement?

   ```
   print( 'Have a great day!')
   ```

   a. 'Have a great day!'

   b. 'Have a great day'

   c. Have a great day!

   d. 'Have a great day!'

2. What is the output of the following statements?
   ```
   print('Hi there!')
   print('How are you doing?')
   ```
   a. Hi there! How are you doing?
   b. How are you doing? Hi there!
   c. 'Hi there!'
      How are you doing?'
   d. Hi there!
      How are you doing?

3. Write a program that prints a message saying

   I love Python!

4. Write a program that prints a message saying your name and your age, e.g.

   My name is Colin. I am 20 years old!

5. Write a program to display the message 'Welcome to Python' three times, on separate lines using three `print` statements.
6.
7. Write a program to display the message 'Python is awesome!' two times, on separate lines, using only one `print` statement and \n

8. What are the syntax errors in the following statements:

   ```
   print( 'Hello ! Goodbye! )
   print( Hello ! Goodbye!' )
   print( 'Hello ! Goodbye!'
   print 'Hello ! Goodbye!' )
   prlnt( 'Hello ! Goodbye!' )
   ```

Lesson 1 Assignments – you email these to UCD

1. Write a program that prints out your favourite food, followed by a blank line, followed by your favourite colour, followed by a blank line, followed by your favourite animal. Use either the `\n` character in one `print` statement or use separate `print` statements.

2. Print a square made up of 4 @ characters per line using a single `print` statement.

```
@@@@
@@@@
@@@@
@@@@
```

Lesson 2: Input and Variables

Input is the term used to describe the transfer of information from the keyboard (or a disk) to the computer. We can use the word **read** for input e.g read information from the keyboard. The question arises – where do we store the information we read in. This introduces the concept of **variables**.

A variable is a container for information.

This means that we can store information in a variable. It is called a variable because at any time we can change (**vary**) the information it stores.

So when we input information, we store it in a one or more variables. We **give each variable a unique name**, which we use to identify it. The following are examples of variable names we could use in a Python program:

```
colour
my_age
pension_age
name
taxcode
tax_rate
temperature
name
hourly_pay
```

Fundamental principle of writing clear programs

Choose meaningful names for variables

Using meaningful variable names it makes your programs easier to understand. For example, if you are writing a program which deals with pension ages then you could use any of the following variable names to store the pension age but which one makes is easiest to understand:

```
pension_age
pa
p
x
pna
```

The variable name `pension_age` is the obvious choice. When you see this name you automatically know what it the variable is being used for. If you use a name like `p` or `x` then the name gives you no idea about what the variable is being used for.

We use the Python `input` statement to read information from the keyboard, into a variable.

Example 1: Write a program to ask the user to enter their favourite colour. The program reads this colour that the user types on the keyboard and displays a message followed by the colour entered by the user.

```
    favourite_colour = input('Enter your favourite colour: ')

    print( favourite_colour )
```

If we execute the program the following appears on the screen (the bolded text is that entered by the user on the keyboard. We will use this convention throughout the text).

```
Enter your favourite colour: blue
blue
```

The variable `favourite_colour` is used to store the characters that the user types on the keyboard that is, it will store a list of characters.

When you use a variable name with `print` it will display the **value** of the variable i.e. the string `blue` in the example above.

The `input()` statement does two tasks: it displays the string in quotes and then reads text from the keyboard, (for example the word *blue* may be entered), and it places this text in the variable `favourite_colour`.

We have seen that the `print` statement is used to display output on the screen. It can be used to display strings and numbers in the same `print` statement.

Example 2: Write a program to ask the user to enter their favourite colour. The program reads this colour that the user types on the keyboard and displays a message followed by the colour entered by the user.

```
    favourite_colour = input('Enter your favourite colour: ')

    print('Yuk ! I hate ', favourite_colour )
```

If we execute the program the following appears on the screen:

```
Enter your favourite colour: blue
Yuk ! I hate blue
```

The statement

```
print('Yuk ! I hate ', favourite_colour )
```

instructs the computer to display the message *Yuk ! I hate* followed by the value of the variable `favourite_colour` i.e. *blue* in this example.

We use the expression 'the value of a variable' to mean 'the value contained in a variable'.

We take the phrase 'the value of `favourite_colour` is `blue`' to mean 'the value contained in the variable `favourite_colour is` blue'. We will use the shorter form from now on.

Make sure you understand the difference between:

```
          print( 'favourite_colour' )
```

and

```
          print( favourite_colour )
```

In the first case, the word (string) `favourite_colour` appears on the screen.

In the second case, the **value** of a variable called `favourite_colour is` displayed which could be anything, for example the word *blue* or whatever value the user has given the variable like *red*, *pink* and *orange*. You can store **many words** in a string variable.

Rules for Variable Names

Python and every programming language, has rules on how you name variables:

- **A variable name can only contain the following**:
    - **letters** (lowercase and uppercase, ie a–z and A–Z)
    - **digits** (0–9)
    - the underscore character '_'

- A variable name **cannot start** with a digit

- Variable names in Python are **case-sensitive –** it distinguishes between uppercase and lower case letters so that `colour` and `Colour` are **different variables**

- There are a number of **reserved words** or **keywords** that have built-in meanings in Python and cannot be used as variable names (e.g. if, `return, def, del, break, for, in,  else, while, import`)

The following are legal or valid variable names in Python:

```
Colour, name, firstname, surname, class1, class 602,
first_name, second_name, address_line1.
```

The use of the underscore '_' character is very useful in creating meaningful names made up of 2 or more words.

Do not confuse the underscore character with the minus sign '-'. The minus sign (or any other symbol) cannot be used in a variable name. thus `first-name` is not a valid variable name.

Comments

In a Python program, any text after **#** is called a comment and **is ignored by Python**. This text is there to help explain to someone reading the program, what the program does and how the program works. **Comments are an important component of programs**. This is because when you read your programs some time after writing them, you may find them difficult to understand, if you have not included comments to explain what you were doing. They are even more important if someone else will have to read your programs e.g. your tutor who is going to grade them!

It is a useful idea to start all programs with comments which give the name of the file containing the program, the purpose of the program, the authors name and the date on which the program was written, as the first comments in any program. Example 2 could be written as:

```
# colour.py: Prompt user to enter colour and display a message
# Author: Joe Carthy
# Date: Oct 20 2022

favourite_colour = input('Enter your favourite colour: ')

print('Yuk ! I hate ', favourite_colour )
```

Lesson 2 Exercises

1. Valid or invalid variable names
    a. Is the variable name `TotalMarks` correct?
    b. Is the variable name `number-of-students correct?`
    c. Is the variable name `firstName` correct?
    d. Is the variable name `myVar1` correct?
    e. Is the variable name `customerName` correct?
    f. Is the variable name `productPrice` correct?
    g. Is the variable name 3`rdStudent` correct?
    h. Is the variable name `isAvailable`? correct?
    i. Is the variable name `total-sales` correct?
    j. Is the variable name `customer_email` correct?

3. Write a program that asks the user for their name using `input`. Store the name in a variable and display a personalized greeting using the variable.

4. What is the output, if any, of the following program:

```
# print('hello\n')
# print('bye bye\n')
```

5. Write a program that prompts the user to enter their favourite colour and favourite animal using the `input`. Store these values in separate variables and display them in a sentence :

```
My favourite colour is … and my favourite animal is ..
```

Lesson 2 Assignments

i. Write a program that prompts the user to enter their favourite number using `input`. Store the number in a variable and print a message that includes the user's favourite number.

ii. Create a program that simulates a hospital registration system. Prompt the user to enter the following information:

Name
Surname
Age
Height (in cm)

Store each piece of information in a separate variable with an appropriate name. Finally, print the information in the following format:

Name: Joe
Surname: Carthy
Age: 100

Height: 180

Lesson 3: Variables and Assignment

In Lesson 2 we used `input` to give a variable its value.. Giving a variable a value is called **assignment**. We can use assignment to give a value to a variable directly in a program without input. We may give the variable a value or compute a value based on the values of other variables. For example, suppose we have a variable called `metres`, to which we wish to give the value `12`. In Python we write:

```
metres = 12
```

This is usually read as **'`metres` is assigned the value 12'**. Of course, we could use any value instead of `12`. Other examples of assigning values to variables are:

```
centimetres = 50
litres = 10
metres = 4
colour = 'red'
name = 'Joe Carthy'
pay_per_hour = 10.5
```

Example L3.1: Write a program to convert 5 metres to centimetres. A simple Python program to do this is given below.

```
#convert1.py: converts metres to centimetres
#Author: Joe Carthy
#Date: 21/10/2023

metres = 5
centimetres = metres * 100
print('5 m is ', centimetres, 'cms',)
```

Executing this program produces as output:

```
5 m is 500 cms
```

Here we use the value of the variable `metres` to compute the value of the variable `centimetres`.

Other examples of such an assignment are:

```
gallons = 4
pints = gallons * 8
kilometres = 4
metres = 18
cms = (kilometres * 100000) + (metres * 100)
```

The program to convert metres to centimetres as presented in Example L3.1 is very limited in that it always produces the same answer. It always converts 5 metres to centimetres. A more useful version would prompt the user to enter the number of metres to be converted and display the result:
Example L3.2: Converting metres to centimetres, version 2.

```
#convert2.py: converts metres to centimetres version 2
#Author: Joe Carthy
#Date: 21/10/2023

m = input('Enter number of metres: ')

metres = float ( m )

centimetres = metres * 100
print(metres, 'metres is ', centimetres, 'cms',)
```

Executing this program produces as output:

```
Enter number of metres: 4
4.0 metres is 400.0 cms
```

Variable Types

Important note: The `input` function reads from the keyboard and returns a list of characters i.e. a string. Thus the variable m in the example above contains the string '4' and not the number 4.

This is very confusing for beginners to programming. A fundamental aspect of variables is that they have a **type**. The type of a variable tells you what kind of data it stores.

In our programs we will use three types: **int** (whole numbers), **float** (numbers with decimal point) and **string** (list of characters).

When you are working with numbers and wish to do arithmetic with them (add, subtract, multiply and divide) then you must use either the type **int** or **float**.

So it is crucial to understand the difference between the number 42 and the string '42' as used in the following:

```
a = 42
b = a * 2
```

This results in `b` having the value 84. A and b are of type **int** in this case.

```
x = '42'        # x is type string
y = x * 2       # y is type string
```

This results in `y` having the value `4242` – string of characters.

When you ***'multiply'*** a string variable by `n` you get `n` copies of the string e.g.

The code:

```
x = 'bye'
y = x * 3
```

gives `y` the string value `'byebyebye'`

This brings us back to Example L3.2 and the statements

```
m = input('Enter number of metres: ')

metres = float ( m )
```

The variable `m` is of **type string**.

The `float` function converts the string `m` to a number with a decimal point (real number).

This means that `metres` now contains a number which we can do arithmetic with.

The output of L3.2 is 'crowded' in that there is no blank line before or after the output or between the two lines of output. This makes it hard to read the output. You can use the '\n' character in strings to start new lines.

The version below fixes this issue by putting one '\n' in the `input`() function and 3 in the `print`() function.

It also uses a shortcut to avoid using a string variable, m, in the previous examples. It does this by converting the string from `input` to a float in one statement:

```
    metres = float (input('\nEnter number of metres: '))
```

Example L3.3: Converting metres to centimetres, version 4

```
# convert4.py: converts metres to centimetres version 3
# Outputs extra blank lines to make it easier to read the output

#Author: Joe Carthy
#Date: 21/10/2022

metres = float (input('\nEnter number of metres: '))

centimetres = metres * 100

print('\n', metres, 'metres is ', centimetres, ' centimetres\n\n'  )
```

When you run it, notice the extra blank lines

```

Enter number of metres: 3.5
```

```
3.5 metres is 350.0 centimetres
```

Some Fun making the computer beep!

When you use '\a' (called the BEL character) in `print`, the computer makes a beep sound – it does not display anything. So the program below simply plays 3 beeps.

```
# beep.py: Just for fun - beep 3 times !!

print('\a \a \a')
```

Example L3.4: As another example of the use of I/O and variables consider a simple calculator program. This program prompts for two numbers, adds them and displays the sum.

```
# calc.py: Calculator program to add 2 numbers
# Author: Joe Carthy
# Date: 01/10/2023

number1 = float(input('\nEnter first number: '))

number2 = float(input('\nEnter second number: '))

sum = number1 + number2

print('\n\nThe sum of', number1, 'and', number2, 'is', sum, '\n\n')
```

calc.py outputs:

```
Enter first number: 2.4

Enter second number: 5.76

The sum of 2.4 and 5.76 is 8.16
```

Variables must be defined before you use them other statements

Variables must be defined before you use them – you must give them a value.

If a variable is 'not defined' (not assigned a value), trying to use it will generate an error.

So if you run the 1 line program:

```
    print ('x = ', x)
```

An error is displayed because the variable `x` has not been defined. The error message may not be very user friendly such as: that below – the last line is the helpful one:

```
Traceback (most recent call last):
File "<string>", line 1, in <module>
NameError: name 'x' is not defined
```

The most common reason for this error is mis-spelling the name of.variable as in the code below

```
metres = 5
cms = metres / 100
print (f'{metrs} = {cms} centimetres')
```

Here we have misspelled `metres` in the `print` statement and an error is displayed:

```
Traceback (most recent call last):
  File "<string>", line 3, in <module>
NameError: name 'metrs' is not defined
```

More on print() function and displaying variables

It can get quite complicated when we output strings and variables using print as in the statement

```
print('\n\nThe sum of', number1, 'and', number2, 'is', sum, '\n\n')
```

There is a simpler way to display this message with print using **f-strings**. We put the character **f** as the first item in print:

```
print(f'\n\nThe sum of {number1} and {number2} is {sum} \n\n')
```

which produces identical output to the earlier print

```
The sum of 2.4 and 5.76 is 8.16
```

When using an f-string, **we enclose any variable we wish to display in {}** brackets.

print will display the value of each variable in {}.

This avoids having to have separate strings in quotes, separated by commas, as in the earlier version of print.

As another example consider the following variables and how we want to display them:

```
Name = ‘Joe Bloggs’
rate = 10.00
num_hours = 40
pay = rate * num_hours
```

Without using an f-string we display using:

```
print('Pay for ', name, 'at ', rate, 'per hour is', pay)
```

which outputs

```
Pay for Joe Bloggs at 10.0 per hour is 400.0
```

We can display the same output with a simpler print using **f-strings**:

```
print(f'Pay for {name} at {rate} per hour is {pay}')
```

displays the same output as the first print() above.

```
Pay for Joe Bloggs at 10.00 per hour is 400.00
```

Displaying a fixed number of decimal places

Python will display the result of numeric calculation to many decimal places.

For example,

```
x = 19/3.768

print(f'x = {x}' )
```

will output on my Mac computer:

```
x = 5.042462845010616
```

In most of our calculations it is enough to display result with 2 decimal places.

We use an f-string to do this by following the variable in {} with **:*.number of decimal points*f** you wish to output e.g. {x:.2f} specifies to display x to 2 decimal places.

You can change the number from 2 to whatever you wish, to have that number of places displayed after the decimal point.

To print x to 2 decimal points

```
x = 19/3.768

print(f'x = {x:.2f}')
```

outputs

```
x = 5.04
```

Lesson 3 Exercises

1. What is the data type of each variable?
    a. What is the data type of the variable 'age'? `age = 25`
    b. What is the data type of the variable 'name'? `name = 'John Doe'`
    c. What is the data type of the variable 'price'? `price = 9.99`
    d. What is the data type of the variable 'quantity'? `quantity = 10`
    e. What is the data type of the variable 'message'? `message = 'Hello'`
    f. What is the data type of the variable 'discount'? `discount = 0.2`

2. Write a program to convert 10 dollars to kyats using an exchange rate of 1 dollar = 2100 kyats.

```
10 dollars = 21000 kyats
```

Use `print` with f-strings in all of the following programs

3. Write a program that takes a single length (a float) and calculates the following:
    • The area of a square with side of that length. (length * length)
    • The volume of a cube with side of that length. (length ** 3)
    • The area of a circle with diameter of that length (3.14 * (length/2)**2))

```
Enter length: 4

Area of square:     16.0
Volume of cube:     64.0
Area of circle:     12.56
```

4. Write a program that takes an amount (a float), and calculates the tax due according to a tax rate of 20%

```
Enter amount for tax at 20%: 200.0

Tax: 40.00
```

5. Write a program to simulate a cash register for a single purchase. The program reads the unit cost of an item and the numbers of items purchased. The program displays the total cost for that number of units:

```
Enter unit cost: 5
Enter number of units: 6

Total cost of 6 units: 30.00
```

6. Modify the programs 3, 4 and 5 above to display the output to one decimal point e.g.

```
Area of circle:          12.5
```

Lesson 3 Assignments

Use `print` with f-strings in all of the following programs

1. Ask the user to enter a temperature in Celsius and convert it to Fahrenheit using the formula:
   Fahrenheit = (Celsius * 1.8) + 32.
   Print the converted temperature in Fahrenheit.

```
Enter temperature in Celsius: 100
100 degrees Celsius = 212.0 degrees Fahrenheit
```

2. Convert dollars to kyat as follows:
   a. Display 'Dollar to Kyat conversion program'
   b. Ask the user to enter an amount in dollars.
   c. Ask the user to enter the kyat exchange rate for dollars.
   d. Calculate kyat amount by multiplying the dollar amount by the exchange rate.
   e. Print out the dollar and kyat amounts

```
Dollar to Kyat conversion program
Enter amount in dollars: 10
Enter dollar to kyat exchange rate: 2100
10 dollars = 21000 kyats
```

3. Write a program to calculate how much someone gets paid per week based on the number of hours they work per week. The program asks the user to enter the number of hours worked and the rate per hour and then displays the total pay, with a blank line between each line of output:

```
Enter number of hours worked: 20.5
Enter rate per hour: 10
Total pay = 205.0
```

4. Write a program to display your total savings for 3 weeks based on saving $10 in Week 1, $15 in Week 2, and $20 in Week 3. Use 3 variables, one for each week's savings and one the total amount saved. Then calculate the total amount of money saved over the three weeks by adding the 3 variables. Print the result as follows:

```
You saved a total of 45 dollars
Week 1 you saved 10 dollars
Week 2 you saved 10 dollars
Week 3 you saved 10 dollars
```

5. You sell 10 cups of lemonade at $2.50 each. Calculate the total amount of money you earned by multiplying the number of cups sold by the price per cup using 3 variables, one for the number of cups, one for the cost per cup and one for the total amount sold. Print the result as follows, with a blank line between each line of output:

```
Number of cups sold: 10
Price per cup: 2.50
Total sold: 25.00
```

Lesson 4: Conditional Statements – `if` statement

All of our programs to date have been made up of one or more statements. The statements have been executed one after the other in our programs, that is sequentially. However there are many times in programming where we do not wish to execute statements in this way. Sometimes we wish to skip over some statements and sometimes we wish to repeat some statements. This is the purpose of conditional statements.

People are familiar with making decisions. For example, consider the following sentences:

> If I get hungry, I will eat my lunch.
> If the weather is cold, I will wear my coat.

These two sentences are called **conditional sentences**. Such sentences have two parts: a **condition part** ('If I get hungry', 'If the weather is cold') and an **action part** ('I will eat my lunch', 'I will wear my coat').

The action will be only be carried out if the condition is satisfied. To test if the condition is satisfied we can rephrase the condition as a question with a yes or no answer. In the case of the first sentence, the condition may be rephrased as 'Am I hungry ?' If the answer to the question is yes, then the action will be carried out (i.e. the lunch gets eaten), otherwise the action is not carried out.

We say the condition **is true** (**evaluates to true**) in the case of a yes answer. We say the condition **is false** (**evaluates to false**) in the case of a no answer. Only when the condition is true will we carry out the action. This is how we handle decisions daily.

In programming, we have the same concept. We have **conditional statements**. They operate exactly as described above. One of these is known as the i**f statement**. This statement allows us evaluate (test) a condition and carry out an action if the condition is true.

In Python, the keyword **`if`** is used for such a statement. As an example, we modify the program to convert metres to centimetres to test if the value of metres is positive (greater than 0) before converting it to centimetres.

Note you put a `':'` after the condition in an `if` statement

The action statement(s) are **indented** in Python. This allows Python to identify the statements making up the actions to be carried out, when the condition is true.

The action statements end with the first non-indented statement follow the `if`.

Example L4.1

```
# convert5.py: converts metres to centimetres
# check quantity of metres is positive
# Author: Joe Carthy
# Date: 21/10/2023

metres = float (input('\nEnter number of metres: '))

# Check if metres is positive

if metres > 0:
    centimetres = metres * 100
    print(f'\n {metres} metres is {centimetres} centimetres\n\n')

if metres <= 0:
    print(f'\nEnter a positive number for metres\n')
    print(f'\nYou entered: {metres} \n\n')
```

Executing this program with -42 as input produces as output:

```
Enter number of metres: -42

Enter a number for metres

You entered -42
```

The first `if` statement tests if the value of `metres` is greater than `0 (metres > 0)`**.** If this is the case, then the conversion is carried out and the result displayed.

Otherwise, if the value of `metres` is not greater than 0, this does not happen i.e. the two action statements are skipped.

The second `if` statement tests if `metres` is less than or equal to 0. If this is the case, then the message to enter a positive value is displayed and the value entered is displayed. If this is not the case the two `print` statements are skipped and the program terminates.

In this particular example, only one of the conditions can evaluate to true, since they are **mutually exclusive** i.e. `metres` cannot be greater than 0 and at the same time be less than or equal to 0.

Because this type of situation arises very frequently in programming i.e. we wish to carry out some statements when a condition is true and other statements when the same condition is false, a special form of the `if` statement is provided called the **`if-else`** statement. The general format may be written as

```
if (condition):
     action statements1 #carried out if condition is true
else:
     action statements2 #carried out if condition is false
```

Example L4.2

We rewrite the program L4.1 using `if-else`:

```
# convert6.py: converts metres to centimetres
# check quantity of metres is positive
# Author: Joe Carthy
# Date: 21/10/2023

metres = float (input('\nEnter number of metres: '))

if metres > 0:
    centimetres = metres * 100
    print(f'\n {metres} metres is {centimetres} centimetres\n\n')
else:
    print(f'\nPlease enter a positive number for metres\n')
    print(f'\nYou entered: {metres} \n\n')
```

This program operates in the same way as the previous example. However, it is more efficient, in that the condition has only to be evaluated once, whereas in first example, the condition is evaluated twice. Note the action statements for `else` must be indented just as for `if`.

Example L4.3

This program prompts the user to enter the number of hours worked in a week and the rate of pay per hour. It displays the weekly pay calculated as (hours worked) * (rate [per hour).

There are two conditions. Workers can work a maximum of 100 hours per week and the maximum hourly pay rate is 50. The program checks these two conditions

```
# pay.py: Calculate and display hourly pay

hours_worked = float(input('\nEnter hours worked: '))

if hours_worked > 100:
    print(f'\nHours worked cannot exceed 100: {hours_worked}')
else:
    rate_per_hour = float(input('\nEnter rate per hour: '))
    if rate_per_hour > 50:
        print(f'\nRate too large: {rate_per_hour}')
    else:
        pay = rate_per_hour * hours_worked
        print(f'\nPay = {pay} for {hours_worked} hours worked at
{rate_per_hour} per hour')
```

Enter number of hours worked: **20**

Enter rate per hour: **20**

```
Pay =  400.0 for 20.0 hours worked at 10 per hour
```

In L4.2, if a user enters a value greater than 100 for hours worked, the condition in the first `if` is true so the action is carried out (display that this number is too large) and all of the statements following `else` are skipped.

If a valid number of hours is entered, then we carry out the statements for the first `else`. Here we read the hourly rate.

If this. number is too large we display an error and skip the statements in the second `else`, otherwise we carry out these statements i.e. calculate the pay and display it.

This programs shows that we can have `if` and `if-else` statements as part of the actions for any `if` statement.

We call a condition (e.g. `metres > 0`) a **Boolean expression** or a **conditional expression**.

This simply means that there are only two possible values (**true** or **false**) which the condition can yield.

A **Boolean expression** evaluates to either **true** or **false**.

More on Conditions

There are only six types of condition that can arise when comparing two numbers.

We can compare for:

1. equality - are they the same ?
2. inequality – are they different ?
3. is one greater than the other ?
4. is one less than the other ?
5. is one greater than or equal to the other ?
6. is one less than or equal to the other ?

The following illustrates how to write the various conditions to compare the variable feet to the number 0 in Python:

```
( feet  == 0 )           is feet equal to 0?
( feet  != 0 )           is feet not equal to 0?
( feet  >  0 )           is feet greater than 0?
( feet  <  0 )           is feet less than 0?
( feet  >= 0 )           is feet greater than or equal to 0?
( feet  <= 0 )           is feet less than or equal to 0?
```

Technically, the symbols ==, <>, <, >, <=, and >=, are called **relational operators**, since they are concerned with the relationship between numbers.
We can also compare two strings using the same operators. We often want to test if one string is the same as (equal to) another string as we shall in the next example.

Example L4.4

A calculator program to handle both subtraction and addition. The user is prompted for the first number, then for a '+' or '-' character to indicate the operation to be carried out, and finally for the second number. The program calculates and displays the appropriate result:

```
# calc2.py: Calculator program to add or subtract numbers
# Author: Joe Carthy
# Date: 01/10/2023

number1 = float(input('\nEnter first number: '))

operation = input('\nEnter operation [+ or –]  ')

number2 = float(input('\nEnter second number: '))

if operation == '+':
   sum = number1 + number2
   print(f'\n\nThe sum of {number1} and {number2} is {sum} \n\n')
else:
   diff = number1 - number2
   print(f'\n\nTaking {number2} from {number1} is {diff} \n\n')
```

Executing this program produces as output:

```
Enter first number:  9

Enter operation [+ or -]: -

Enter second number: 4

Taking 4.0 from 9.0 is 5.0
```

We compared the string operation with the string '+' in the above code.

The code in L4.4 'assumes' that if the operation is not '+' then it must be '-' but the user could have hit the wrong key. Example L4.5 below, checks that the actual characters '+', or '-' were entered. It deals with the possibility that it was neither '+', or '-' that is the user made a mistake.

User data entry mistakes are very common and professional programmers always check that the user input is as was expected.

Example L4.5

```
# calc3.py: Calculator program to add or subtract 2 numbers
# Author: Joe Carthy
# Date: 01/10/2023

number1 = float(input('\nEnter first number: '))

operation = input('\nEnter operation [+ or –] ')

number2 = float(input('\nEnter second number: '))

if operation == '+':
   sum = number1 + number2
   print(f'\n\nThe sum of {number1} and {number2} is {sum} \n\n')

else:
   if operation == '-':
      diff = number1 - number2
      print(f'\n\nTaking {number2} from {number1} is {diff} \n\n')

   else:
      print(f'\nInvalid operation only + and – allowed\n')
      print(f'You entered: {operation} \n')
```

Executing this program produces as output:

```
Enter first number:  9

Enter operation [+ or -]: *

Enter second number: 4

Invalid operation – only + and – allowed
You entered: *
```

The code in L4.5 checks that a valid operation is entered (= or -). However, if the user enters an invalid operation, then there is no need to ask for the second number, as we did in L4.5. L4.6 below, addresses this issue but we have to understand combining conditions first.

Combining conditions: and/or

We often need to combine two or more conditions in a statement. For example when we make a decision on wearing a coat we might decide based on:

> If *it is raining* **and** *it is cold* then I will wear a warm raincoat

Here we test two conditions: is it raining **and** is it cold. We only carry out the action (wear a warm raincoat) if both conditions are true.

When we use **`and`** to combine conditions, we only carry out the action if both (***all***) the conditions are true.

Sometimes we wish to test if either one of two conditions is true as in

If it is raining or it is cold then I will wear a coat

In this case, if any one of the conditions (is it raining /is it cold) is true, then we carry out the action (wear a coat).

In our calculator program, we only need to ask the user to enter a second number if the operation is either '+' or '-'. We can use the statement

> if (operation == '+') **or** (operation == '-'):

to test if ***either*** '+' **or** '-' has been entered. If the user has entered one of these operations then the actions for the if are carried out.

If an invalid operation has been entered, then the actions of the last else are carried out.

Example L4.6

```
# calc4.py: Calculator program to add or subtract 2 numbers
# ask for second number if a valid operation has been entered
# Author: Joe Carthy
# Date: 01/10/2023

num1 = float(input('\nEnter first number: '))

operation = input('\nEnter operation [+ or -] ')

if (operation == '+' ) or (operation == '-'):
# user must have entered + or -
    num2 = float(input('\nEnter second number: '))

    if operation == '+':
        sum = num1 + num2
        print(f'\n\nThe sum of {num1} and {num2} is {sum} \n\n')
    else: # must be a -
        diff = num1 - num2
        print(f'\n\nTaking {num2} from {num1} is {diff} \n\n')

else:
    print(f'\nInvalid operation only + and - allowed\n')
    print(f'You entered: {operation} \n')
```

In L4.7 we rewrite L4.6 using **`and`** to test if a valid operation was entered. In this case we test if the operation was not '+' **and** was not '-'. If both conditions are true, then the operation is not a '+ and it's not a '-', so it is invalid.

```
if (operation != '+' ) and (operation != '-'):
          # then it must be an invalid operation
```

Example L4.7

```
# calc5.py: Calculator program to add or subtract 2 numbers
#  ask  for  second  number  if  a  valid  operation  +  or  -  has  been
entered
# Author: Joe Carthy
# Date: 01/10/2023

num1 = float(input('\nEnter first number: '))

operation = input('\nEnter operation [+ or -] ')

if (operation != '+' ) and (operation != '-'):
    print(f'\nInvalid operation only + and - allowed\n')
    print(f'You entered: {operation} \n')


else:
    num2 = float(input('\nEnter second number: '))

    if operation == '+':
        sum = num1 + num2
        print(f'\n\nThe sum of {num1} and {num2} is {sum} \n\n')

    else:
        diff = number1 - number2
        print(f'\n\nTaking {num2} from {num1} is {diff} \n\n')
```

Example L4.8: Range checking

We often need to check if the input to a program **lies in a range.** For example, the age of a child in Ireland lies between 1 and 18 years which can be expressed as great than 0 and less than 18 years (age > 0 and age <18). The age of an adult is from 18 upwards, but we need an upper limit such as 122, so we test if age lies between 18 and 122: (age >18 and age <= 122). Note that 122 years is the age of oldest person to have lived so far.

Example L4.8 is a program to read a person's age and output whether they are a child or an adult using range checking.

```
# age.py: Check if age is for a child or an adult
# Author: Joe Carthy
# Date: 01/10/2023

age = float(input('\n Enter age: '))

if ( age > 0 ) and ( age < 18 ):
    print(f'\n Child age: {age}')

if (age >= 18 ) and ( age <= 122 ):
    print(f'\n Adult age: {age}')

if (age <= 0 ) or ( age > 122 ):
    print(f'\n {age} not in age range for a person \n')
```

L4.8 outputs

```
 Enter age: 34

 Adult age: 34.0
```

and

```
 Enter age: 12

 Child age: 12.0
```

and

```
 Enter age: 150

 150.0 not in age range for a person
```

General format of if, if-else

```
if condition:
    action1A
    action1B
```

The statements `action1A` and `action1B` will only be executed **if the condition is true**

```
if condition:
    action1A
    action1B
else:
    action2A
    action2B
```

The statements `action2A` and `action2B` will only be executed **if the condition is false**

Note that the `action` statements can be **any Python statement,** including another `if` statement.

```
if condition1 and condition2:
    action1A
    action1B
```

The statements `action1A` and `action1B` will be executed **if both** `condition1` **and** `condition2` **are true**

```
if condition1 or condition2:
    action1A
    action1B
```

The statements `action1A` and `action1B` will be executed **if any one (or both) of** `condition1` **or** `condition2` **is true**

Lesson 4 Exercises

1. What are the 6 conditions that we can use to compare two numbers?

2. Write a program that asks the user to enter their exam score. If the score is greater than or equal to 60, display 'Congratulations! You passed the exam'. Otherwise, display 'Sorry, you did not pass the exam'.
   ```
   score = int(input('Enter your exam score: '))
   if score >= 60:
        print('Congratulations! You passed the exam.')
   else:
        print('Sorry, you did not pass the exam.')
   ```

3. Write a program that asks the user to enter a password. If the password is 'password123', display 'Access granted' Otherwise, display 'Access denied'

4. Write a program that prompts the user to enter their age and whether they have a driver's license ('yes' or 'no). If the person is 18 or older and has a driver's license, display 'You can legally drive'.
   If the person is 18 or older but does not have a driver's license, display 'You can apply for a driver's license'.
   If the person is under 18, display 'You are not old enough to drive'.

   ```
   # Prompt the user to enter their age

   age = int (input('Enter your age: '))

   # Prompt whether they have a driver's license

   has_license = input('Do you have a drivers license? [yes/no]:
   ')

   # Check the driving eligibility

   if (age >= 18) and (has_license == 'yes'):
       print('You can legally drive.')

   if (age >= 18) and (has_license == 'no'):
       print('You can apply for a drivers licence')

   if (age < 18):
       print ('You are not old enough to drive.')
   ```

5. Write a program to simulate a cash register for a single purchase. The program should read the unit cost (real number) of an item and the numbers of items purchased. The program should display the total cost for the items. **If the unit cost is greater than 10000, the program should display an error message, 'Invalid unit cost – too large**.'

If the unit cost is 0 or a negative number it should display an error message, 'Unit cost must be greater than zero'.

```
Enter unit cost: 5.5
Enter number of units: 10
Total cost: 55.0


Enter unit cost: 0
Unit cost must be greater than  0
```

6. Write a program to show a menu of areas to be calculated and to calculate the area chosen by the user. The output you are to display, is shown in italics below

   Choose the area you wish to calculate from the menu below

```
Compute Area of one of the following:

     s      for the area of a square
     c      for the area of a circle
     r      for the area of a rectangle

     Enter your choice: r

     Enter length: 4
     Enter breadth: 5

     Area of rectangle is: 20.0
```

The program should then prompt for the dimensions of the area:

length of a side in the case of a. square (area = length **2)

radius in the case of a circle (area = 3.14 * radius**2)

length and breadth in the case of a rectangle (area = length * breadth).

7. Write a program to read two numbers and display which is the largest and smallest of the numbers entered.

```
Enter first number: 4
Enter second number: 5

5 is the largest number
4 is the smallest number
```

Lesson 4 Assignments

1. Write a program that prompts the user to enter their exam score (out of 100). If the score is 90 or above, print 'You got an A!' If the score is between 80 and 89, print 'You got a B.' If the score is between 70 and 79, print 'You got a C.'

2. Write a program to calculate how much someone gets paid per week based on the number of hours they work per week. The program asks the user to enter the number of hours worked and the rate per hour and then displays the total pay

   The program must check that the number of hours worked does not exceed 100 and display an appropriate message if this is the case. It must also check that the rate per hour does not exceed 25. It should also check that the above numbers are greater than zero e.g.
   ```
   Enter number of hours worked: 120
   Number of hours too large
   Enter number of hours worked: -5
   Number of hours must be greater than 0
   ```
3. Write a program to play a guessing game. Th eprogram 'knows' a secret word e.g. 'car'. The user is asked to guess the secret word and an appropriate message is displayed:
   ```
   Guess the secret word: blue
   blue is not the secret word ! Try again !

   Guess the secret word: car
   Well done - you guessed it!
   ```
4. Write a program to read **three** numbers and display which is the largest and smallest of the numbers entered.

   ```
   Enter first number: 4
   Enter second number: 7
   Enter third number: 1

   7 is the largest number
   1 is the smallest number
   ```

5. Write a program that prompts the user to enter an exam score (out of 100). If the score is 90 or above, display 'You got an A'. If the score is between 80 and 89, display 'You got a 'B'. If the score is between 70 and 79, display 'You got a C'. If the score is between 60 and 69, display 'You got a D'. If the score is below 60, display 'You got an E'.

Lesson 5: Loops - `while` statement

We often wish to repeat one or more statements in a program. This is called **looping** or repetition.

There are a number of looping techniques, but basically all program looping can be performed using a **while** loop.

Loops are another form of **conditional statement**. In the case of a loop, we use the condition to decide whether to repeat a statement(s) or not.

```
while condition:
    action statement(s)    # loop body

rest of program statements
```

The action statement(s) is only carried out if the condition is true in the same way as for an `if` statement.

The action statement(s) of a loop is referred to as the **loop body**. This may be a single statement or a group of statements.

After executing the loop body, the loop condition is tested again.

If the condition is still true, we execute the loop body and test the condition again.

This process continues until the condition evaluates to false.

When the loop condition evaluates to false, then the loop body is skipped and the *`rest of program statements`* are executed.

The loop body statements may never be executed – this happens if the loop condition evaluates to false the first time the `while` statement is executed.

Example L5.1

Modify the calculator program to sum pairs of numbers, until the user enters 0 as one of the numbers. We read in the two numbers to be summed, calculate the sum and display the result. We repeat these steps until the user enters 0.

We use a `while` loop to repeat the necessary statements:

```
# calc4.py: Repeat adding 2 numbers until user enters 0

n1 = 1       # Assign non-zero so that we can start the loop
n2 = 1

while (n1 != 0) and (n2 != 0) :
    n1 = float(input('\nEnter first number [0 to quit]: '))
    n2 = float(input('\nEnter second number [0 to quit]: '))
    sum = n1 + n2
    print(f'\nThe sum of {n1} and {n2} is {sum} \n\n')

print ('\n\nFinished summing\n')
```

The loop body is highlighted in blue. The loop body statements are repeated until the user enters 0 for one of the numbers, as shown below.

When the loop condition evaluates to false, the loop terminates and the first statement after the loop body is executed – here it is a `print` to display that the program is finished.

```
Enter first number:  4
Enter second number: 6
The sum of 4.0 and 6.0 is 10.0

Enter first number:  20
Enter second number: 30
The sum of 20.0 and 30.0 is 50.0

Enter first number:  0
Enter second number: 6
The sum of 0.0 and 6.0 is 6.0

Finished summing
```

How Many Loop Iterations ?

The user may wish to sum 1 pair of numbers or 100 pairs. The user indicates if they wish to finish by entering 0 for the one of the numbers. This type of loop is called aa **non-deterministic loop**, as you do not know in advance how many times it will be repeated.

In the next example we specify how many times we want to repeat the loop body. We call this a counting loop. It is also called a **deterministic** loop, as it is *determined* in advance how many repetitions (iterations) to carry out that is how many times we repeat (iterate) the loop body.

Example L5.2

Modify the L5.1 to sum **three** pairs of numbers. In other words we wish to read in the two numbers to be summed, calculate the sum and display the result, **three** times.

We sometime call such a loop a counting loop.

```
# calc5.py: Calculator program to add 2 numbers, 3 times

count = 1

while count <= 3:
    n1 = float(input('\nEnter first number: '))
    n2 = float(input('\nEnter second number: '))
    sum = n1 + n2
    print(f'\nThe sum of {n1} and {n2} is {sum} \n\n')
    count = count + 1

print ('Finished summing\n')
```

`calc5.py` outputs:

```
>>> %Run calc4.py

Enter first number: 1
Enter second number: 2

The sum of 1.0 and 2.0 is 3.0

Enter first number: 3
Enter second number: 4

The sum of 3.0 and 4.0 is 7.0

Enter first number: 4
Enter second number: 5

The sum of 4.0 and 5.0 is 9.0

Finished summing
```

The `while` statement tests the condition `( count <= 4 )` and if it evaluates to true, the statements in the loop body are executed and the condition is re-evaluated.

We assigned `count` the value 1 which is called initialising `count`. When we first assign a value to a variable, we say we have initialised the variable.

When we run the program, `count` has the value 1, and so the condition evaluates to true and the loop body is executed.

Inside the loop body we increase `count` by 1, so it has the value 2 after the first iteration of the loop i.e. the first time we carry out the loop body statements.

The loop condition is then tested again and since `count` has the value 2, the loop body will be executed again, increasing count to 3.

We repeat this process until `count` has the value 5. Now the loop condition evaluates to false (`count` is no longer <= 3) and the loop is finished.

When the condition is false i.e. when `count` reaches 4, we skip the actions specified by the loop body, and in this example, we execute the final `print` statement and the program terminates.

The variable `count` is used in this example to control how many times we execute the loop body. Such a variable is called a **loop counter**.

Each time we execute the loop body (go around the loop), we process one pair of numbers and **add 1 to `count`**.

So after executing the loop 3 times, `count` will have the value 4. Each time you execute the loop, the condition is tested. You only execute the loop body if the result is true. So when `count` has value 4, we leave the loop (the loop terminates), i.e. we go to the next statement after the loop body if any.

What would happen if we omitted the statement

```
count = count + 1
```

from the loop body?

This is a very common error to make when using counting loops. If we omit the statement to increment count, the **loop will never terminate**, **as count will always be less than 5**. It is an example of an **infinite** or **endless loop**.

An endless loop may be terminated by interrupting the program or switching off the computer, both of which terminate the program as. To interrupt a program, a combination of keys is pressed, such as pressing the control key and the C key simultaneously (denoted by Ctrl/C).

Such an error is a **logical or runtime error**. These differ from syntax errors because the program can be executed but produces incorrect results.

For this reason, they are more serious than syntax errors. In large programs, it is very difficult to ensure that there are no logical errors. Thorough testing of programs may increase our confidence that a program is correct, but such **testing on its own, can never establish the correctness of a program**. It is important to bear this fact in mind and it is worthwhile investigating the area of program correctness.

Example L5.3

Write a program to sum the integers 1 to 99 (i.e. calculate the sum of 1+2+3+...+99) and display the result.

```
# sum.py: calculate 1+2+3+.....+99

sum = 0                # contains the sum we wish to compute
i = 1                  # the loop counter

while i <= 99:
     sum = sum + i
     i = i + 1

print(f'\nSum of 1 to 99 is: {sum}\n' )
```

Executing this program produces as output:

```
Sum of 1 to 99 is: 4950
```

The loop body is executed only if the condition `(i <= 99)` evaluates to true. Since we have initialised `i` to 1, the condition evaluates to true and the loop body is executed.

In the loop body, a running total for `sum` is calculated by adding the value of i to `sum`. The variable `sum` is assigned the value `sum + i`. The variable `i` is then increased by 1.

We then test the condition again. The variable i now has the value 2 and the condition `(i <= 99)` remains true so we execute the loop body assigning `sum` the value 3 (1+2) and increasing `i` to 3.

Next time around the loop, `sum` becomes 6 (3+3) and `i` becomes 4. We test the condition again and continue in this manner until `i` eventually reaches the value 100.

When we test the condition in this case, it evaluates to false (i.e. `i` is greater than 99) and so the loop body is not executed. Instead we continue at the first statement after the loop body i.e. the `print` statement.

Example L5.4

Sometimes it is useful to put a `print` in the loop body so you can see what's happening and also to get a better understanding of looping.

```
# sum2.py: calculate the sum of 1 to 9

sum = 0               # contains the sum we wish to compute
i = 1                 # the loop counter

while i <= 9:
     sum = sum + i
     print(f'\nSum = {sum} i = {i}) # display what's happening
     i = i + 1

print(f'\nSum of 1 to 9 is: {sum}\n' )
```

Executing this program produces as output:

```
Sum =  1  i =  1

Sum =  3  i =  2

Sum =  6  i =  3

Sum =  10  i =  4

Sum =  15  i =  5

Sum =  21  i =  6

Sum =  28  i =  7

Sum =  36  i =  8

Sum =  45  i =  9

Sum of 1 to 9 is: 45
```

Programmers often use the **short variable names** **`i, j, k,`** and so on, as loop counters.

Variable Initialisation

In the last two examples it is crucial that the variables `count` and `i` are initialised to appropriate values for the loop to operate correctly. As a general programming principle, all variables should be initialised to appropriate values, usually at the beginning of a program.

The code for Example L5.1 can be improved. L5.1 does stop after 0 has been input for the first number, it still reads the second number and adds it to 0 and displays that result. We want the loop to stop after 0 has been entered as either first or second number. We also do not want to be asked to enter the second number, if the first one was 0.:

The program below is an improved version:

Example L5.5

```
# calc6.py: Calculator program to add 2 numbers until 0 entered

n1 = 1     # Assign non-zero so that we can start the loop
n2 = 1

while (n1 != 0) and (n2 != 0) :
    n1 = float(input('\nEnter first number [0 to quit]: '))
    if ( n1 != 0 ):
        n2 = float(input('\nEnter second number [0 to quit]: '))
        sum = n1 + n2
        if (n2 != 0):
            print(f'\nThe sum of {n1} and {n2} is {sum} \n\n')

print ('\n\nFinished summing\n')
```

This program runs as follows:

```
Enter first number: 3

Enter second number: 3

The sum of 3.0 and 3.0 is 6.0

Enter first number: 0

Finished summing
```

And again

```
Enter first number: 3

Enter second number: 0

Finished summing
```

As you can see from the above program, you can use any statement in the loop body including more conditionals. This time, once we read the 1st number, we check if it is 0 and only if it is not 0, will we read the 2nd number. We only calculate the sum and display the result if the second number is not 0.

Example L5.6

Write a program to convert dollars to kyats using an exchange rate of 1 dollar = 2100 kyats. Allow the user to keep entering vales until 0 is entered to quit.

```
# convert_dollars.py: convert dollars to kyats until 0 entered

dollars = float(input('\nDollars [0 to quit]: '))

while ( dollars != 0 ):
     kyats = dollars * 2100
     print(f'{dollars} dollars = {kyats:.2f} kyats ' )
     dollars = float(input('\nDollars [0 to quit]: '))
print(f'\nFinished converting\n')
```

L5.6 outputs:

```
Dollars [0 to quit]: 7
7.0 dollars = 14700.0 kyats

Dollars [0 to quit]: 0

Finished converting
```

Debugging with Loops

As mentioned earlier, if you have difficulty understanding loops, it is a good idea when you implement any of the loop programs to put a `print` statement in the loop body, so that you can see how what is happening as the loop is repeated. For example in L5.2 the following `print` could be inserted in the loop body:

```
print (f'\ncount = {count}') # display count as loop runs
```

This is also a useful debugging technique. Debugging is the term used for finding and correcting errors (bugs) in your program.

By placing `print` statements in your code, you can **trace** (follow) the execution of your program, inspecting the values of variables and checking if loops are executed the correct number of times.

A `print` in the action part of an if statement allows you verify that the action was indeed carried out. When your program is working correctly, these debugging `print` statements are removed.

The **`break`** statement

Sometimes we wish to terminate a `while` loop without having to wait for the lop condition to become false. We use the **`break`** statement to do this. It stops the loop and the program continues at the first statement after the loop body the loop.

Example L5.7

A guessing game program. The user has to try to guess a "secret" word built into the program.

```
# guess2.py: Guess the secret word

secret = 'blue'
guess = ' '

while (guess != secret) and (guess != 'quit'):

     guess = input('Guess the secret word:[quit to finish]  ')

     if guess == 'quit':
          break                    # Exit the loop
     if guess != secret:
          print(f'\nWrong guess: {guess}')
     else:
          print(f'\nWell done !')  # end of loop

if (guess == 'quit'):
     print(f'\nThe secret word was: {secret}')
```

If the user enters `'quit'` then the **`break`** statement terminates the loop and the first statement after the loop body is executed i.e. the `print` to display the secret word.
Note: The loop in this program can terminate in **two** ways. It will terminate if the loop condition is false (for example the user guesses the word) OR if the user enters `'quit'`.

This means that when the loop terminates, we need to check if it was because the user entered `'quit'`. And display the appropriate message in that case.

```
Running guess2.py:


Guess the secret word:  man
Wrong guess:  man
Guess the secret word:  dog
Wrong guess:  dog
Guess the secret word:  quit
The secret word was:  blue


Running guess3.py:
Guess the secret word:  black
Wrong guess:  black
Guess the secret word:  blue
Well done !
```

Example L5.8

A guessing game program with limited number of guesses. The user has to guess a "secret" word built into the program but has **only 3 chances** to guess it.

Algorithm for this guessing game program

We explained the concept of an algorithm earlier. It is the set of set of steps to solve a problem. We usually write algorithms in what is called pseudo code. This is a cross between English and programming language statements. There is no defined version of pseudo code, so you can make up your own version.

In my pseudo code, I use the word **`repeat until`** … **`end repeat`** for a loop. It can be read as "repeat the statements from `repeat` to `end repeat` while the condition is true. In the example below the loop body is highlighted in blue.

Because we are now using conditionals (`if` and `while`) our programs are becoming longer and more complex. So it is a good idea to develop an algorithm for your program before writing the actual code.

```
# Guessing game algorithm

Set number of guesses to 1
Set guess to blank
Set the secret word in the program

Repeat until guess is correct, or quit or number guess > 3
     Ask the user to guess the word or quit
     If guess is 'quit'
          Exit the loop
     If guess is incorrect then
          Display error message
          Add 1 to number of guesses
     Else
          Display Correct guess message
End repeat

If guess is quit
     Display quit message
Else
     Display too many guesses message

Program terminates
```

We now implement the algorithm in Python.

```
# L5.8: guess3.py: Guess the secret word in 3 guesses

secret = 'blue'
guess = ' '
num_g = 1           # number of guesses

while (guess != secret) and (guess != 'quit') and (num_g < 4):
     guess = input('Guess the secret word:[quit to finish]  ')
     if guess == 'quit':
          break                   # Exit the loop
     if guess != secret:
          print(f'\nWrong guess: {guess}')
          num_g = num_g + 1
     else:
          print(f'\nWell done !')  # end of loop

if (guess == 'quit'):
     print(f'\nThe secret word was: {secret}')
else:
     print(f'\n Sorry you have used 3 guesses')
     print(f'\nThe secret word was: {secret}')
```

```
Running guess3.py:


Guess the secret word:  man
Wrong guess:  man
Guess the secret word:  dog
Wrong guess:  dog
Guess the secret word:  red
Sorry you have used 3 guesses')

The secret word was: blue
```

```
Running guess3.py:
Guess the secret word:  black
Wrong guess:  black
Guess the secret word:  blue
Well done !
```

Nested Loops

A loop may contain as part of its loop body any statement including another loop. A loop inside the body of a loop is called an **inner loop** or **nested loop.** The nested loop may in turn contain a loop as part of its loop body and so on.

Example 5.9: Write a program to read in the marks for a group students and display the average mark for each student based on the entered marks. There are 3 grades for each student. The programs allows the user enter as many students as they wish, finishing when the name `'quit'` is entered.

Algorithm

```
Read name
Repeat until name is quit
     Set sum to 0
     Set number of marks to 1
     Repeat until number of marks > 3   # nested loop
          Read mark
          Add mark to sum
          Add 1 to number of marks
     End repeat                         # end of nested loop

     Compute average - = sum / 3
     Display Average mark for name
     Read next name
End repeat                    # end of outer loop
Display finished message
```

```
# L5.9 average.py: Compute average nark for students
# There are 3 marks for each student

name = input('\nEnter name: [quit] :')

while ( name != 'quit' ):
    nm = 1              # number of marks entered
    sum = 0.0
    while ( nm <= 3 ):
        mark = float(input(f'Enter mark {nm}: '))
        sum = sum + mark
        nm = nm + 1          #end of inner loop
    average = sum / 3

    print(f'Average mark for {name} : {average:.2f}' )

    name = input('\nEnter name: [quit] : ')
    # end of outer loop

print(f'\nFinished \n')
```

L5.9 runs as follows

```
Enter name: [quit] : Joe
Enter mark 1: 50
Enter mark 2: 60
Enter mark 3: 70
Average mark for Joe : 60.00

Enter name: [quit] : Mary
Enter mark 1: 70
Enter mark 2: 80
Enter mark 3: 85
Average mark for Mary : 78.33

Enter name: [quit] : quit

Finished
```

Note the use of an **f-string** in the `input` statement:

```
         mark = float(input(f'Enter mark {nm}: '))
```

This allows us display which of the three marks is being entered (1, 2, or 3) as shown in the output above.

Example 5.10:

Use a nested loop to display triangle made of stars (*). Display 4 lines so that:

1 star is displayed on line 1;
2 stars displayed on line 2,
3 stars displayed on line 3 and
4 stars displayed on line 4.

The output appears as follows which is the shape of a triangle made of stars:

```
*
**
***
****
```

```
# tri.py: displays triangle composed of *'s
# This program does NOT work

num_lines = 1

while num_lines <= 4:
    num_stars = 1
    while num_stars <= num_lines:      # inner loop
        print(f'*')
        num_stars = num_stars + 1                 #end inner loop

    print('\n')             # start new line
    num_lines = num_lines + 1      # end outer loop
```

The inner loop displays the correct number of * characters on each line. The outer loop controls the number of lines displayed.

However, this program does not work as intended. It displays the stars on separate lines:

```
*

*
*


*
*
*


*
*
*
*
```

This is because the `print` function adds the newline character at the end of its output. To stop `print` doing this, we add a new element called **end** to `print`, as follows:

```
print(f'*', end = '') # instructs print not to output newline
```

There is no space between the quotes in **`end = ''`**

We use this version of `print` in the L5.11 below and we get the following output:

```
*

**

***

****
```

Example 5.11

This is the amended version of L5.11 to display a triangle.

```
# tri2.py: displays triangle composed of *'s

num_lines = 1

while num_lines <= 4:
    num_stars = 1
    while num_stars <= num_lines:      # inner loop
        print(f'*', end = '')
        num_stars = num_stars + 1      #end inner loop

    print('\n')             # start new line
    num_lines = num_lines + 1      # end outer loop
```

`while` loop summary

We use the while loop to repat statements. From the examples above we can see two common ways it is used

1. **Repeat statements until user enters data to indicate they are finished (e.g. 'quit', 0)**

```
name = input('\nEnter name: [quit] :')

while ( name != 'quit' ):
    statement1
    statement2
    # as many statements as you wish

    name = input('\nEnter name: [quit] : ')

# rest of program after loop
```

2. **Repeat statements a fixed number of times – counting loop.**

```
    i = 0           # counter
    n = 10          # number of iterations, can be read in etc

    while ( i < n ):
        statement1
        statement2
        # as many statements as you wish
        i = i + 1   #increase loop counter

# rest of program after loop
```

We often count from 0 in programming. Thus to repeat the above loop 10 times the counter **`i`** has the value 0, 1, 2, ,3, 4, 5, 6, 7, 8, and 9 as we go around the loop. When **`i`** becomes 10, the loop terminates because 0 to 9 is 10 iterations.

Lesson 5 Exercises

1. Will the following loop finish ?

```
j = 0
while j < 10:
     print(f'j =  {j}')
j = j + 1
```

2. Display the text `'Hello World'` on 5 separate lines using a `while` loop.

3. Write a program that uses a while loop to display the numbers 1 to 10 on separate lines.

4. Write a program that uses a `while` loop to sum the numbers: 2, 4, 6, 8, 10, 12, 14,16,18 and 20. The program then displays the sum of the numbers. [Hint: The loop counter is increased in increments of 2].

5. Write a program that reads numbers entered by the user, **until the user enters 0**. The program computes the sum and average of the numbers. The program then displays the sum and average.

6. Write a program to show a menu of areas and to calculate the area chosen by the user. After each area has been calculated, the program displays the menu again. The program continues until the **user chooses the 'x'** option. The output is shown in italics below

```
Choose one of the following options:

     s      for the area of a square
     c      for the area of a circle
     r      for the area of a rectangle

     x      Exit program

     Enter your choice: r
     Enter length: 4
     Enter breadth: 5

     Area of rectangle is: 20.0

Compute Area of one of the following:

     s      for the area of a square
     c      for the area of a circle
     r      for the area of a rectangle

     Enter your choice: x

     Area calculation program finished
```

Lesson 5 Assignments

1. Write a program that prompts the user to enter their exam score (out of 100). If the score is 90 or above, display 'You got an A!' If the score is between 80 and 89, display 'You got a B.' If the score is between 70 and 79, display 'You got a C. The programs continues the above process **until the user enters a score of 0**

2. Write a program to keep calculating how much someone gets paid per week until the user enters 'quit'. The program asks the user to enter the number of hours worked and the rate per hour and then displays the total pay.
   The program must check that the number of hours worked does not exceed 100 and that the rate per hour does not exceed 25.

3. Modify Example L5.9 to allow the user enter as many grades as they wish for each student, finishing when the mark 0 is entered for that student. It then displays the average mark for that student as in L5.9. The programs allows the user enter marks for as many students as they wish, finishing when the name `'quit'` is entered.

4. Write a program to display 6 lines with
   5 Spaces followed by 1 star on line 1
   4 spaces followed by 2 stars on line 2
   3 spaces followed by 3 stars on line 3
   2 spaces followed by 4 stars on line 4
   1 spaces followed by 5 stars on line 5
   0 spaces followed by 6 stars on line 6

   The output should appear as follows:

```
     *
    **
   ***
  ****
 *****
******
```

5. Modify program 3 above to output a what looks like a “tree” as follows

```
    *
   ***
 *******
*********
***********
```

Appendix 1: Solutions

Lesson 1 Solutions

1. What is the output of the following print statement?

```
print( 'Have a great day!')
```

c. Have a great day!

a. What is the output of the following statements?

```
print('Hi there!')
print('How are you doing?')
```

```
Hi there!
How are you doing?
```

b. Write a program that prints a message saying

I love Python!

```
print('I love Python! ')
```

c. Write a program that prints a message saying your name and your age, e.g.

My name is Colin. I am 20 years old!

```
print('My name is Colin. I am 20 years old! ')
```

d. Write a program to display the message 'Welcome to Python' three times, on separate lines using three `print` statements.

```
print('Welcome to Python! ')

print('Welcome to Python! ')

print('Welcome to Python! ')
```

e. Write a program to display the message 'Python is awesome!' two times, on separate lines, using only **one** `print` statement and \n

```
print('Python is awesome!\n Python is awesome!\n ')
```

f. What are the syntax errors in the following statements:

```
print( 'Hello ! Goodbye! )- missing closing '
print( Hello ! Goodbye!' ) – missing opening '
print( 'Hello ! Goodbye!'  - missing closing )
print 'Hello ! Goodbye!' ). – missing opening (
prlnt( 'Hello ! Goodbye!' ) – misspelt print
```

Lesson 2 Solutions

1. Valid or invalid variable names
   a. Is the variable name `TotalMarks` correct - **Yes**
   b. Is the variable name `number-of-students correct?` **NO** – cannot use –in variable name
   c. Is the variable name `firstName` correct? **Yes**
   d. Is the variable name `myVar1` correct? **Yes**
   e. Is the variable name `customerName` correct? **Yes**
   f. Is the variable name `productPrice` correct? **Yes**
   g. Is the variable name 3`rdStudent` correct? **NO** – cannot start with a digit
   h. Is the variable name `isAvailable`? correct? Yes
   i. Is the variable name `total-sales` correct? **NO** – cannot use –in variable name
   j. Is the variable name `customer_email` correct? **Yes**

2. Write a program that asks the user for their name using `input`. Store the name in a variable and display a personalized greeting using the variable.

```
name = input('Enter your name: ')

print('Hello, how are you ', name )
```

3. What is the output, if any, of the following program:

```
# print('hello\n')
# print('bye bye\n')
```

   No output because any text following # is treated as a comment and ignored by Python

4. Write a program that prompts the user to enter their favourite colour and favourite animal using the `input`. Store these values in separate variables and display them in a sentence :

```
f_colour = input('Enter your favourite colour: ')
f_animal = input('Enter your favourite animal: ')
```

```
    print('My fav colour is ', f_colour, 'and my fav animal is',
f_animal )
```

Lesson 3 Solutions

Q1:

a. The data type of the variable 'age' is integer.

b. The data type of the variable 'name' is string.

c. The data type of the variable 'price' is float.

d. The data type of the variable 'is_valid' is boolean.

e. The data type of the variable 'quantity' is integer.

f. The data type of the variable 'message' is string.

g. The data type of the variable 'discount' is float.

2. Write a program to convert 10 dollars to kyats using an exchange rate of 1 dollar = 2100 kyats.

```
# convert 10 dollars to kyats

dollars = 10
kyats = dollars * 2100
print(f '{dollars} dollars = {kyats} kyats ' )
```

9. Write a program that takes a single length (a float) and calculates the following:
    - The area of a square with side of that length. (length * length)
    - The volume of a cube with side of that length. (length ** 3)
    - The area of a circle with diameter of that length (3.14 * (length/2)**2))

```
# calculate area of square, volume of cube and area of circle

length = float(input('Enter length: '))

area_of_square  = length * length
cube_volume = length ** 3
area_of_circle = 3.14 * ((length / 2)**2)
print(f' Area of square: {area_of_square:.2f}' )
print(f' Volume of cube: {cube_volume:.2f}' )
print(f' Area of circle: {area_of_circle:.2f}' )
```

7. Write a program that takes an amount (a float), and calculates the tax due according to a tax rate of 20%

```
# calculate tax due at 20%

amount = float(input('Enter amount for tax at 20%: '))

tax = amount * 0.20
print(f'Tax: {tax:.2f}' )
```

8. Write a program to simulate a cash register for a single purchase. The program reads the unit cost of an item and the numbers of items purchased. The program displays the total cost for that number of units:

```
Enter unit cost: 5
Enter number of units: 6

Total cost of 6 units: 30.00
```

```
# calculate total cost as number of unit * unit cost

unit_cost  = float(input('Enter unit cost: '))

number_units  = float(input('Enter number of units: '))

total = unit_cost * number_units

print(f'\nTotal cost of {number_units} units: {total:.2f}')
```

Lesson 4 Solutions

1. What are the 6 conditions that we can use to compare two numbers?
   See Lesson 4 in Handbook

2. Write a program that asks the user to enter their exam score. If the score is greater than or equal to 60, display 'Congratulations! You passed the exam'. Otherwise, display 'Sorry, you did not pass the exam'.
```
score = int(input('Enter your exam score: '))
if score >= 60:
     print('Congratulations! You passed the exam.')
else:
     print('Sorry, you did not pass the exam.')
```

3. Write a program that asks the user to enter a password. If the password is 'password123', display 'Access granted' Otherwise, display 'Access denied'
```
     password = input('Enter your password: ')
     if password == 'password123':
          print('Access granted')
     else:
          print('Access denied')
```
4. Write a program that prompts the user to enter their age and whether they have a driver's license ('yes' or 'no). If the person is 18 or older and has a driver's license, display 'You can legally drive'.
   If the person is 18 or older but does not have a driver's license, display 'You can apply for a driver's license'.
   If the person is under 18, display 'You are not old enough to drive'.
```
# Prompt the user to enter their age

age = int (input('Enter your age: '))

# Prompt whether they have a driver's license

has_license = input('Do you have a drivers license? [yes/no]: 
')

# Check the driving eligibility

if (age >= 18) and (has_license == 'yes'):
    print('You can legally drive.')

if (age >= 18) and (has_license == 'no'):
    print('You can apply for a drivers licence')

if (age < 18):
    print ('You are not old enough to drive.')
```

5. Write a program to simulate a cash register for a single purchase. The program should read the unit cost (real number) of an item and the numbers of items purchased. The program should display the total cost for the items. If the unit cost is greater than 10000, the program should display an error message, 'Invalid unit cost – too large.'
   If the number of units is 0 or a negative number it should display an error message, 'Number of units must be greater than zero'.

```
# cash.py: Calculate and display bill for items purchased

unit_cost = float(input('\n Unit cost: '))

if unit_cost > 10000:
    print(f'\n Unit cost {unit_cost} cannot exceed 10000: ')
else:

    num_units = float(input('\n Number of units: '))

    if num_units <= 0:
         print(f'\n Num of units {num_units} must be > 0')
    else:

        cost = num_units * unit_cost
        print(f'\n Total cost: {cost} ')
```

6. Write a program to show a menu of areas to be calculated and to calculate the area chosen by the user.

```
# calculate areas giving user options in a menu

print(f'    s      for the area of a square \n')
print(f'    c      for the area of a circle \n')
print(f'    r      for the area of a rectangle \n')

shape = input('\n Enter you choice [s, c, r]   ')

if ( shape == 's') :
    length = float(input(' Enter length: '))
    area_of_square  = length * length
    print(f' Area of square: {area_of_square:.2f}' )

if (shape == 'c'):
    radius = float(input(' Enter radius: '))
    area_of_circle = 3.14 * radius ** 2
    print(f' Area of circle: {area_of_circle:.2f}' )

if (shape == 'r'):
    length = float(input(' Enter length: '))
    breadth = float(input(' Enter length: '))
    area_of_rectangle = length * breadth
    print(f' Area of rectangle: {area_of_rectangle:.2f}' )
```

7. Write a program to read two numbers and display which is the largest and smallest of the numbers entered.

```
# Find largest and smallest of two numbers

n1  = float(input('First number: '))

n2  = float(input('Second number: '))

if n1 > n2 :
     large = n1
     small = n2
if n1 < n2 :
     large = n2
     small = n1
if n1 == n2 :
     print(f' First number {n1} = Second number {n2} \n')
else :
     print(f' Largest is {large} and smallest is {small} \n')
```

Lesson 5 Solutions

1. Will the following loop finish ?

   ```
   j = 0
   while j < 10:
        print(f'j =  {j}')
   j = j + 1
   ```

   No! The loop body will continue printing the message, because `j` is not increased in the loop body.

   ```
        j = 0
        j = 0
   ```

   The statement `j = j + 1` is not indented, so it is not part of the loop body. This is a common error.

2. Display the text `'Hello World'` on 5 separate lines using a `while` loop.

   ```
   j = 0
   while j < 5:
        print(f'Hello World')
        j = j + 1
   ```

3. Write a program that uses a while loop to display the numbers 1 to 10 on separate lines.

   ```
   j = 0
   while j < 11:
        print(f'{j}')
        j = j + 1
   ```

4. Write a program that uses a `while` loop to sum the numbers: 2, 4, 6, 8, 10, 12, 14,16,18 and 20. The program then displays the sum of the numbers. [Hint: The loop counter is increased in increments of 2].

   ```
   sum = 0
   j = 2
   while j < 21:
        sum = sum + j
        j = j + 2

   print(f'Sum is {sum}')
   ```

5. Write a program that reads numbers entered by the user, **until the user enters 0**. The program computes the sum and average of the numbers. The program then displays the sum and average.

   ```
   sum = 0
   count = 0      # Number of numbers summed
   n = float(input('Enter a number: '))
   while n != 0:
        sum = sum + n
        count = county + 1
   ```

```
        n = float(input('Enter a number: '))

    if count > 0:
        average = sum / count
        print(f'Sum is {sum} and Average is {average}')
```

Note: We need to test `count > 0` because if the user enters 0 as the first number we do not display sum and average.

More importantly, it is an error in any programming language to divide a number by 0. If we try to divide a number by 0 the program will crash!

6. Write a program to show a menu of areas and to calculate the area chosen by the user. After each area has been calculated, the program displays the menu again. The program continues until the **user chooses the 'x'** option.

```
# Menu driven program to calculate areas

option = ''
while option != 'x':
    print(f'\n\n\nChoose one of the following options:\n')
    print(f's      for the area of a square\n')
    print(f'c      for the area of a circle\n')
    print(f'r      for the area of a rectangle\n\n')
    print(f'x      Exit program\n')

    option = input('Enter your choice: [s, c, r, x] ')
    if option == 'x':
        break

    if option == 's':
        length = float(input('Enter length: '))
        area_of_square  = length * length
        print(f'Area of square: {area_of_square:.2f}' )

    if option == 'c':
        radius = float(input('Enter radius: '))
        area_of_circle = 3.14 * ( radius **2)
        print(f'Area of circle: {area_of_circle:.2f}' )

    if option == 'r':
        length = float(input('Enter length: '))
        breadth = float(input('Enter breadth: '))
        area_of_rect= length * breadth
        print(f'Area of rect: {area_of_rect:.2f}' )
```