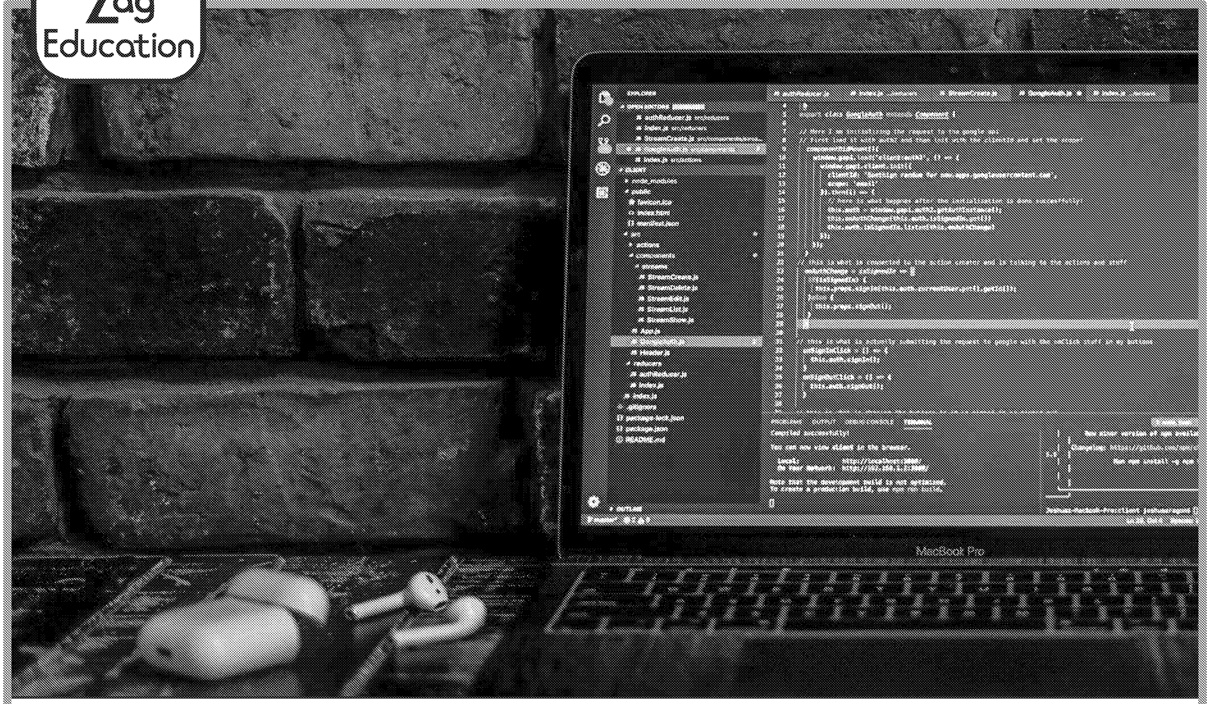
Computer Science GCSE / IGCSE



Python Programming for GCSE

Third Edition

Third Edition, September 2025

Download support files from zzed.uk/productsupport

zigzageducation.co.uk

POD 12854

Contents

Teacher's Introduction

The digital files for this resource are provided on the ZigZag Education Support Files system, which can be accessed via **zzed.uk/productsupport**

This resource provides a thorough introduction to the practical problem-solving and programming skills that students will need to develop for GCSE/IGCSE Computer Science. The resource introduces each topic using clear examples and annotated code snippets and a range of exercises for students to practise, with answers provided for each exercise.

As students develop their knowledge of Python, the tasks set become more challenging and demonstrate possible methods of approaching programming project-style tasks, with an emphasis on how to make the code both efficient and robust at every opportunity.

This resource also includes five stand-alone practical programming problems to stretch the most able students; suggested solutions are included for all problems together with any additional files required to attempt each task. In addition to a digital copy of the task solutions, this resource also includes a collection of 24 lesson starters in a PowerPoint presentation, with solutions.

With a range of starters and exercises available providing both stretch and challenge, this is the ideal resource for delivering GCSE programming in Python. Digital copies of all task solutions provide teachers with the option to customise and develop tasks if necessary. Students can use this resource independently, either to work through systematically or to refer to on an ad hoc basis when required; helping them to develop confidence and competence in their programming skills.

For information about the key conventions used, including PEP 8 guidelines and more, refer to the *Introductory Notes*.

Digital format

All of the activities are provided electronically on the ZigZag Education support files system, which can be accessed via **zzed.uk/productsupport** To use on a school network:

- Download the .zip folder
- Locate the .zip folder in your downloads folder
- Right-click on the .zip folder > click 'Extract all' > select a destination > click 'Extract'. **This step is essential as the files will not function properly without it.**

Access to the activities is via the following user interfaces:

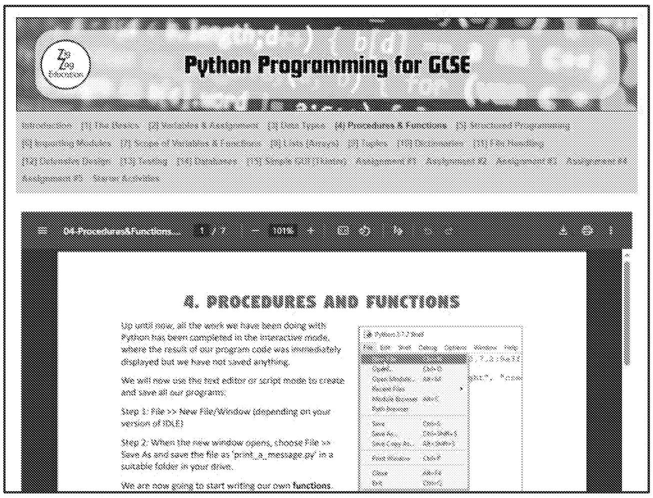
- ***index.html***
  This the main user interface for the resource; the one that your students will use.

  All of the topic notes/exercises, practice assignments and starter activities can be selected using the horizontal navigation bar.

  There are controls enabling zoom in/out, jump to page, table of contents and print.

- ***solutions***
  Worked solutions for every exercise in the resource are located inside the <u>*teacher*</u> subfolder.

  Note: there is no link to any of the solutions from the main interface. If you wish for students to be able to access this page, you will need to provide them with a link or shortcut to it.



Python Programming for GCSE

Introduction [1] The Basics [2] Variables & Assignment [3] Data Types [4] Procedures & Functions [5] Structured Programming [6] Importing Modules [7] Scope of Variables & Functions [8] Lists (Arrays) [9] Tuples [10] Dictionaries [11] File Handling [12] Defensive Design [13] Testing [14] Databases [15] Simple GUI (Tkinter) Assignment #1 Assignment #2 Assignment #3 Assignment #4 Assignment #5 Starter Activities

04-Procedures&Functions... 1 / 7 101%

4. PROCEDURES AND FUNCTIONS

Up until now, all the work we have been doing with Python has been completed in the interactive mode, where the result of our program code was immediately displayed but we have not saved anything.

We will now use the text editor or script mode to create and save all our programs:

Step 1: File >> New File/Window (depending on your version of IDLE)

Step 2: When the new window opens, choose File >> Save As and save the file as 'print_a_message.py' in a suitable folder in your drive.

We are now going to start writing our own **functions**.

To use on a *secure* school network: copy the *PythonProgramming* folder to a location that your students can access, and provide them with a link or shortcut to *index.html*. Repeat for the *teacher/solutions* folders to give access to the worked solutions.

INTRODUCTORY NOTES

WELCOME

Python Programming provides a comprehensive introduction to the proble programming skills that you will need for GCSE/IGCSE Computer Science.

There are 15 topic areas, each with explanations, examples and annotated range of exercises for you to practise.

As you develop your knowledge of Python, the tasks set become more cha possible methods of approaching programming project-style tasks, with an make the code both efficient and robust at every opportunity.

In addition, there are five stand-alone practical programming problems des abilit[illegible], as well as 24 starter activities.

KEY CONVENTIONS

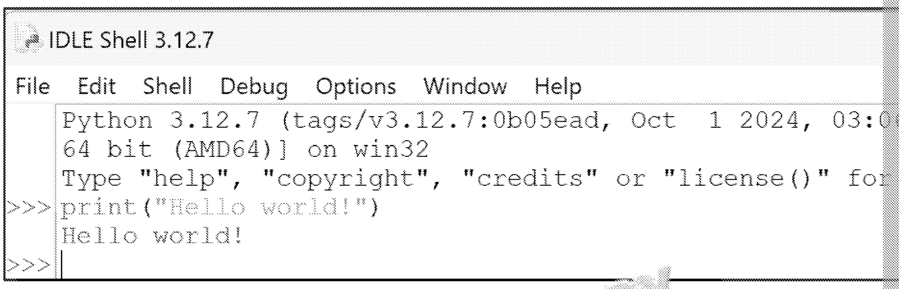
VERSION OF PYTHON

The version of Python that is used in all examples in this resource is versio

PYTHON MODES

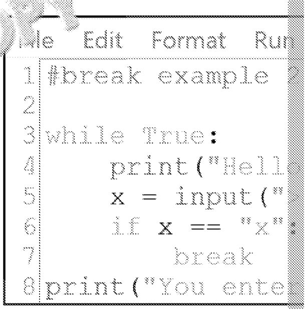
Python has two modes: *interactive* and *script*.

In *interactive* mode, any code we type into the code window will immediat the shell.



```
Python 3.12.7 (tags/v3.12.7:0b05ead, Oct  1 2024, 03:0
64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for
>>> print("Hello world!")
Hello world!
>>>
```

In *script* mode, there is no immediate response to our code. To view the results we must save our file and execute the program by pressing F5 or choosing Run >> Run Module.



```
#break example 2

while True:
    print("Hell
    x = input("
    if x == "x":
        break
print("You ente
```

IDE (INTEGRATED DEVELOPMENT ENVIRONMENT)

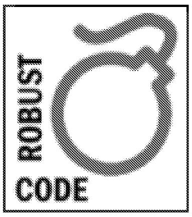
There are many different IDEs which students can use to understand how t problems within their code. This resource uses IDLE, the integrated develo written in Python and Tkinter, as it is freely available and easy to use.

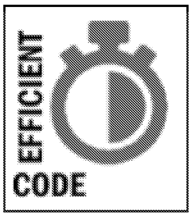
PYTHON AND SQL

In topic ***14: Databases***, the method of connecting to a simple database app which is imported as a module into Python 3. The SQLite browser is also us visual method of interacting with a simple database. This open source softw here: **https://sqlitebrowser.org/**

EFFICIENT AND ROBUST CODE

The ability to write efficient and robust code is an important part of learnin structure effective programs. Specific examples of efficient and robust cod these icons appear on the page:





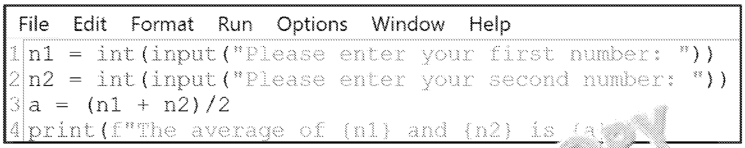
WRITING READABLE CODE – USING PEP 8 GUIDELINES

Readability involves the use of consistent styles and layout to make your co others to read and understand.

VARIABLE AND FUNCTION NAMES

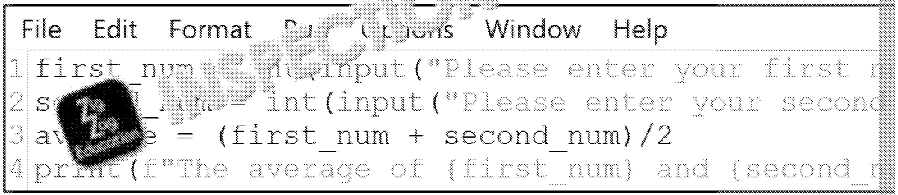
The key points to follow are the consistent use of lower case and undersco function names easier to read and make the purpose of the variable cleare

Example 1:



```
n1 = int(input("Please enter your first number: "))
n2 = int(input("Please enter your second number: "))
a = (n1 + n2)/2
print(f"The average of {n1} and {n2} is {a}
```

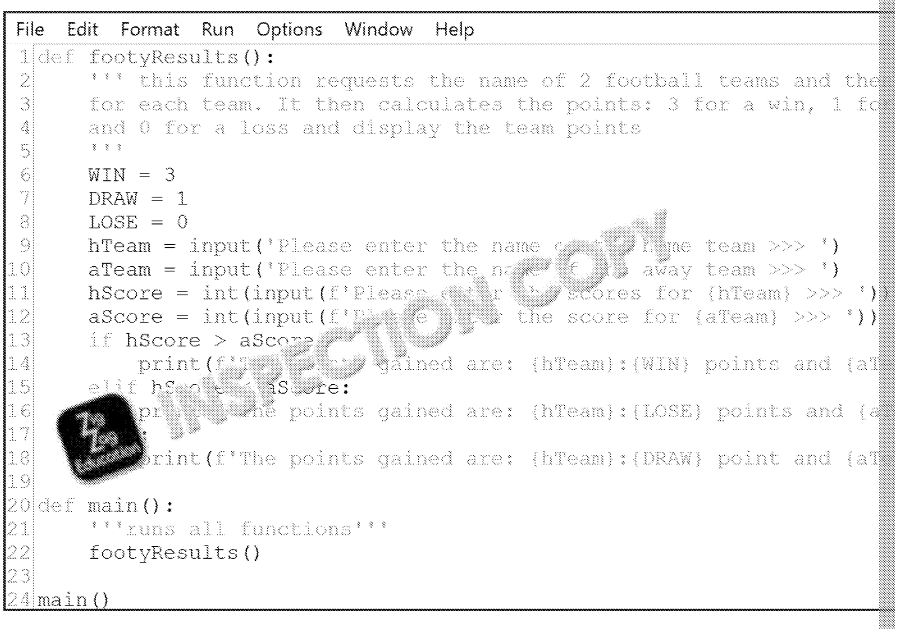
Following PEP 8 guidelines:



```
first_num = int(input("Please enter your first n
second_num = int(input("Please enter your second
average = (first_num + second_num)/2
print(f"The average of {first_num} and {second_n
```

The exception to this rule is the naming of variables which are constants, i. remains unchanged when the program is executed.

Example 2:
Constant names should be all upper case, separating words using an under number of points for winning, losing or drawing a football game does not c as constant values.

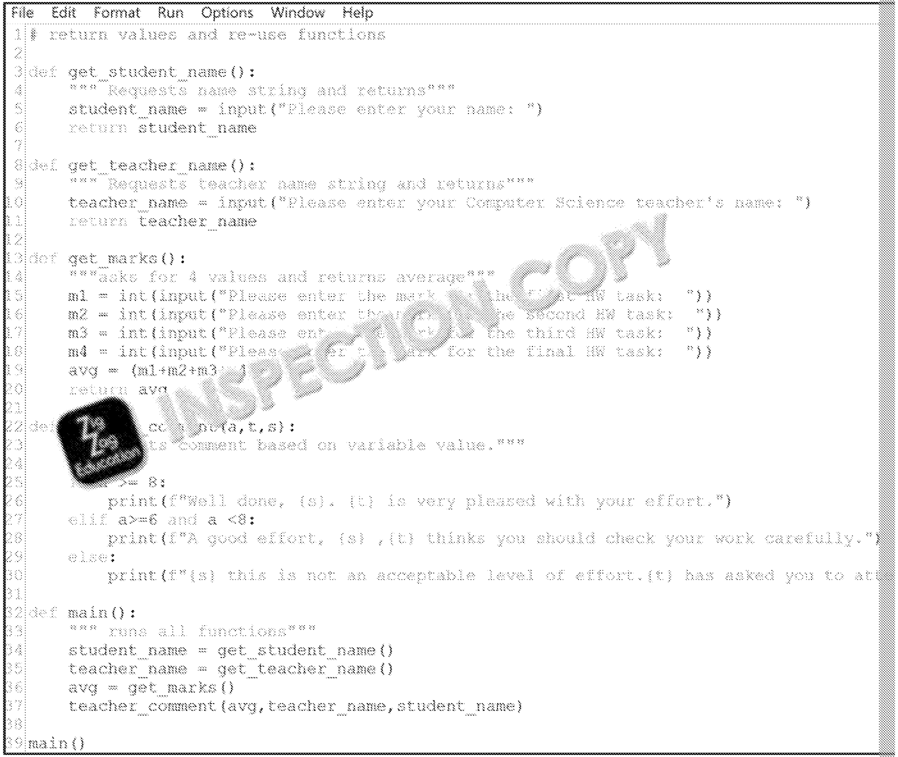


```
def footyResults():
    ''' this function requests the name of 2 football teams and the
    for each team. It then calculates the points: 3 for a win, 1 fo
    and 0 for a loss and display the team points
    '''
    WIN = 3
    DRAW = 1
    LOSE = 0
    hTeam = input('Please enter the name of the home team >>> ')
    aTeam = input('Please enter the name of the away team >>> ')
    hScore = int(input(f'Please enter the scores for {hTeam} >>> '))
    aScore = int(input(f'Please enter the score for {aTeam} >>> '))
    if hScore > aScore:
        print(f'The points gained are: {hTeam}:{WIN} points and {aT
    elif hScore < aScore:
        print(f'The points gained are: {hTeam}:{LOSE} points and {a
    else:
        print(f'The points gained are: {hTeam}:{DRAW} point and {aT

def main():
    '''runs all functions'''
    footyResults()

main()
```

USE OF WHITESPACE IN EXPRESSIONS AND STATEMEN

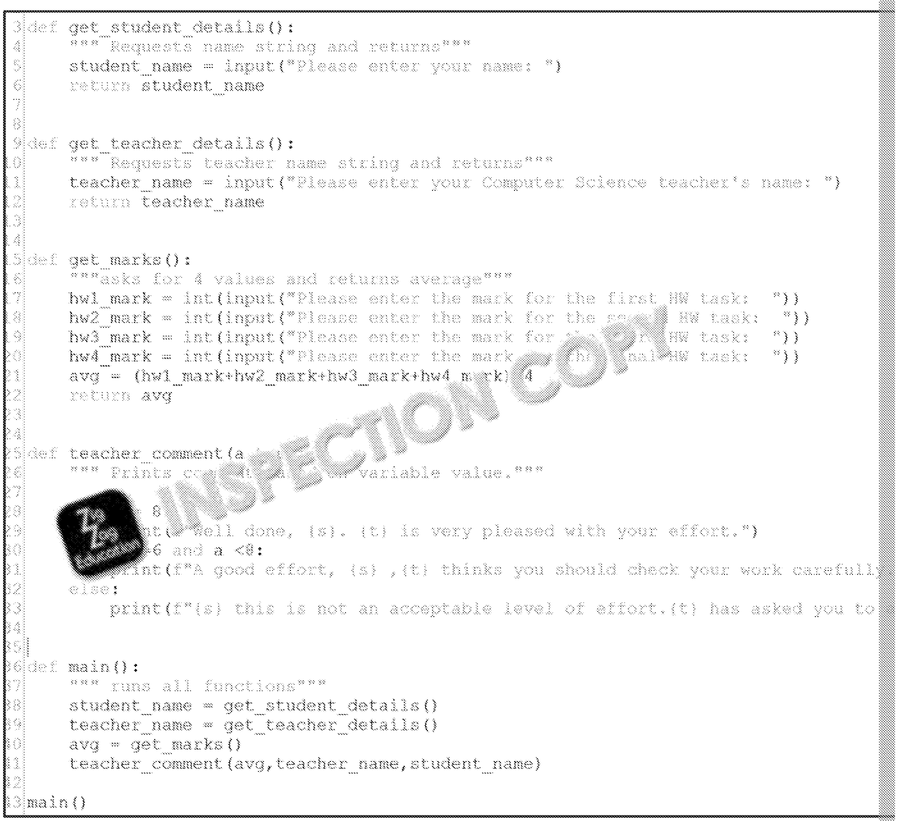
The key points to follow here are ensuring that there is a space either side of is a symbol that indicates a **mathematical operation** is to be performed, e.g. 'result=a+b'. Make sure that the space bar is used, NOT the TAB key as this w the code is executed. The use of whitespace by leaving a line space betwee also improve the readability of the code, as will leaving two lines space betw longer sections of code.

Example:



```
# return values and re-use functions

def get_student_name():
    """ Requests name string and returns"""
    student_name = input("Please enter your name: ")
    return student_name

def get_teacher_name():
    """ Requests teacher name string and returns"""
    teacher_name = input("Please enter your Computer Science teacher's name: ")
    return teacher_name

def get_marks():
    """asks for 4 values and returns average"""
    m1 = int(input("Please enter the mark for the first HW task:  "))
    m2 = int(input("Please enter the mark for the second HW task:  "))
    m3 = int(input("Please enter the mark for the third HW task:  "))
    m4 = int(input("Please enter the mark for the final HW task:  "))
    avg = (m1+m2+m3+m4)/4
    return avg

def teacher_comment(a,t,s):
    """Prints comment based on variable value."""

    if a >= 8:
        print(f"Well done, {s}. {t} is very pleased with your effort.")
    elif a>=6 and a <8:
        print(f"A good effort, {s} ,{t} thinks you should check your work carefully.")
    else:
        print(f"{s} this is not an acceptable level of effort.{t} has asked you to at

def main():
    """ runs all functions"""
    student_name = get_student_name()
    teacher_name = get_teacher_name()
    avg = get_marks()
    teacher_comment(avg,teacher_name,student_name)

main()
```

Following PEP 8 guidelines:



```
def get_student_details():
    """ Requests name string and returns"""
    student_name = input("Please enter your name: ")
    return student_name


def get_teacher_details():
    """ Requests teacher name string and returns"""
    teacher_name = input("Please enter your Computer Science teacher's name: ")
    return teacher_name


def get_marks():
    """asks for 4 values and returns average"""
    hw1_mark = int(input("Please enter the mark for the first HW task:  "))
    hw2_mark = int(input("Please enter the mark for the second HW task:  "))
    hw3_mark = int(input("Please enter the mark for the third HW task:  "))
    hw4_mark = int(input("Please enter the mark for the final HW task:  "))
    avg = (hw1_mark+hw2_mark+hw3_mark+hw4_mark)/4
    return avg


def teacher_comment(a [illegible]
    """ Prints co[illegible] variable value."""

    [illegible] 8
        [illegible]t(f"Well done, {s}. {t} is very pleased with your effort.")
    [illegible]6 and a <8:
        print(f"A good effort, {s} ,{t} thinks you should check your work carefully
    else:
        print(f"{s} this is not an acceptable level of effort.{t} has asked you to


def main():
    """ runs all functions"""
    student_name = get_student_details()
    teacher_name = get_teacher_details()
    avg = get_marks()
    teacher_comment(avg,teacher_name,student_name)

main()
```

The function names used in the first example were also too close to the va these were amended as well. Looking at the two improved versions, it is o version is clearer and easier to read.

Using these guidelines will help you and your teacher understand what y you return to it each lesson.

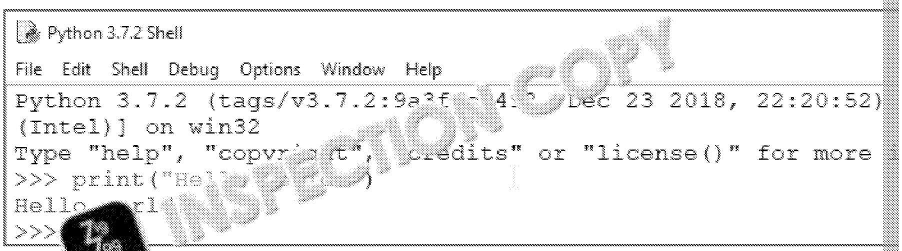
1. THE BASICS

Your first program: “Hello world!”

We are going to use the print() function to write our first program. Python enter in the brackets and print out a text **string** OR the result of a mathem

Open IDLE and type this into the interpreter or ‘shell’ and press ENTER:

You should get this response from the ‘shell’:



```
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52)
(Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more i
>>> print("Hello world!")
Hello world!
>>>
```

All the interpreter is doing is printing to the screen the string of characters that you entered inside the brackets. The print() function is a ready-made to print to the screen.

There are many more **built-in functions** in Python that you will learn about

USING PYTHON AS A CALCULATOR

When we use numbers in Python, normal mathematical operators apply. B commonly used operators.

Command	Name	Example
+	Addition	4+5
-	Subtraction	8-5
*	Multiplication	4*5
/	Division	19/3
%	Remainder	19%3
**	Exponent	2**4
//	Whole Number Division (Floored Quotient)	7//2 (rounded dow

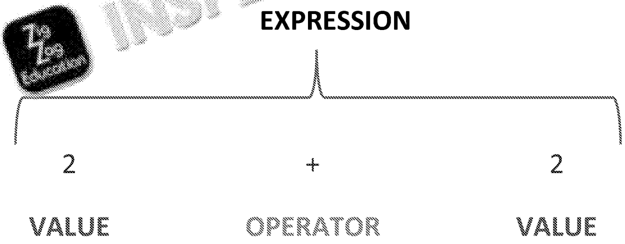
PSEUDO NUMBERS – WHAT ARE THEY?

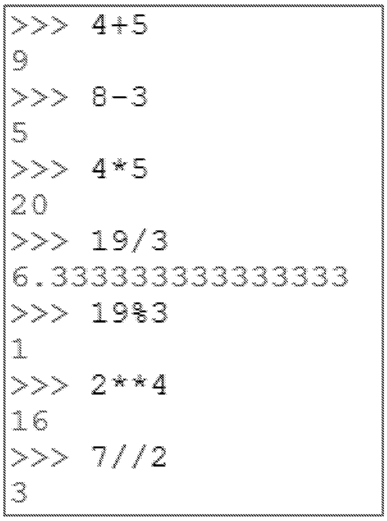
These are numbers that are not meant for calculation, like a telephone or n security number, so we must use a STRING data type to record them, rathe data type.

You will learn more about what a data type is in the next few pages.

EXPRESSIONS

In mathematics, an expression means the numbers, symbols and operators together to show the value of something. In Python, if we type an expressio window, the 'shell', Python will interpret the answer and display it:





```
>>> 4+5
9
>>> 8-3
5
>>> 4*5
20
>>> 19/3
6.333333333333333
>>> 19%3
1
>>> 2**4
16
>>> 7//2
3
```

1.1 ORDER OF OPERATION

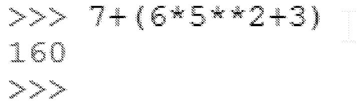
Python automatically understands the order of operation of any mathema
remember this as BIDMAS or BODMAS from your Mathematics lessons:

Brackets	**B**rackets
Indices	**O**rder
Division	**D**ivision
Multiplication	**M**ultiplication
Addition	**A**ddition
Subtraction	**S**ubtraction

Example:

7 + (6 × 5^2 + 3)

1. Evaluate the part of the calculation inside the brackets
   a. Inside the brackets the first part to calculate is 5^2 = 25
   b. Next, multiply 6 × 25 = 150
   c. Add 3 = 153
2. Now evaluate the part of the calculation outside the brackets
   a. 7 + 153 = 160



```
>>> 7+(6*5**2+3)
160
>>>
```

EXERCISE 01: CALCULATE USING PYTHON

Use the Python interpreter 'shell' and the print() function to complete the

1. 50-5*6
2. (50-5*6)/4
3. 8/5
4. 4*3
5. 12/3
6. 21%6
7. 17//5
8. 5*3+2
9. 5**2
10. 1//2

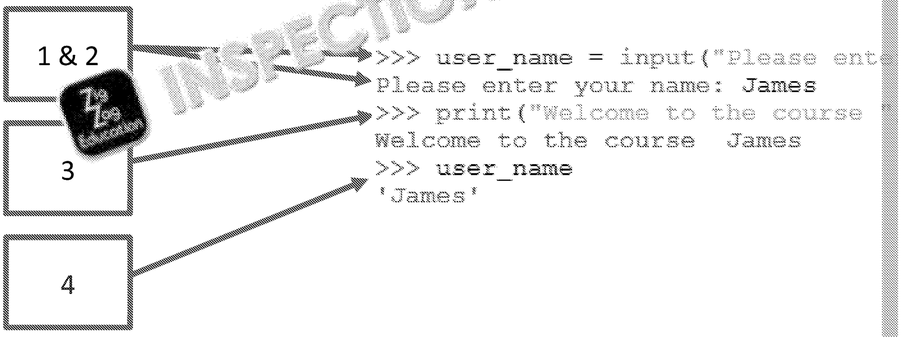
2. VARIABLES AND ASSIGNMENT S

In computer programming, a LITERAL is any type of data that represents a v text. Variables are reserved memory locations to store values. **This means t variable you reserve some space in memory.**

The data used in a program **must be stored** in main memory while the prog names for these areas in memory so we can refer to them in our programs.

- When we create a variable, the name we use refers to the location of t
- Each variable will have a data type (see Data Types). The data type det with the data and how much space it needs in memory.

We can create a variable, store the data entered and use that data again. Lo



```
>>> user_name = input("Please ente
Please enter your name: James
>>> print("Welcome to the course "
Welcome to the course  James
>>> user_name
'James'
```

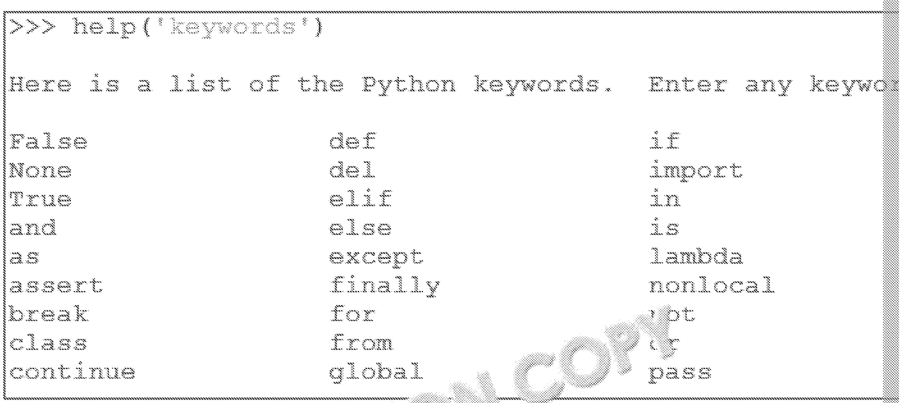
1. A variable 'user_name' is created and the value entered from the inpu to the variable.
2. The value of 'user_name' is now James.
3. The print() statement uses the variable to output a message to screen.
4. The value of the variable can be checked by typing it and pressing the

2.1 VARIABLE NAMES

When we create variables and choose names for them there are some basi

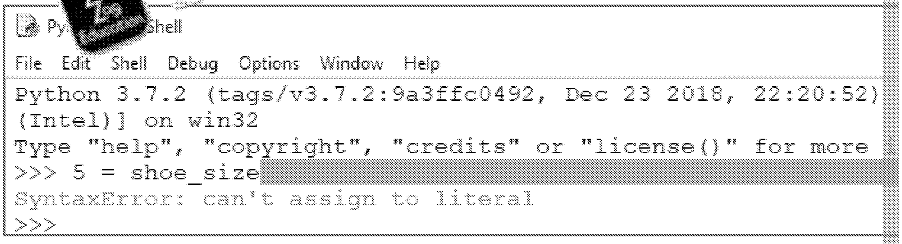
1. Variable names must NOT have spaces
   e.g. my name ☒ myName ☑ my_name ☑ my_Name ☑
2. Variable names must not use any reserved keywords (see below)
3. Choose a consistent naming style (and stick to it!)
4. Must begin with a letter (a–z, A–Z) or underscore ('_)
5. Other characters can be letters, numbers o
6. Variable names are case-sensitive (this means you must use the same wherever you use the variable name)
7. Use a name that makes sense (in the context of the program)
   e.g. studentName **NOT** sn

The reserved keywords have a special meaning and purpose in Python. You
by typing help ('keywords') in the Python interpreter window.



```
>>> help('keywords')

Here is a list of the Python keywords.  Enter any keywo

False               def                 if
None                del                 import
True                elif                in
and                 else                is
as                  except              lambda
assert              finally             nonlocal
break               for                 not
class               from                or
continue            global              pass
```

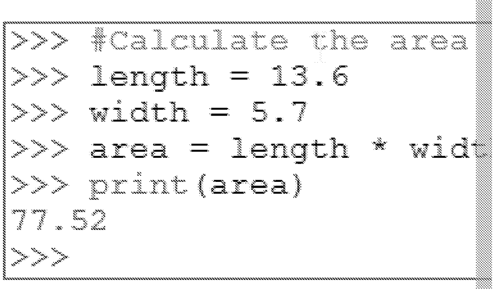
When we ASSIGN a value to a variable it is important that the variable nam
value is assigned.



```
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52)
(Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more i
>>> 5 = shoe_size
SyntaxError: can't assign to literal
>>>
```

Python returns a syntax error as the rules about variable names have not b
encounter this error message regularly as you develop your skills and know

When we assign values to variables, we can use the variable names in calculations instead of the values. As you develop more programming skills you should try to follow this method in your work.

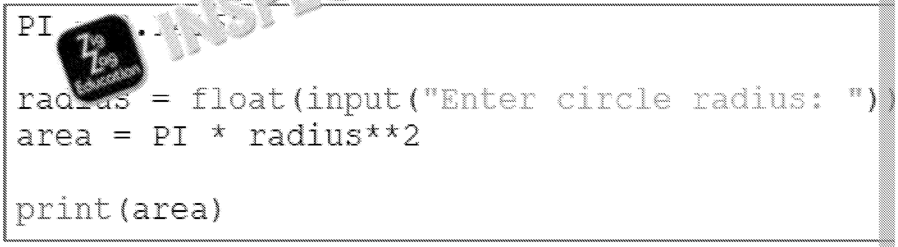


```
>>> #Calculate the area
>>> length = 13.6
>>> width = 5.7
>>> area = length * widt
>>> print(area)
77.52
>>>
```

2.2 CONSTANTS

A constant is a value that once declared in the code remains the same and
Unlike many other programming languages like C++ or Java, there is no eas
create a constant. At GCSE level we can use the PEP 8 guidelines and write
be used as a constant in all capitals.

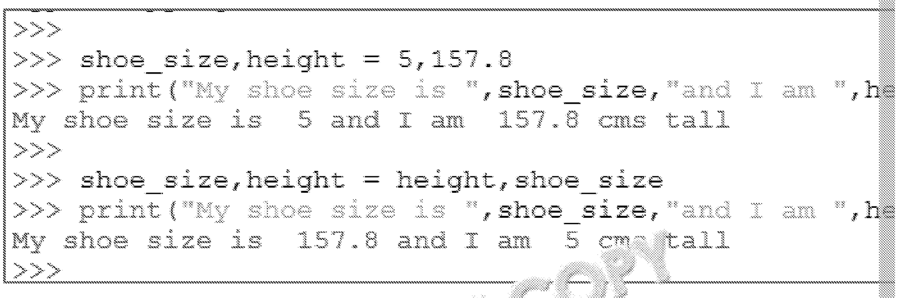
For example:



```
PI = 3.14159

radius = float(input("Enter circle radius: "))
area = PI * radius**2

print(area)
```

ASSIGNING MULTIPLE VALUES

Python also allows us to make our code more **efficient** by allowing multiple
to variables:



```
>>>
>>> shoe_size,height = 5,157.8
>>> print("My shoe size is ",shoe_size,"and I am ",he
My shoe size is  5 and I am  157.8 cms tall
>>>
>>> shoe_size,height = height,shoe_size
>>> print("My shoe size is ",shoe_size,"and I am ",he
My shoe size is  157.8 and I am  5 cms tall
>>>
```

The variable names are separated by commas; as you can see they print in
second **assignment statement** these have been swapped around.

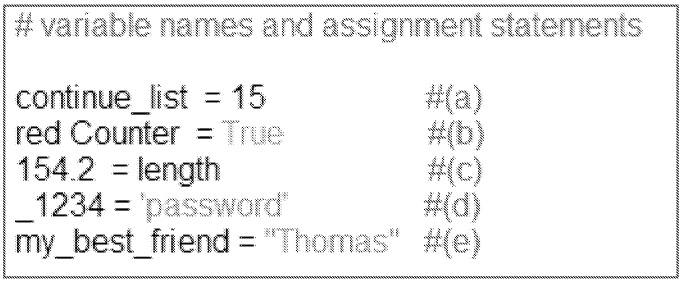
There are three different DATA TYPES shown in the example above:

1. The variable **shoe_size** is an INTEGER data type. This is a whole numbe
2. The variable **height** is a REAL or FLOAT data type. This is a number wit
3. The text surrounded by speech marks inside the brackets of the print func

Look at *Chapter 3: Data Types* for more information.

EXERCISE 02: VARIABLES AND ASSIGNMENT

1. Which of these variable names and assignment statements will NOT c



```
# variable names and assignment statements

continue_list  = 15             #(a)
red Counter  = True             #(b)
154.2  = length                 #(c)
_1234 = 'password'              #(d)
my_best_friend = "Thomas"   #(e)
```

2. Write a program to calculate the area of a circle with a radius of 6.7 cn
   the calculation.
   a. Set the value of Π = 3.142
   b. The area of a circle is Π × radius squared

3. Write a program to store the following variables
   a. Forename
   b. Surname
   c. Age
   d. Height

   Use the print function to display the information in a sentence.

3. DATA TYPES

The data types you will be using are predefined in the Python programmin
correct data type to use is important because it determines what actions w
data, e.g. we cannot divide a text string by an integer. Every value you stor
following data types:

STRING

- A string is a combination of characters that can include letters, numbe be within single or double quotes.
- We can add them together, e.g. 'birth' + 'day' would give you 'birthda
- Used to represent pseudo-numbers, e.g. a telephone number.

INTEGER

- An Integer is a whole number, it can be positive or negative.
- We can use a range of mathematical operators (+ - * / etc.) on intege

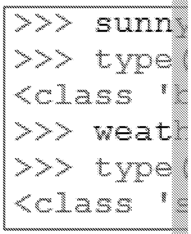
REAL

- This enables us to store a number with a fractional part.
- A real number is sometimes called a float or single or double.

BOOLEAN

- A Boolean value can be True or False.
- Many questions have the answer 'Yes' or 'No'.
- We want to find out whether things are true or not in our programs.

I can check what data type a variable is by using: *type (variable_name).*

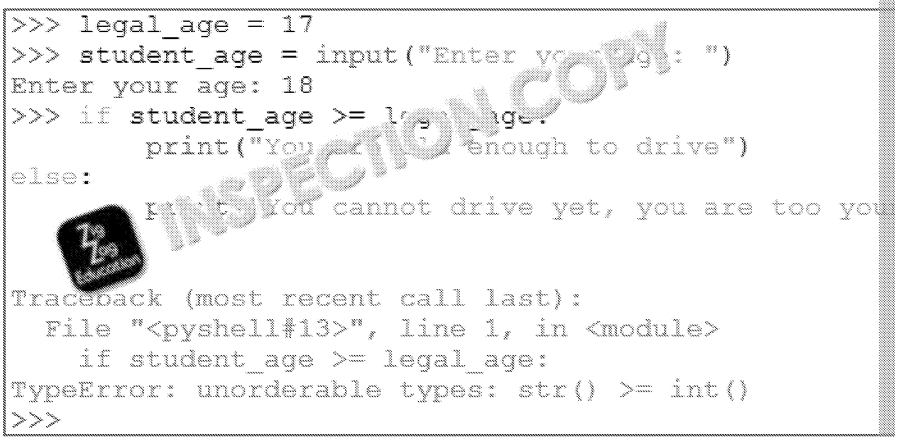


```
>>> sunny
>>> type(
<class 'b
>>> weat
>>> type(
<class 's
```

Why is this important?
You might use this to ensure that the result of a calculation is shown as a decimal as this may affect the rest of your program and give a false result.

You may want to compare values which are not the same data type, e.g. yo
a string and then needs to be converted to an integer to check whether th
less than a numeric value.

Look at this example, which causes a Type Error when the code is execute



```
>>> legal_age = 17
>>> student_age = input("Enter your age: ")
Enter your age: 18
>>> if student_age >= legal_age:
        print("You are old enough to drive")
else:
        print("You cannot drive yet, you are too you


Traceback (most recent call last):
  File "<pyshell#13>", line 1, in <module>
    if student_age >= legal_age:
TypeError: unorderable types: str() >= int()
>>>
```

3.1 CASTING DATA TYPES

Python assumes that all data entered using the **input()** built-function is a ST type is CAST into a different data type. This means telling Python to treat th INTEGER or a FLOAT.

In the improved version, the data input is CAST to an INTEGER when the in could also be done like this:

```
>>> student_age = input("Enter your age: ")
Enter your age: 18
>>> student_age = int(student_age)
>>> type(student_age)
<class 'int'>
```

Used to sho changed.

The example below is more **efficient** as only one line of code has been use data rather than two lines of code as shown above.

Efficient code:

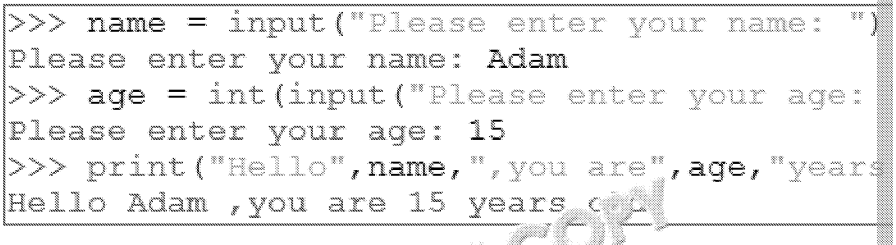
```
>>> legal_age = 17
>>> student_age = int(input("Enter your age
Enter your age: 18
>>> if student_age >= legal_age:
        print("You are old enough to drive")
else:
        print("You cannot drive yet, you ar


You are old enough to drive
```

More examples of casting data types:

```
>>> student_age = '16'
>>> type(student_age)
<class 'str'>
>>> type(int(student_age))
<class 'int'>
>>> type(float(student_age))
<class 'float'>
>>> type(str(student_age))
<class 'str'>
```

3.2 GETTING DATA INPUT

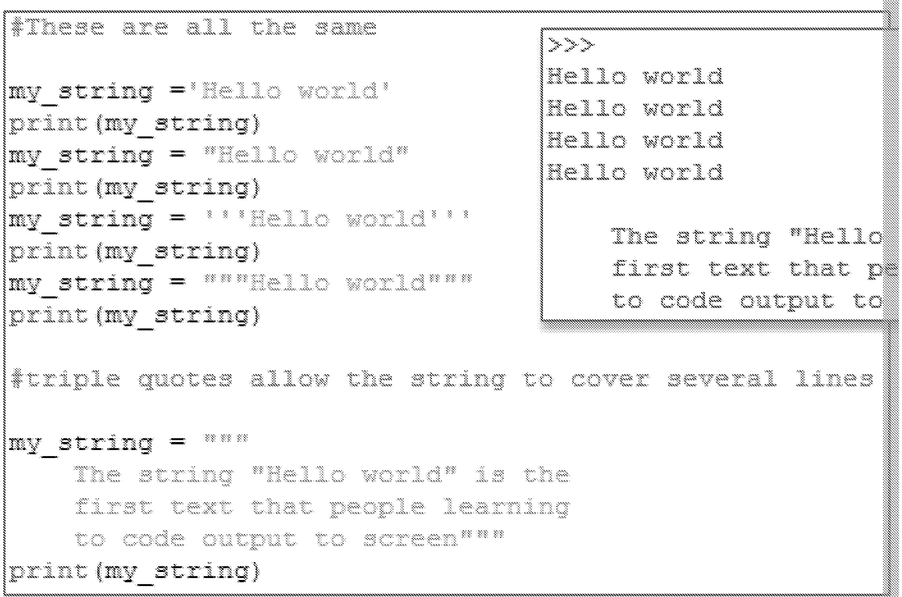
We have already seen examples of the built-in function input() being used t
The efficient code example above shows how the input from the user can b
data type by putting the input() code inside a data type.



```
>>> name = input("Please enter your name: ")
Please enter your name: Adam
>>> age = int(input("Please enter your age: 
Please enter your age: 15
>>> print("Hello",name,",you are",age,"years
Hello Adam ,you are 15 years old
```

3.3 STRINGS

As already mentioned in Data Types, strings are sequences of characters e
They can be enclosed by single quotes, double quotes or three quotes of e
the output from each version is the same.



```
#These are all the same

my_string ='Hello world'
print(my_string)
my_string = "Hello world"
print(my_string)
my_string = '''Hello world'''
print(my_string)
my_string = """Hello world"""
print(my_string)

#triple quotes allow the string to cover several lines

my_string = """
    The string "Hello world" is the
    first text that people learning
    to code output to screen"""
print(my_string)
```

```
>>>
Hello world
Hello world
Hello world
Hello world

    The string "Hello
    first text that pe
    to code output to
```

Which one should I use?

- Using single quotes means that you need to use escape characters in t
  to use characters like a backslash, an apostrophe or double quotes.
- Using double quotes means that you do not need to use escape chara

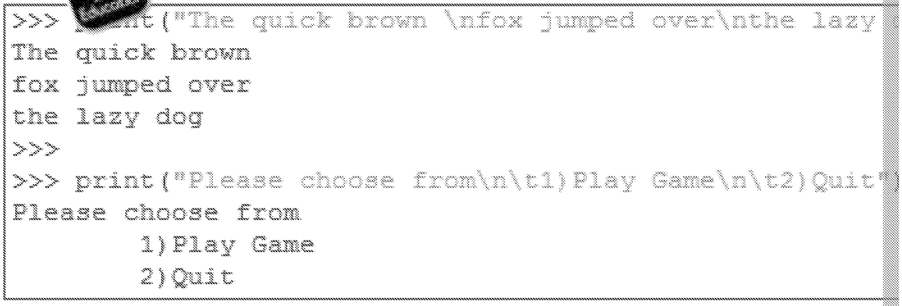
Using triple quotes means that text can span several lines, although these a
docstrings in functions; see Docstring and Comments for more informatio

Escape characters

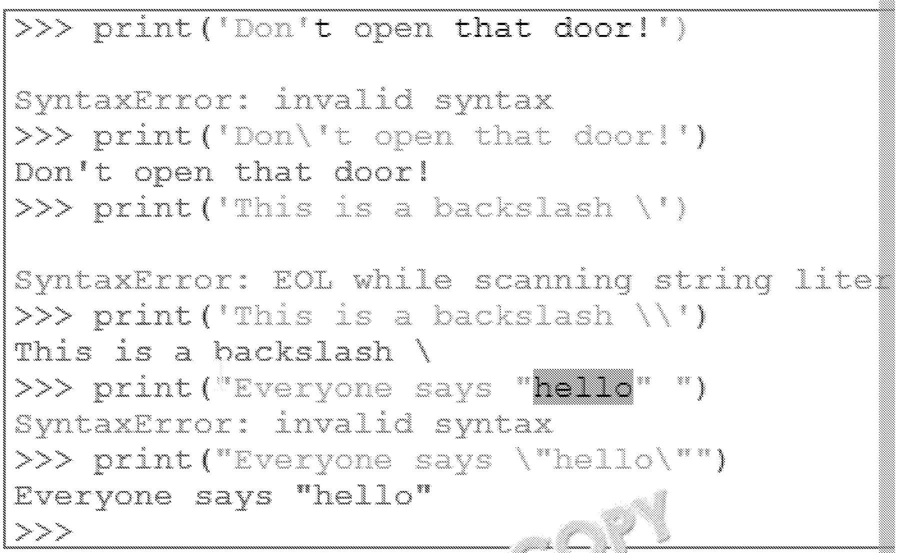
Some useful escape characters are shown below.

Escape character	What it does
\\	Allows the use of a backslash inside a single quote
\’	Allows the use of an apostrophe inside a single qu
\”	Allows the use of double quotes inside a single qu
\n	ASCII linefeed-newlin
\t	Horizontal tab (indents your text string)

Examples



```
>>> print("The quick brown \nfox jumped over\nthe lazy 
The quick brown
fox jumped over
the lazy dog
>>>
>>> print("Please choose from\n\t1)Play Game\n\t2)Quit")
Please choose from
	1)Play Game
	2)Quit
```



```
>>> print('Don't open that door!')

SyntaxError: invalid syntax
>>> print('Don\'t open that door!')
Don't open that door!
>>> print('This is a backslash \')

SyntaxError: EOL while scanning string liter
>>> print('This is a backslash \\')
This is a backslash \
>>> print("Everyone says "hello" ")
SyntaxError: invalid syntax
>>> print("Everyone says \"hello\"")
Everyone says "hello"
>>>
```

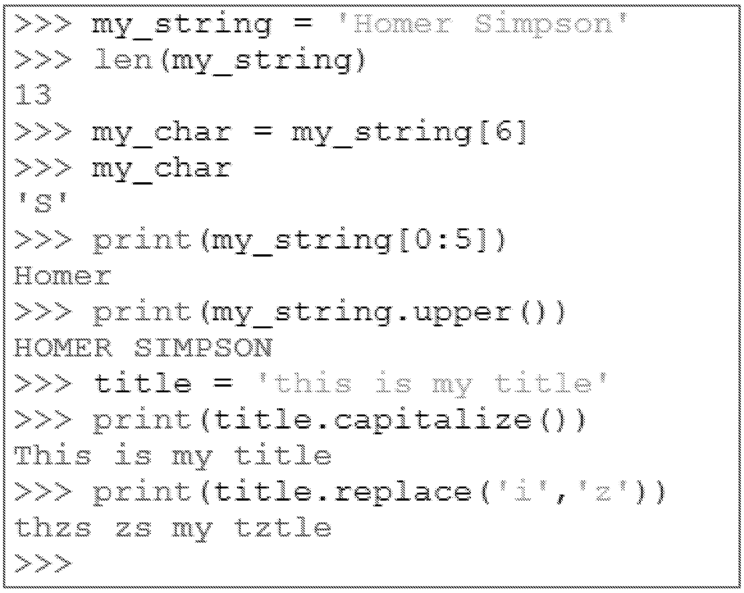
STRING OPERATIONS

Strings are immutable; this means that once we have created a string variable

There are a number of string operations we can perform on a string variabl writing your code. Here are some common examples:

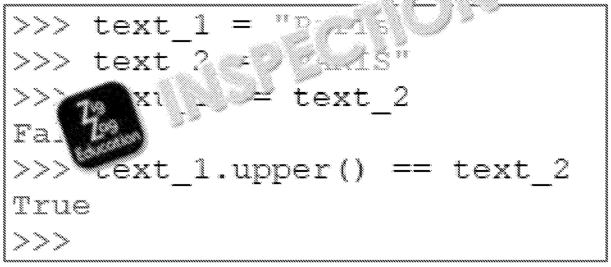
- *len (myString)* – returns the number of characters in the string, includi
- *myString.upper()* – returns the string in upper case.
- *myString.lower()* – returns the string in lower case.
- *myString.capitalize()* – returns the string with the first letter of the stri
- *myString.title()* – returns the string with the first letter of each word ca
- *myString.replace(x, y)* – returns the string with the characters represe characters represented by y.
- *myString[x:y]* – returns the a substring of the original string starting a character y.

Examples of use:



```
>>> my_string = 'Homer Simpson'
>>> len(my_string)
13
>>> my_char = my_string[6]
>>> my_char
'S'
>>> print(my_string[0:5])
Homer
>>> print(my_string.upper())
HOMER SIMPSON
>>> title = 'this is my title'
>>> print(title.capitalize())
This is my title
>>> print(title.replace('i','z'))
thzs zs my tztle
>>>
```

Why is this important?
Look at this example; I am trying to test whether two strings are the sa equality operator.

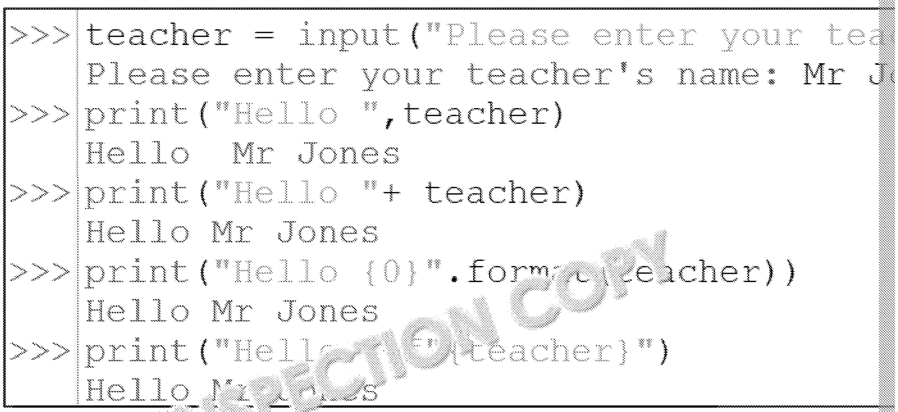


```
>>> text_1 = "Paris"
>>> text_2 = "PARIS"
>>> text_1 == text_2
False
>>> text_1.upper() == text_2
True
>>>
```

If I test equality WITHOUT converting the first string to upper case, the res false; Python does not recognise that the strings are the same.

FORMATTING STRINGS

When we need to use variables inside a print statement there are a numbe
do this:



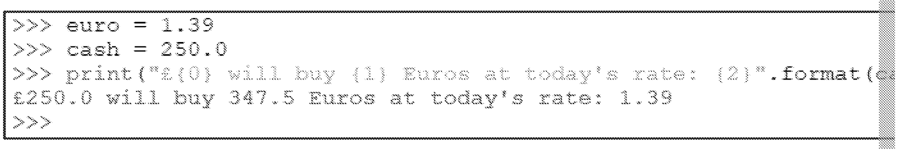
```
>>> teacher = input("Please enter your tea
    Please enter your teacher's name: Mr J
>>> print("Hello ",teacher)
    Hello  Mr Jones
>>> print("Hello "+ teacher)
    Hello Mr Jones
>>> print("Hello {0}".format(teacher))
    Hello Mr Jones
>>> print("Hello {teacher}")
    Hello Mr Jones
```

1. Use a comma to add the variable into the print statement ☒ Avoid this me
2. Concatenate the string with the variable inside the print statement ☑
3. Use the string method .format() ☑ Depending o
   ** This is outdated from Python 3.6*
4. The f-string method ☑ Depending o

Note: If you are using a version of Python before 3.6, then you would use the .form string method used from Python 3.6 is f-strings.

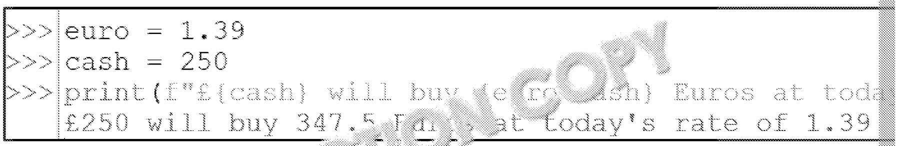
The .format() method allows the flexible use of combining strings with a va needing concatenation.

Example:



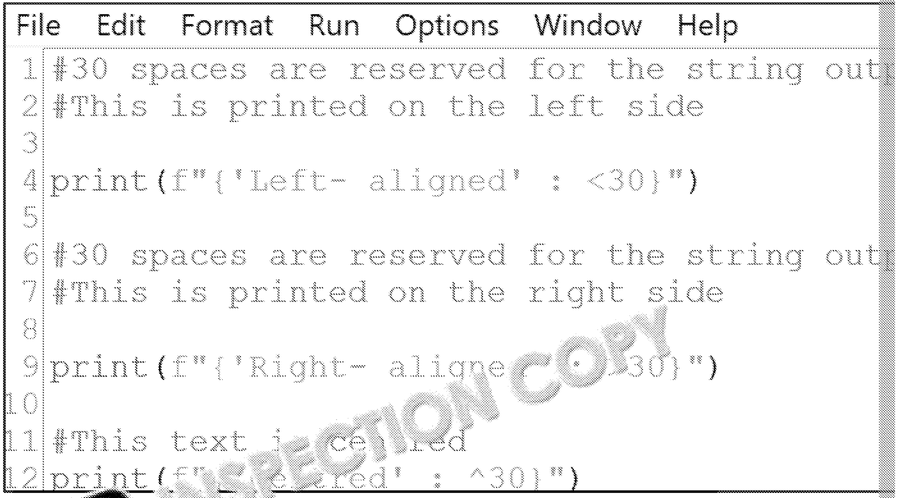
```
>>> euro = 1.39
>>> cash = 250.0
>>> print("£{0} will buy {1} Euros at today's rate: {2}".format(ca
£250.0 will buy 347.5 Euros at today's rate: 1.39
>>>
```

The f-string method allows the same level of flexibility but is more efficien script above with this version:

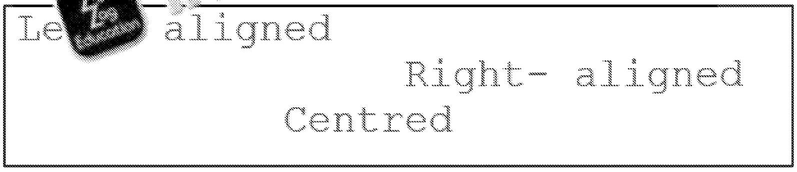


```
>>> euro = 1.39
>>> cash = 250
>>> print(f"£{cash} will buy {euro*cash} Euros at toda
    £250 will buy 347.5 Euros at today's rate of 1.39
```

String alignment tricks:



```
#30 spaces are reserved for the string out
#This is printed on the left side

print(f"{'Left- aligned' : <30}")

#30 spaces are reserved for the string out
#This is printed on the right side

print(f"{'Right- aligned' : >30}")

#This text is centred
print(f"{'Centred' : ^30}")
```

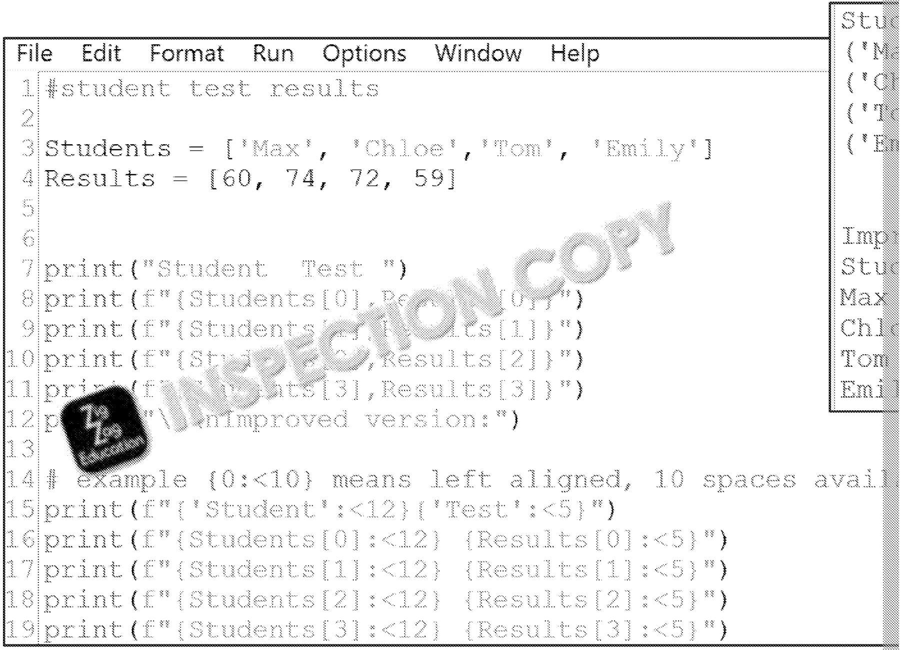


```
Left- aligned
                  Right- aligned
           Centred
```

Character	Alignment
<	left alignment
^	centre alignment
>	right alignment

We can also use additional 'control characters' to improve the way any var

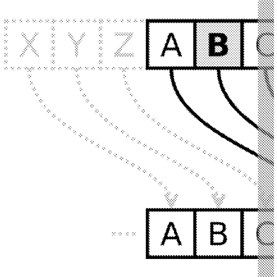
The code in this screenshot has been created in script mode. This means it pressing F5; the file name must have the file extension .py



```
#student test results

Students = ['Max', 'Chloe','Tom', 'Emily']
Results = [60, 74, 72, 59]


print("Student  Test ")
print(f"{Students[0],Results[0]}")
print(f"{Students[1],Results[1]}")
print(f"{Students[2],Results[2]}")
print(f"{Students[3],Results[3]}")
print("\nImproved version:")

# example {0:<10} means left aligned, 10 spaces avail
print(f"{'Student':<12}{'Test':<5}")
print(f"{Students[0]:<12} {Results[0]:<5}")
print(f"{Students[1]:<12} {Results[1]:<5}")
print(f"{Students[2]:<12} {Results[2]:<5}")
print(f"{Students[3]:<12} {Results[3]:<5}")
```

```
Stu
('Ma
('Ch
('To
('Em


Imp
Stu
Max
Chl
Tom
Emi
```

USING ORD() AND CHR()



These two built-in functions are commonly used when creating a simple Caesar Cipher program to encrypt a text string.

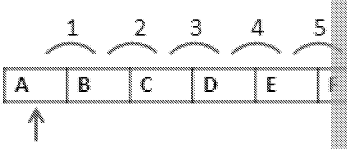
A Caesar Cipher simply substitutes the actual character in the string with another letter a certain number of spaces further on in the alphabet.

The ord() function allows us to represent each letter as an ordinal number. We can then add the required number of letters to shift by, which must be between 1 and 26, to find the value of the new number. The new number can be changed back to a letter by using the chr() function.

The numbers used to represent each upper or lower case letter in the alphabet are based on the ASCII character set. Computers only understand binary, so the ASCII codes represent text and other punctuation symbols. The diagram shows some of the letters and their numerical representations in binary, octal, decimal and hexadecimal.[1]

Example:

```
>>> ord('A')
65
>>> ord('a')
97
>>> letter = 'A'
>>> new_letter = ord(letter) + 10
>>> chr(new_letter)
'K'
>>>
```



This simple example shows how we can use these two built-in functions to
Cipher. The key is 7 and **z** is 7 letters away from **s** in the alphabet.

```
password = "secret"
encrypted = ''    # empty string for the encrypted password

# encrypt the password
key = 7               # the number of letters to the right
for each in password:
    new_letter = ord(each)+key
    encrypted += chr(new_letter) # each new letter is added

print(encrypted)
```

EXERCISE 03: STRINGS

Complete these exercises in script mode in IDLE.

1. Prompt the user to enter their name. Output a message: 'Hello (name) message should be output on two lines using one print statement.

2. Prompt the user to enter two whole numbers. Output a message 'The numbers, (number 1) and (number 2) is (average)'.

3. Prompt the user to enter a string for encryption and a key value betw original string and the encrypted string with a suitable message.

[1] Binary is Base 2; Octal is Base 8; Decimal is Base 10; Hexadecimal is Base 16

4. PROCEDURES AND FUNCT

Up until now, all the work we have been doing with Python has been completed in the interactive mode, where the result of our program code was immediately displayed but we have not saved anything.

We will now use the text editor or script mode to create and save all our programs:

Step 1: File >> New File/Window (depending on your version of IDLE)

Step 2: When the new window opens, choose File >> Save As and save the file as 'print_a_message.py' in a suitable folder on your drive.

We are now going to start writing our own **functions**.

4.1 WHAT ARE FUNCTIONS?

Functions are blocks of organised code that perform specific tasks; we can code more EFFICIENT. Python has many ready-made functions that we can built-in functions, and you will already have used the print() and input() fu

Code Efficiency

What does this mean?

At GCSE level it means:

- using the most suitable variable names and data types
- reusing code wherever possible
  - NOT copying and pasting but using a function seve complete a task
- avoiding unnecessary code
  - by using loops to perform tasks
  - by using IF/ELIF statements to test conditions

Why should we use functions?

- They make your code easier to read as you can see what tasks each bl (providing you use a suitable name).

Functions can be reused instead of repeating code

- If changes are needed, they can be made to just one block of code.
- It is easier to organise the order in which the program runs.

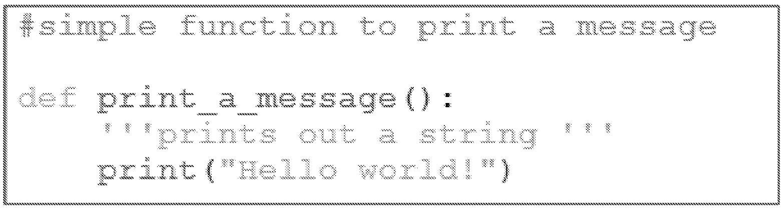
PROCEDURES VS FUNCTIONS

In Python the blocks of code we write are ALL called functions. In program to know the difference between a function and a procedure.

- A procedure is a self-contained block of code that performs a task, e.g. of two numbers, printing out a list of numbers. The procedure MAY re not have to.
- A function is also a self-contained block of code that performs a task, e from a user and checking that the input is valid. A function will ALWAY Returning a value)

4.2 WRITING A FUNCTION

Here is an example of a very simple function:



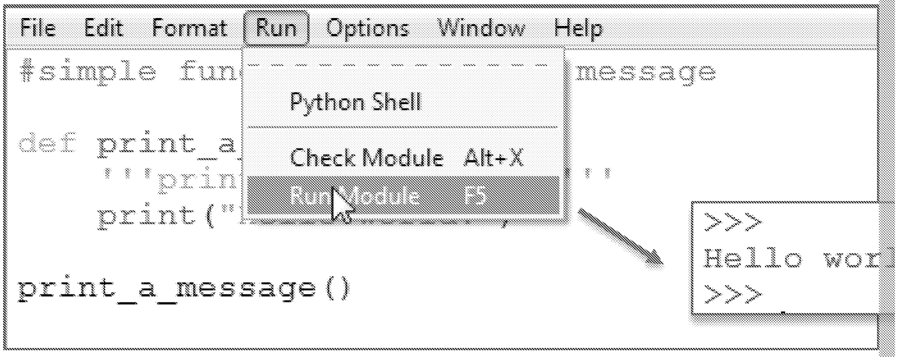
```
#simple function to print a message

def print_a_message():
    '''prints out a string '''
    print("Hello world!")
```

- We use the 'def' statement to tell Python we are defining a function.
- The name should be what the function *does* – keep it simple and use u
  words.
- The name is followed by two brackets (parentheses); this will hold any
  [illegible]on will use. These are called the function parameters.
- After the brackets we add a colon.
- The body of the function, the code statements we want to execute, a
  usually by four spaces. Simply press enter after the colon and IDLE wi

In order to execute or 'run' the function and see the result of the code in t
must 'call' the function.

4.3 CALLING THE FUNCTION

The function will not run until we instruct Python to execute the code by '
EITHER pressing the F5 key OR choosing Run >> Run Module.



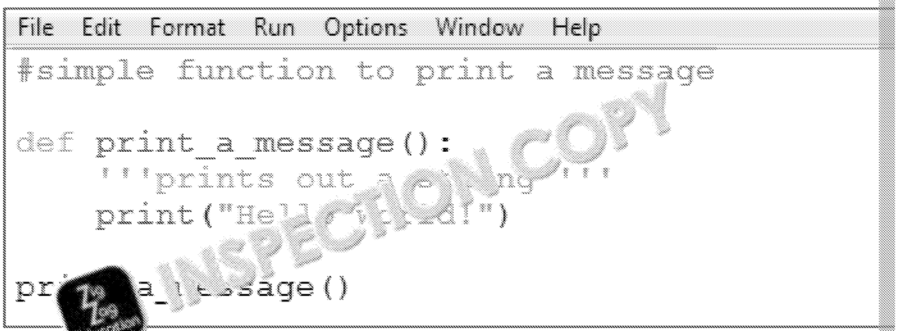
```
#simple fun              message

def print_a
    '''prin              '''
    print("

print_a_message()
```

```
>>>
Hello wor
>>>
```

4.4 DOCSTRING AND COMMENTS

One of the most important parts of programming is making your code eas understand at a later date. Using one-line comments and docstrings in yo helps with this.

Docstrings are used to explain what the function does and one-line comm additional explanation to your code.



```
#simple function to print a message

def print_a_message():
    '''prints out a string'''
    print("Hello World!")

print_a_message()
```

Comments are shown by using a hash symbol at the start; these can be ad code to explain what it does. The docstrings should be added on the first l enclosed by three single OR three double quotation marks.

More guidance on how to maintain your code can be found in ***Chapter 12***

EXERCISE 04: BASIC FUNCTION

We are now saving each exercise.

1. Open IDLE and choose New File/Window and save as BasicFunction1.
2. Write separate functions to do the following:

Function 1	Write a function to add two numbers together and pr
	Use the following variable names and values:
	x = 17
	y = 22

Function 2	Write a function to multiply x and y together and divid
	Use the following variables in your function:
	x = 6
	y = 4
	z = 8

For each function:

a. Ensure you have a docstring

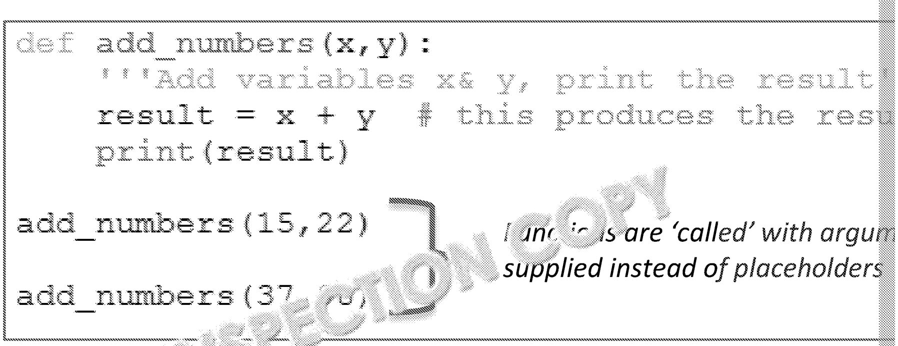
b. Use at least one one-line comment

3. Call each function and ensure that it works

4.5 EXTENDING THE BASIC FUNCTION

So far the functions we have written are very basic; we will now look at ext use parameters or arguments.

Look at this example:



```
def add_numbers(x,y):
    '''Add variables x& y, print the result'
    result = x + y  # this produces the resu
    print(result)

add_numbers(15,22)

add_numbers(37,[illegible])
```

Functions are 'called' with argum supplied instead of placeholders

Parameters are the 'placeholders' used when the function is defined. In this

Arguments are the actual values we use when we 'call' the function.

EXERCISE 05: EXTEND BASIC FUNCTION

1. Develop Function 2 from the previous exercise to use parameters and arguments:

   x = 15
   y = 13
   z = 5

2. Write a function that will take in any string and print it out followed by Supply the following argument to the function:

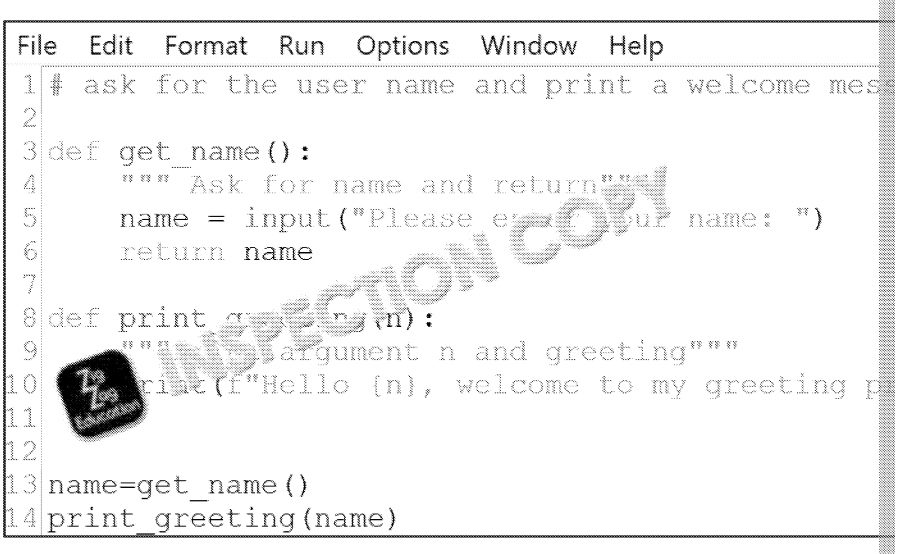
   'There are only 10 types of people in the world.'

The next step is to develop our function to RETURN values. A function parameters or not, it can also return a value or not.

4.6 RETURNING A VALUE

When we return a value from a function we do this so we can use that valu another function or part of the program code.

Look at this example using two more built-in functions:

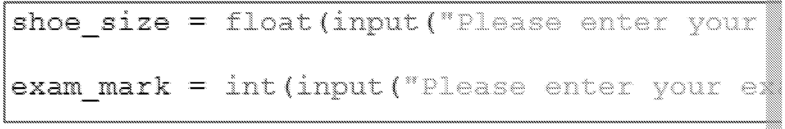


```
# ask for the user name and print a welcome mes

def get_name():
    """ Ask for name and return"""
    name = input("Please enter your name: ")
    return name

def print_greeting(n):
    """[illegible] argument n and greeting"""
    print(f"Hello {n}, welcome to my greeting p



name=get_name()
print_greeting(name)
```

This time when we call the get_name() function, we must tell Python wher created *before* we use it as an argument in the print_greeting() function.

This example includes two more built-in functions:

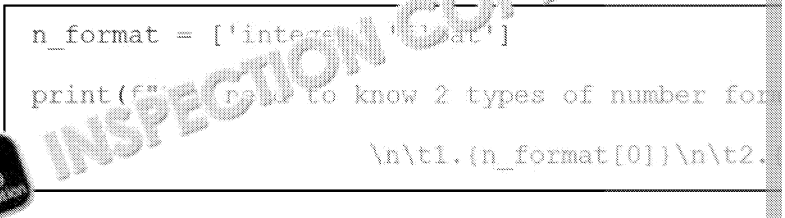
- input() – this gets input from the user into our program.
  - Unless specifically instructed otherwise, Python treats the values en
  - it is important to use the correct data types to avoid errors (see C more on this.
  - This means we must 'cast' the data type to the one we want to us



```
shoe_size = float(input("Please enter your

exam_mark = int(input("Please enter your ex
```

- f-strings (formatted string literals) – this helps with the presentation o
  - We can use this method to include variables in our text string. Lo information.

    Simple example:



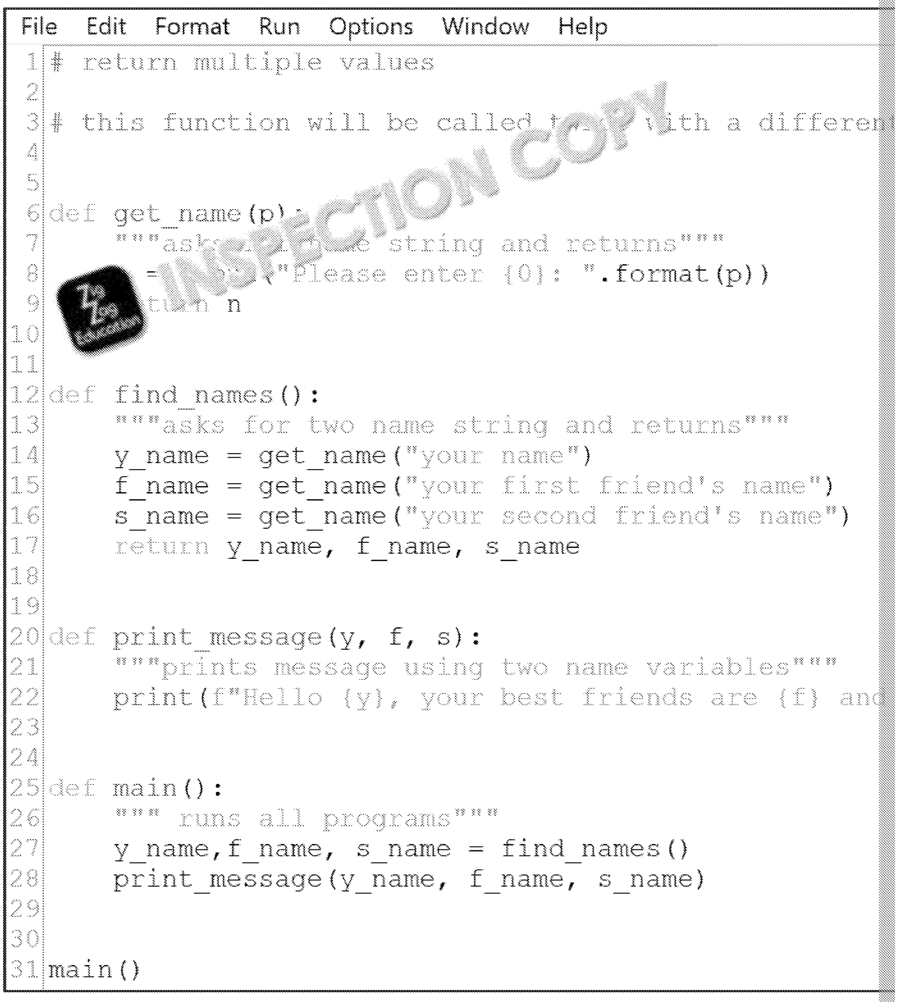
```
n_format = ['integer','float']

print(f"We need to know 2 types of number for

                    \n\t1.{n_format[0]}\n\t2.
```

  - Variables can be placed in the print statement using curly braces. variables in a list, 'n_format'. The code uses square brackets to sh in the variable list.
  - The arguments are placed inside the f-string curly braces.

4.7 RETURNING MULTIPLE VALUES

Sometimes you may write a function where you need to return more than achieved by using a data structure called a tuple (see ***Chapter 9: Tuples***). A values separated by commas. Tuples are sometimes surrounded by parent don't have to be.

Look at this example:



```
# return multiple values

# this function will be called twice with a differen


def get_name(p):
    """asks for name string and returns"""
    n = input("Please enter {0}: ".format(p))
    return n


def find_names():
    """asks for two name string and returns"""
    y_name = get_name("your name")
    f_name = get_name("your first friend's name")
    s_name = get_name("your second friend's name")
    return y_name, f_name, s_name


def print_message(y, f, s):
    """prints message using two name variables"""
    print(f"Hello {y}, your best friends are {f} and


def main():
    """ runs all programs"""
    y_name,f_name, s_name = find_names()
    print_message(y_name, f_name, s_name)


main()
```

4.8 REUSING FUNCTIONS

One of the advantages of creating user-defined functions – writing your ow is that you can design them so that they can be reused.

In the example code on the previous page, there is a 'helper' function on li pass different prompts to the function to ask for different names. Althoug return value 'n', we can assign a different name to this variable each time v lines 14, 15 and 16. This makes the code much more efficient as we are re times, supplying different **arguments** each time we **call** the function.

We are then using a tuple to return the three string variables on line 17, wh function on line 20 so they can be inserted into the printed message.

Finally, it is good practice to use a main function to control the order in whi called in your program. As you can see, the variables which are returned fr be listed in the same order as when they are returned in the design of the f

A common error when passing several variables into a function is that they function in the wrong sequence.

Finally, the main function is called on line 31 so that the functions called in **sequence** specified.

EXERCISE 06: RETURNING VALUES

Write functions that will:

a. Ask for the student name and return it.

b. Ask for the name of their Computer Science teacher and return it.

c. Ask for marks out of 10 for the last four homework tasks and return a *will need to calculate the average inside the function.*

d. Display the following message, depending on the average mark:
   i. average >= 8. 'Well done, X, Y is very pleased with your effort'
   ii. average >= 6 <8. 'A good effort, X .Y thinks you should check your
   iii. average <=5. 'X this is very poor. Y has asked you to try harder'

   Hint: X is the name of the student; Y is the name of the teacher. Remem correctly and declare variables being returned from each function.

5. STRUCTURED PROGRAMM

When you create your own programs, it is important to structure them so t

1. Easy to read
2. Easy to understand
3. Easy to maintain

Programming problems are easier to solve by breaking them down into a se which are easier to understand and solve. The total solution is created whe problems have been solved.

The three constructs that we use in structured programming are:

1. Sequence
2. Selection
3. Iteration

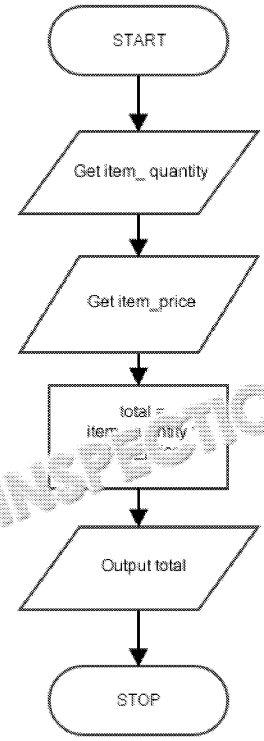
5.1 SEQUENCE

In structured programming, the SEQUENCE in which instructions are execut they appear in the code:

```
# Sequence Example

item_quantity = int(input("Please enter the quant
item_price = float(input("Please enter the item p
total = item_quantity * item_price
print("The total cost is £{0}.".format(total))
```

When trying to solve problems we can also represent them using either pse

Flow chart:



Your programs use **sequencing** whe broken down into a series of steps t after another.

Pseudocode:

```
item_quantity ← INP
item_price ← INPUT
total ← item_quanti
OUTPUT 'The total c
```

RELATIONAL OPERATORS

Operator	What it means
==	Equality operator; checks whether both values are the same
!=	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

LOGICAL OPERATORS

Operator	What it means	Example

AND	Logical AND checks whether both conditions are true or false	>>> x = 6
		>>> x > 0
		True

OR	Logical OR checks whether EITHER of the conditions is true	>>> x = 5
		>>> y = 8
		>>> x/2 =
		True

NOT	Logical NOT reverses a Boolean value. In the example, x > y evaluates to False, using the logical NOT reverses the evaluation to true.	>>> x = 5
		>>> y = 8
		>>> not(x
		True

EXERCISE 07: RELATIONAL AND LOGICAL OPERATORS

Use print statements to find the answers to the following:

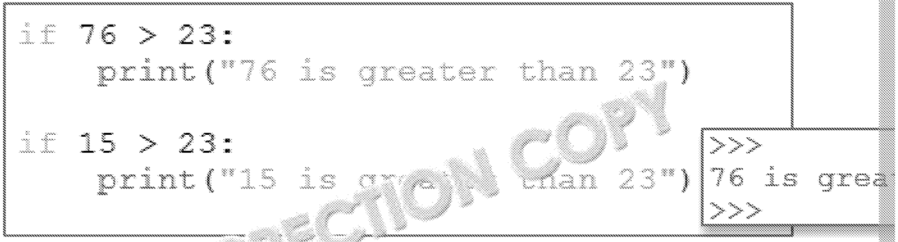
1. Calculate the following (True or False)
   a. 23!=15 b. 5+3<10 c. 6 > 10 == 10< 2
2. [illegible] and b = 8, what are the results of the following statements?
   a. a<b b. 6 >= a c. b > a == False
3. If c = True and d = False, what are the results of the following stateme
   a. c and d b. not d or c c. c == d and True

5.2 SELECTION

IF STATEMENT

Selection or conditional statements execute a block of code based on the r
condition we have set in the code. We are testing to see whether the resul
can set different actions depending on the result of our test.

Example:



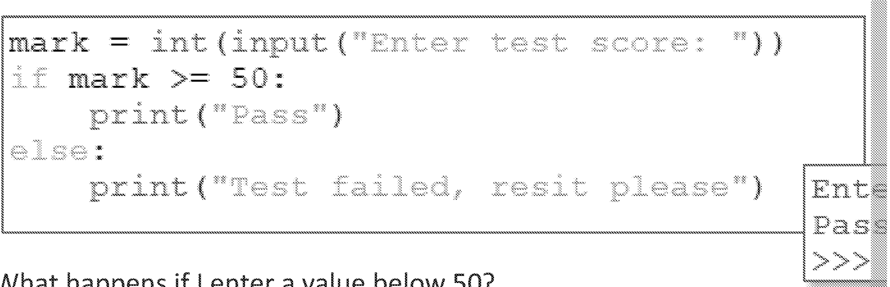
```
if 76 > 23:
    print("76 is greater than 23")

if 15 > 23:
    print("15 is greater than 23")
```

```
>>>
76 is grea
>>>
```

In this example the test in the first block of code evaluates to **true** so the print statement is executed. The test in the secon
false so the print statement is NOT executed.

Note: Remember that it is important that your code is correctly indented to avoid s

This can be used in simple examples like this, where we are asking for input
checking this against a present condition in our code:



```
mark = int(input("Enter test score: "))
if mark >= 50:
    print("Pass")
else:
    print("Test failed, resit please")
```

```
Ente
Pass
>>>
```

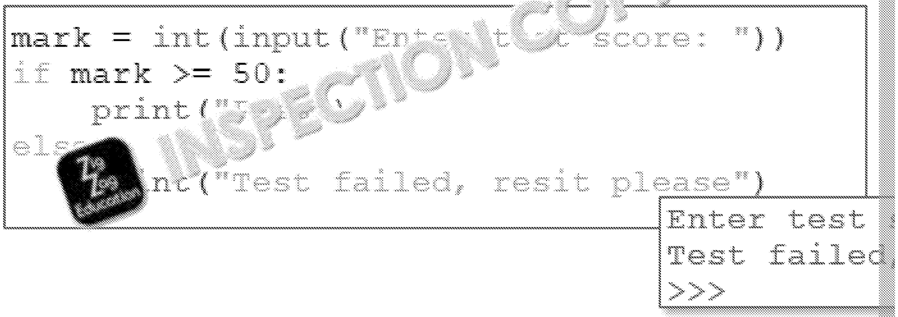
What happens if I enter a value below 50?

We need the code to be able to deal with BOTH **true** and **false** inputs.

IF ELSE STATEMENT

Using an If Else statement, I can now include a false code block so that som
condition does not evaluate to true.

Example:

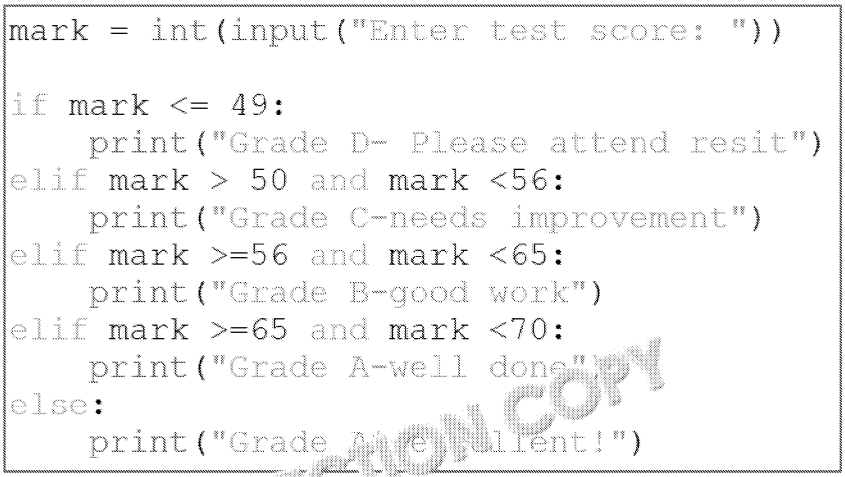


```
mark = int(input("Enter test score: "))
if mark >= 50:
    print("Pass")
else:
    print("Test failed, resit please")
```

```
Enter test s
Test failed,
>>>
```

I may want my code to check several conditions and proceed with the cond
example, a different score in the test used above will result in a different gr

IF/ELIF STATEMENT



```
mark = int(input("Enter test score: "))

if mark <= 49:
    print("Grade D- Please attend resit")
elif mark > 50 and mark <56:
    print("Grade C-needs improvement")
elif mark >=56 and mark <65:
    print("Grade B-good work")
elif mark >=65 and mark <70:
    print("Grade A-well done")
else:
    print("Grade A*excellent!")
```

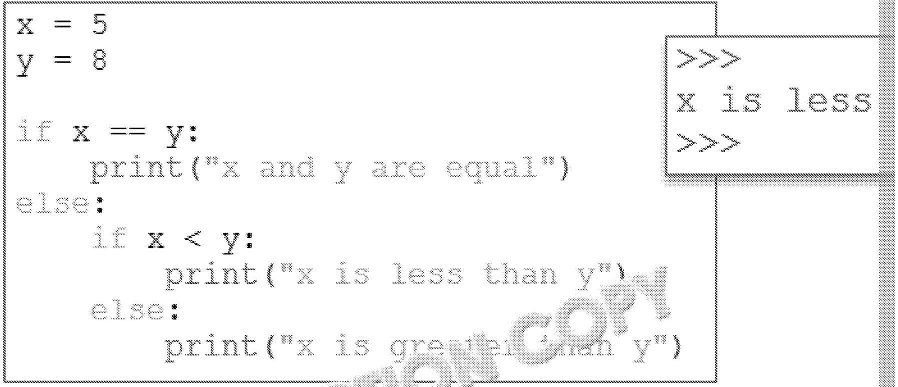
The structure of the IF/ELIF statement should follow these rules:

```
If Condition 1 = True:
    execute Code 1
elif Condition 2 = True:
    execute Code 2
else:
    execute Code 3
```

You can test any number of conditions using this method; your code does r ELSE but, if it does, it must be the last statement.

NESTING IF STATEMENTS

We can also use IF/ELIF statements to check sub-conditions in a program:



```
x = 5
y = 8

if x == y:
    print("x and y are equal")
else:
    if x < y:
        print("x is less than y")
    else:
        print("x is greater than y")
```

```
>>>
x is less
>>>
```

In this example, Condition 1 evaluates to false so the else part of the IF stat Condition 2, i.e. the first part of the nested IF statement, evaluates to true, a print statement is executed.

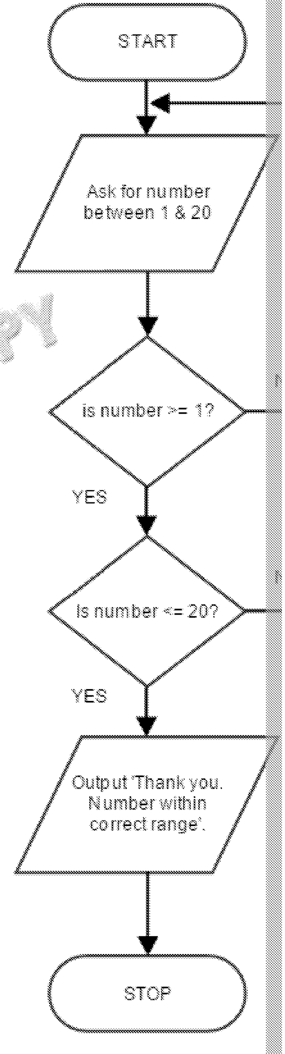
EXERCISE 08: IF/ELIF STATEMENTS

1. Complete the program shown in the flow chart.

2. Write a **program** which asks for the names of two football teams playing against each other and their scores.

   Your program should calculate how many points each team gets (3 for a win, 1 for a draw, 0 if they lose).

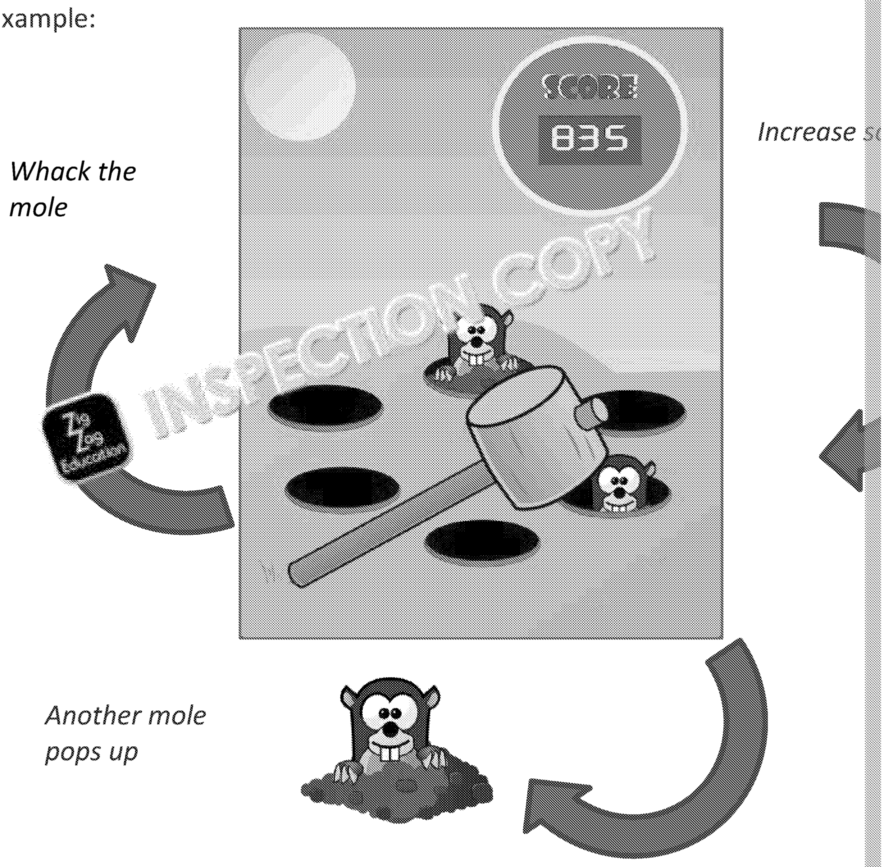
   Hint: you will need to declare constants to use in calculating the points. Remember, one team will be the HOME team, the other the AWAY team.


START
Ask for number between 1 & 20
is number >= 1?
YES
Is number <= 20?
YES
Output 'Thank you. Number within correct range'.
STOP

5.3 ITERATION

In programming, iteration means repeating instructions or processes over a
more commonly known as 'looping'.

Example:



If you do not hit the mole within a certain time limit, the game is over.

In Python, there are two types of loop that can be used, a FOR loop and a V

WHILE LOOPS

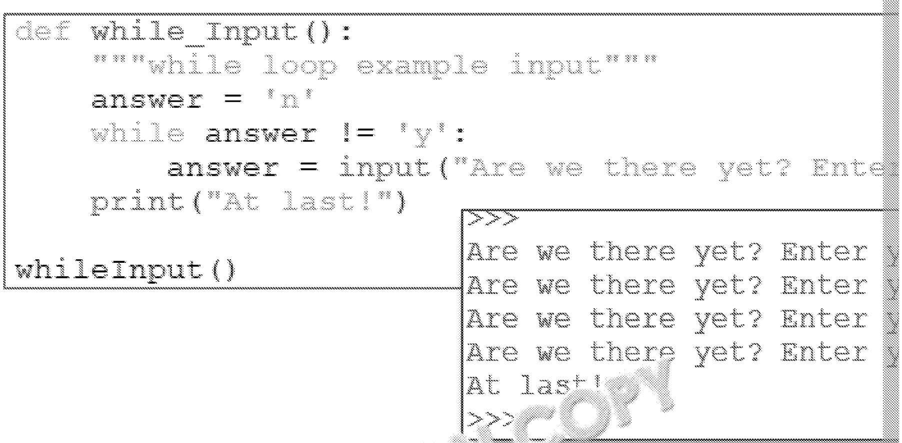
WHILE loops are also known as conditional loops as they will continue to it
condition, which you have set in your code, is met. It is important to make
written code that will allow your loop to finish and avoid an infinite loop.

Example:

```
def numberLoop():
    """ while loop example"""
    number = [illegible]          # initial value of
    while number <= 10: # the condition to
        print(number)
        number += 1     # increments the v

numberLoop()
```

If the value of the variable 'number' is not incremented by 1 each time the
condition to exit the loop will never be reached as 'number' will always be

I can also check a condition entered by a user:



```
def while_Input():
    """while loop example input"""
    answer = 'n'
    while answer != 'y':
        answer = input("Are we there yet? Ente
    print("At last!")

whileInput()
```

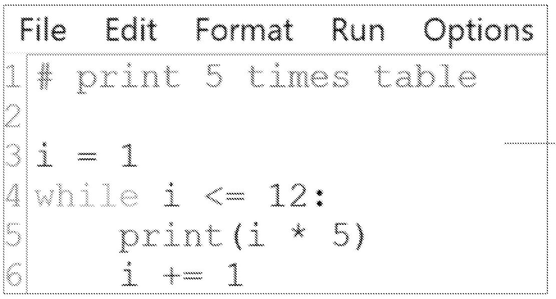
```
>>>
Are we there yet? Enter
Are we there yet? Enter
Are we there yet? Enter
Are we there yet? Enter
At last!
>>>
```

As you can see, the loop continues until the condition is met, and the final pri executed. Remember that the use of the .lower() built-in function changes my capi

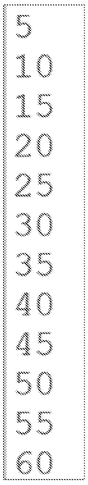
COUNTING AND TOTALLING WITH A WHILE LOOP

We can use a WHILE loop to iterate (loop) and count up (increment) or cou until a preset condition is true or false.

Example 1:



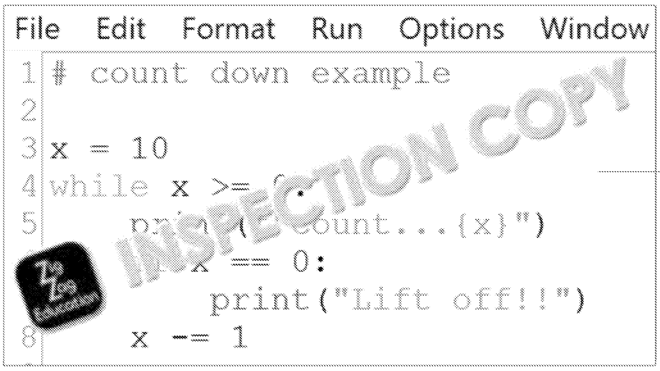
```
1 # print 5 times table
2
3 i = 1
4 while i <= 12:
5     print(i * 5)
6     i += 1
```



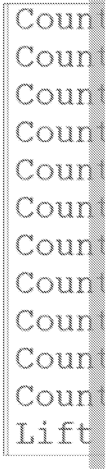
```
5
10
15
20
25
30
35
40
45
50
55
60
```

It is im value c conditi loop is the inc on line the res

Example 2:



```
1 # count down example
2
3 x = 10
4 while x >= 0:
5     print(f"Count...{x}")
6     if x == 0:
7         print("Lift off!!")
8     x -= 1
```

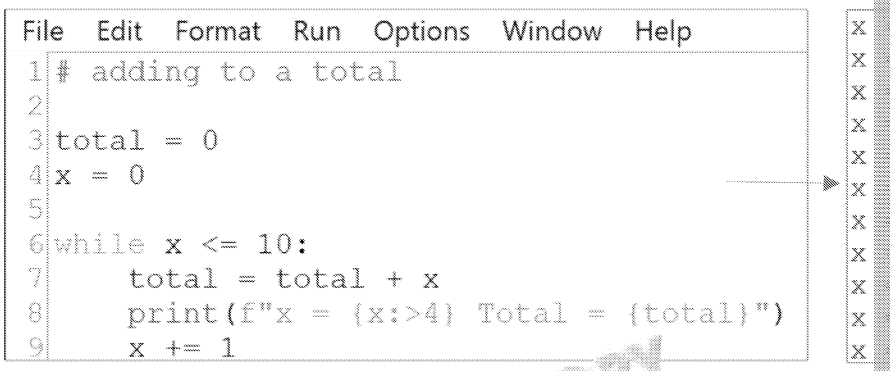


```
Count
Count
Count
Count
Count
Count
Count
Count
Count
Count
Count
Lift
```

In this example of a countdown, the initial value of x, on line 3, must be greater than 0 as we are counting down from 10. The condition for exiting the WHILE loop is set on line 4 and the value of x is concatenated into a print statement on line 5.

An additional check on line 6 checks when the count has reached 0 for an e the value of x is decremented on line 8.

Example 3:

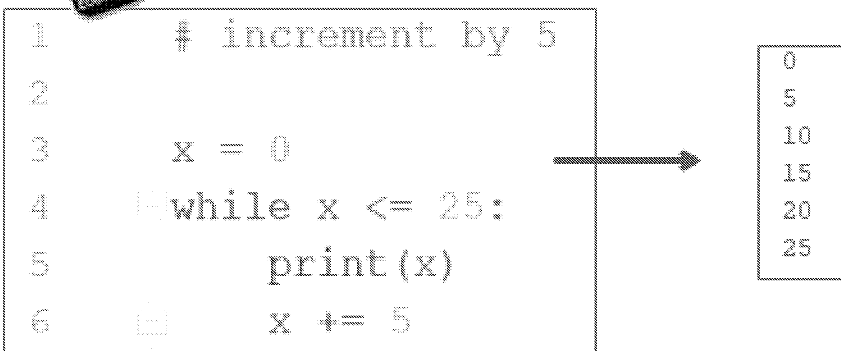


```
1 # adding to a total
2
3 total = 0
4 x = 0
5
6 while x <= 10:
7     total = total + x
8     print(f"x = {x:>4} Total = {total}")
9     x += 1
```

```
x
x
x
x
x
x
x
x
x
x
x
```

In this example, the WHILE loop counts from 0 to 10; look at line 6. In line 7, t the running total and the result is printed on line 8. The value of x is then in

The value of x can be incremented by any value; for example, if you want t 25 in steps of 5 you would do this:

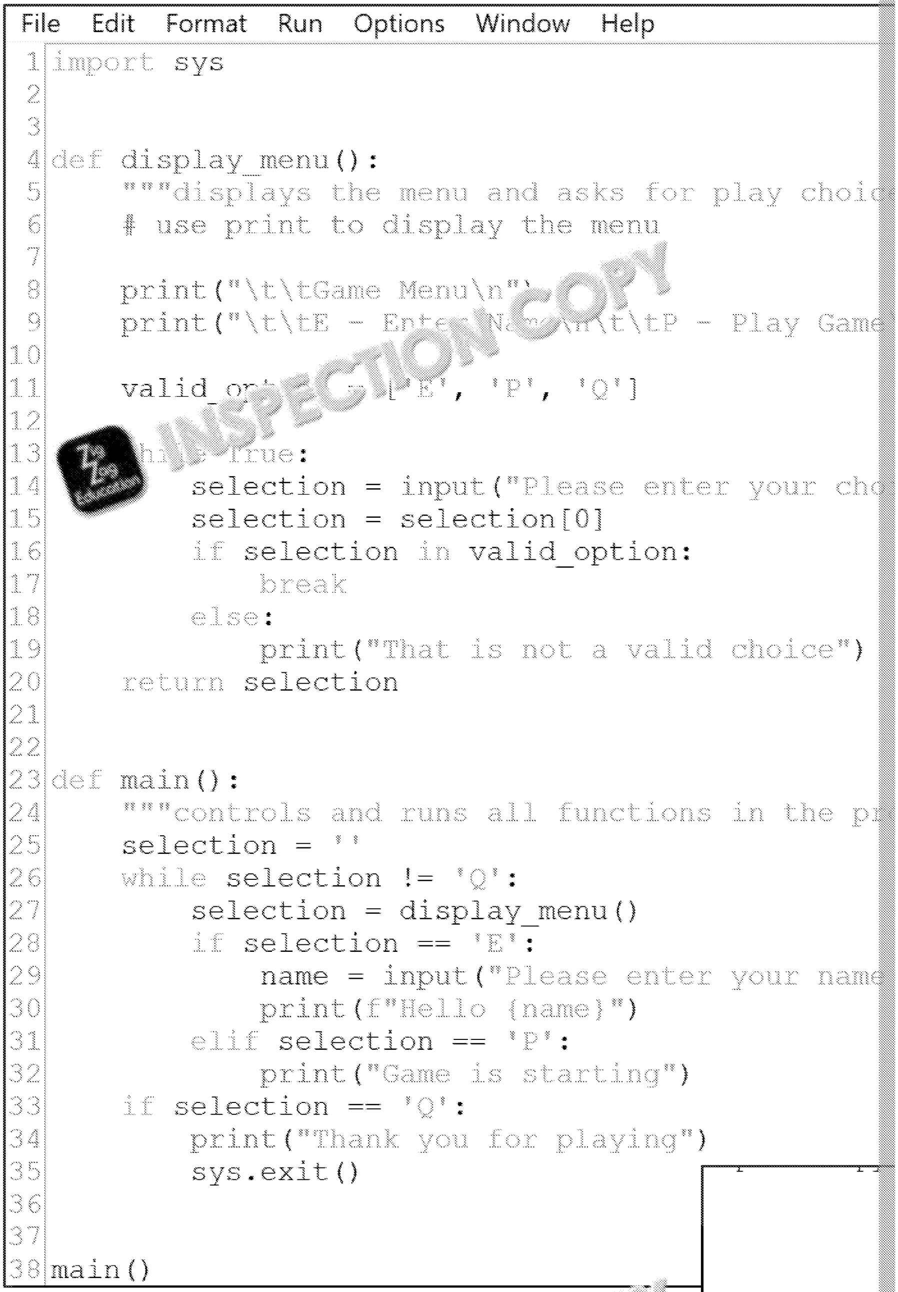


```
# increment by 5

x = 0
while x <= 25:
    print(x)
    x += 5
```

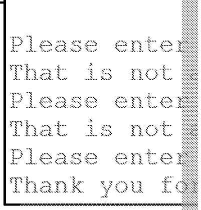
```
0
5
10
15
20
25
```

ROBUST CODE

We can also use a WHILE loop to check whether a valid input h
an ideal way to ensure that your code is robust.



```
 1 import sys
 2
 3
 4 def display_menu():
 5     """displays the menu and asks for play choic
 6     # use print to display the menu
 7
 8     print("\t\tGame Menu\n")
 9     print("\t\tE - Enter Name\n\t\tP - Play Game
10
11     valid_option = ['E', 'P', 'Q']
12
13     while True:
14         selection = input("Please enter your cho
15         selection = selection[0]
16         if selection in valid_option:
17             break
18         else:
19             print("That is not a valid choice")
20     return selection
21
22
23 def main():
24     """controls and runs all functions in the pr
25     selection = ''
26     while selection != 'Q':
27         selection = display_menu()
28         if selection == 'E':
29             name = input("Please enter your name
30             print(f"Hello {name}")
31         elif selection == 'P':
32             print("Game is starting")
33     if selection == 'Q':
34         print("Thank you for playing")
35         sys.exit()
36
37
38 main()
```



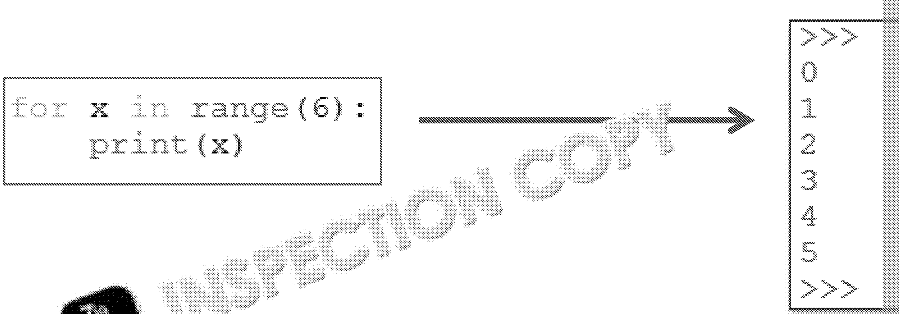
```
Please enter
That is not
Please enter
That is not
Please enter
Thank you fo
```

- On line 13 the WHILE loop will continue until a valid option is entered, i.e. an E, P or Q.
- On line 17 the IF statement is followed by a 'break' to exit the loop when a valid option is entered.

FOR LOOPS

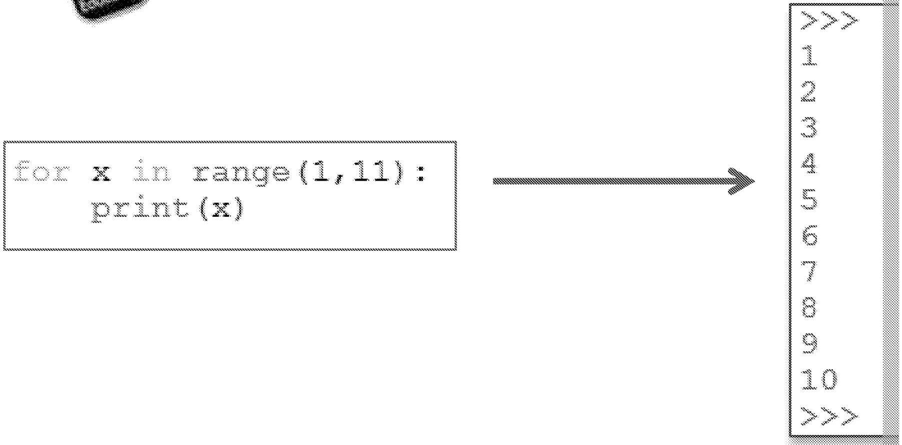
The FOR loop will loop for a set number of times, which can be a number r items in a sequence, such as a string or a list.

A common way to use a FOR loop with numbers is to use the range() built-i this built-in function we usually supply the starting number in the range an **but not include** in the range. Look at these examples:



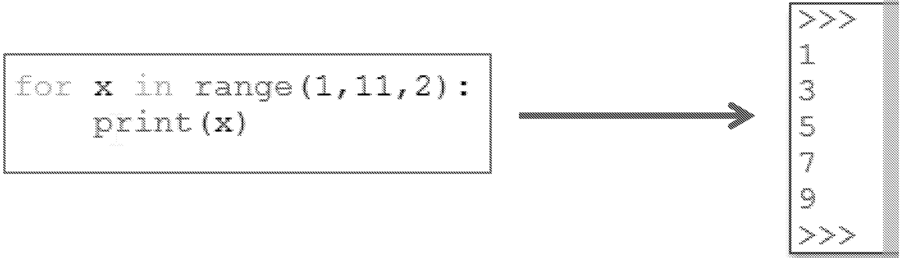
```
for x in range(6):
    print(x)
```

```
>>>
0
1
2
3
4
5
>>>
```



```
for x in range(1,11):
    print(x)
```

```
>>>
1
2
3
4
5
6
7
8
9
10
>>>
```



```
for x in range(1,11,2):
    print(x)
```

```
>>>
1
3
5
7
9
>>>
```

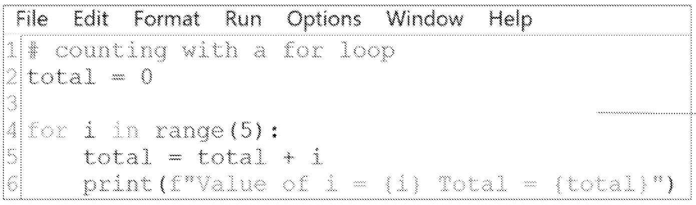
The first example has no starting point for the range of numbers so uses th six numbers from 0 to 5.

In the second example, I have supplied the starting point 1 and the end of t means up to, BUT not including, the last number.

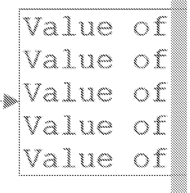
The third example sets the start and end of the range but also specifies tha increases in steps of 2.

COUNTING AND TOTALLING WITH FOR LOOPS

We can use the same techniques as counting and totalling with the WHILE


File Edit Format Run Options Window Help

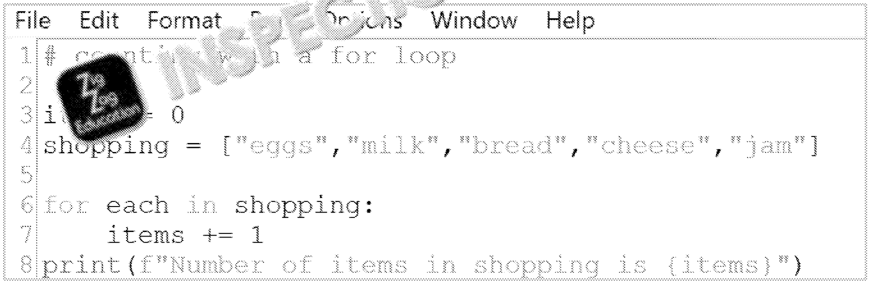
```
# counting with a for loop
total = 0

for i in range(5):
    total = total + i
    print(f"Value of i = {i} Total = {total}")
```



```
Value of
Value of
Value of
Value of
Value of
```

Using the range() method, we already know it will iterate up to, but not inc adds the current value of i to the running total

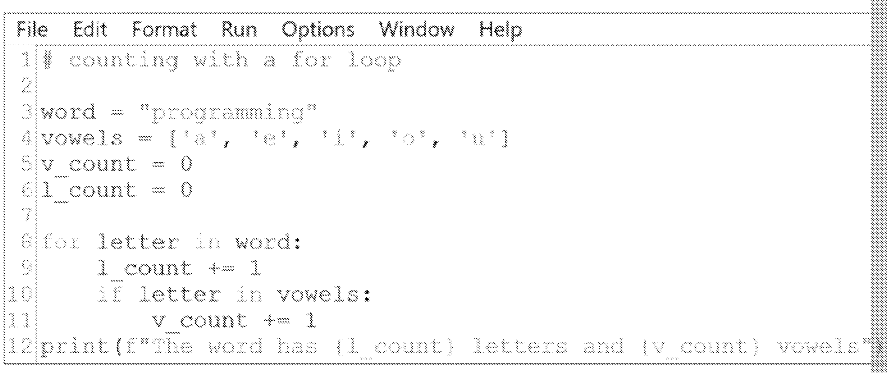
Another common method of iterating and counting or totalling uses arrays


File Edit Format Run Options Window Help

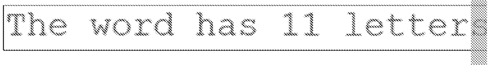
```
# counting with a for loop

items = 0
shopping = ["eggs","milk","bread","cheese","jam"]

for each in shopping:
    items += 1
print(f"Number of items in shopping is {items}")
```

The variable 'each' is used to iterate through the array 'shopping'. Each tim the value of 'items' is incremented by 1.

This method can also be used to count the characters in a string. Look at th


File Edit Format Run Options Window Help

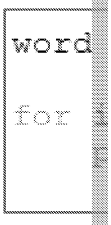
```
# counting with a for loop

word = "programming"
vowels = ['a', 'e', 'i', 'o', 'u']
v_count = 0
l_count = 0

for letter in word:
    l_count += 1
    if letter in vowels:
        v_count += 1
print(f"The word has {l_count} letters and {v_count} vowels")
```



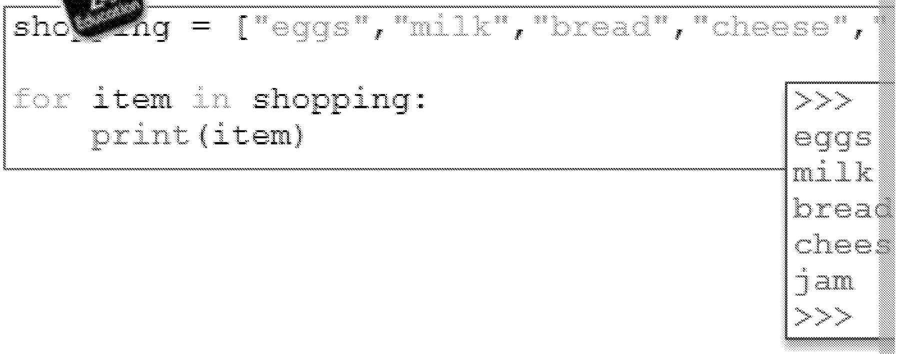
```
The word has 11 letters
```

FOR LOOPS USING STRINGS AND LISTS

Sometimes we want to iterate over data structures like a string or a list. We can do this by using a variable 'i' or 'item' to iterate over the characters in the string or the elements in the list 'a



```
word

for i
    p
```



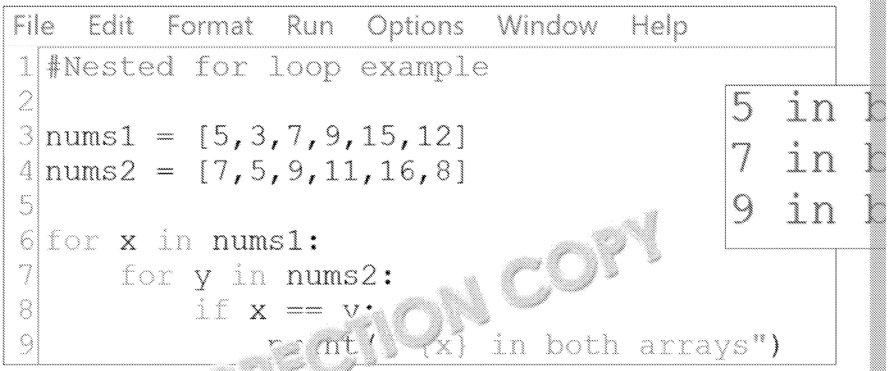
```
shopping = ["eggs","milk","bread","cheese","

for item in shopping:
    print(item)
```

```
>>>
eggs
milk
bread
chees
jam
>>>
```

NESTED LOOPS

Sometimes we want to loop through two sets of integers to compare them

Example 1:



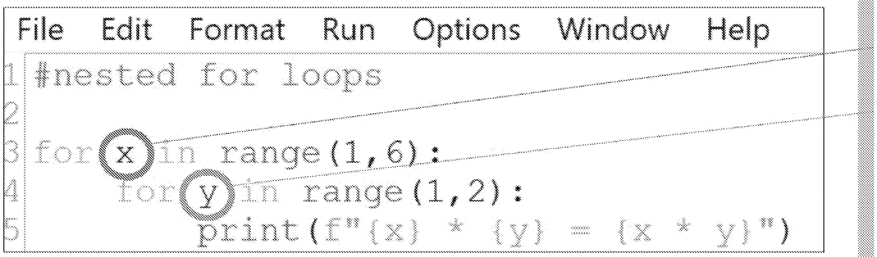
```
#Nested for loop example

nums1 = [5,3,7,9,15,12]
nums2 = [7,5,9,11,16,8]

for x in nums1:
    for y in nums2:
        if x == y:
            print(f"{x} in both arrays")
```

```
5 in b
7 in b
9 in b
```

This example finds the integers that appear in both lists; although they are can identify which are duplicated. We may be writing a program to find du can be processed in some way to solve a problem.

Example 2:

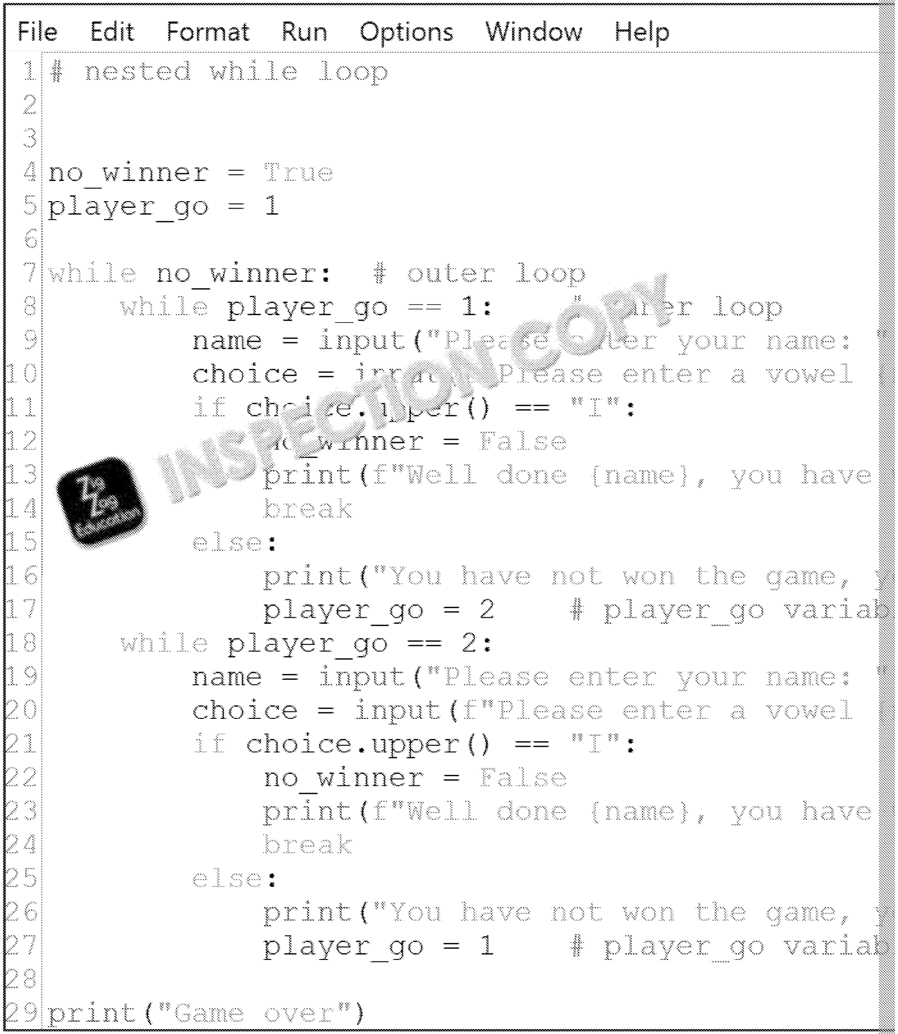
Here, x is used to count through the range from 1 to 5, y is used to count t to 2. The f-string format has been used to print out a simple times table.



```
#nested for loops

for x in range(1,6):
    for y in range(1,2):
        print(f"{x} * {y} = {x * y}")
```

Example 3:

This example shows how you might run a game loop in a program.



```
 1 # nested while loop
 2
 3
 4 no_winner = True
 5 player_go = 1
 6
 7 while no_winner:  # outer loop
 8     while player_go == 1:    # inner loop
 9         name = input("Please enter your name: "
10         choice = input(f"Please enter a vowel
11         if choice.upper() == "I":
12             no_winner = False
13             print(f"Well done {name}, you have
14             break
15         else:
16             print("You have not won the game, y
17             player_go = 2    # player_go variab
18     while player_go == 2:
19         name = input("Please enter your name: "
20         choice = input(f"Please enter a vowel
21         if choice.upper() == "I":
22             no_winner = False
23             print(f"Well done {name}, you have
24             break
25         else:
26             print("You have not won the game, y
27             player_go = 1    # player_go variab
28
29 print("Game over")
```

The two conditions that are being checked in each WHILE loop, **no_winner** the start on lines 4 and 5.

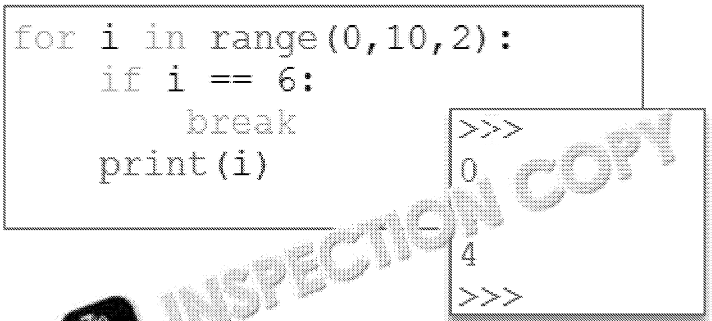
- Line 7 – the outer loop continues to check whether **no_winner** is still t
- Line 8 – the inner WHILE loop controls the turn for each player asks fo choice. This is then compared with the answer on line 11.
- If the choice entered matches the answer then the game has been wo
- Line 12 – the variable **no_winner** is set to false, a message is printed a forces the code out of the inner loop.
- The condition for the outer loop is no longer true and the code jumps
- If the choice entered does not match the answer, the 'else' part of the executed, a message is printed on line 16 and the **player_go** variable i
- The second inner loop then works in exactly the same way.

BREAKING OUT OF LOOPS

A break statement will allow us to 'break' out of a WHILE loop if a test con
example 3 above this happens if the player correctly guesses the vowel.

Here are two more examples:

Example 1:

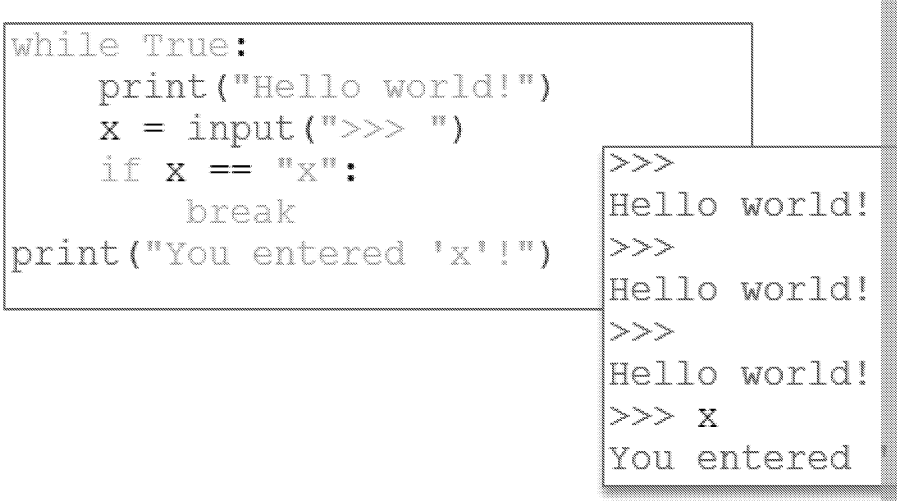


```
for i in range(0,10,2):
    if i == 6:
        break
    print(i)
```

```
>>>
0
2
4
>>>
```

This FOR loop should print from 0 to 8 in steps of 2 but is set to break out
'i' is equal to 6.

Example 2:



```
while True:
    print("Hello world!")
    x = input(">>> ")
    if x == "x":
        break
print("You entered 'x'!")
```

```
>>>
Hello world!
>>>
Hello world!
>>>
Hello world!
>>> x
You entered '
```

In this example, the use of TRUE in the WHILE loop means that until the in
break out of the loop, the program will continue to ask for an input. In the
was entered apart from hitting the ENTER key.

EXERCISE 09: WHILE AND FOR LOOPS

1. Write a program, using at least two functions AND parameter passing
   input a number between 1 and 12 and print the times table for tha

2. Write a program, using at least 2 functions AND parameter passing, t
   series of numbers until the user enters 0. The program will then displ

PRACTICE ASSIGNMENT #1: CIPH

Mr Patel is a Media Studies teacher and a keen film fan. He has set up a
to watch more films and improve their understanding of different types

He has asked students to watch some film clips, spot a range of techniq
explain where they are in each film. As he does not want students to be
comments until all students have completed the task, he wants to be ab
using a simple encryption method.

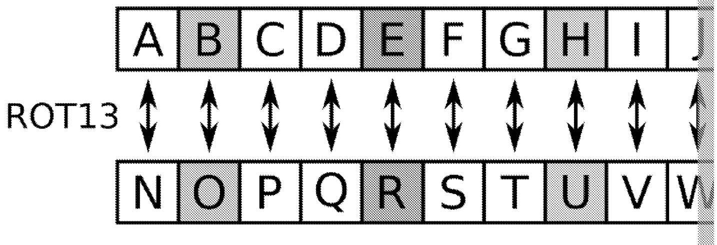
Mr Patel has decided that the ROT13 method commonly used on newsg
but needs a quick method of encrypting his comments.

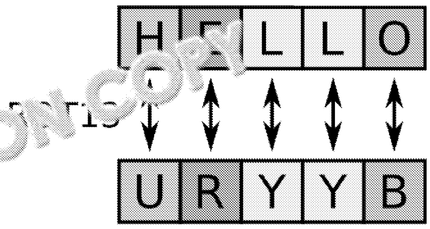
He has asked you to write an application that will:

- Display a menu option to encrypt/decrypt or quit
- Allow text input up to a maximum of 50 characters
- Find the value of each character in the message and calculate th
- Display the encrypted text to the user
- Allow text input of encrypted messages up to a maximum of 50
- Display the decrypted text to the user with the appropriate upp

Explanation of ROT13

ROT13 is a simple Caesar Cipher which can be used to obscure text by su
message with a letter 13 places down the alphabet. ROT13 only handles
and leaves other characters such as spaces and punctuation unchanged.





In the final solution, you should demonstrate the use of:

- Data validation
- Structured code
- Error handling routines