Computing GCSE

Learning to program in Python WORKBOOK

This workbook is designed for use in your Computing lessons and also your reference and guide for your independent learning/tasks.

Essentials to remember before we start programming

1. **Typing mistakes and sloppiness cannot be tolerated** in programming. The language is exact and precise. Typing lines of codes is about being logical, precise and correct.

2. Python is a **case-sensitive** programming language. That means capital letters and lower-case letters are different. To Python `A` is totally different to `a`

   Watch out for this when you start programming. It will be a common cause of programming errors when you first start out.

Adding comments to your programs

It is important to know how to add your own notes and explanations into a program. These are known as **comments** and it is very good programming style to include them to explain your programming.

Even if something seems obvious, it is still worth commenting because when you (or someone else) comes to read it some time later it may not seem half as clear. Long Python programs can be tricky to follow, and without commenting it may be hard to understand the programmer's thoughts.   Programmers say they've 'well-commented' their program if they've typed plenty of useful comments in their program.

Also, you will lose marks on your GCSE controlled assessment if you don't comment your programs. So get the habit!

Syntax:
Start a line with **#**. Anything after is your own comments, and will be ignored by the program compiler.

Program example:

```
# This a comment
# This is a place I can add my own notes and explanations
# To tell me or others what I'm doing
# Or maybe just tell you that my gran rides a Harley Davidson
# It really doesn't matter what you put.
# And you can have as many comments lines as you like
# Wherever in your program you like
```

Basic Text Output

We use the `print` command to output text to the screen. Whatever is inside the quotes will be outputted:

```
print ('something to be outputted')
```

You can also output more than one thing at once if you separate the items by a comma:

```
print ('something' , 'something else')
print ('something' , 'something else' , 'and another')
```

If you put `\n` in your output it will force a new line.

```
print ('hello world \n my first Python output')
```

Also note that the single and double quote are interchangeably. It makes no difference:

```
print (“you can also use double quotes just the same”)
```

Practise Task - LOVE LIST [1]

- Write a program to output a list of things you love.
- Add at least one comment line
- **Experiment with \n**

Variables

What are variables?

For all tasks beyond the trivial, you'll need to keep track and store information. In a computer program **variables** enable you to store information in memory for use later on.

Each variable has a name and a single thing stored in it.

What can I store in a variable?

These are three most important and common types of data you store in a variable:

- Numbers
- Text
- A true or false setting (known as a '**boolean**')

Declaring variables

Before you use variables you should aim to **declare** them, basically giving the variable a name and its starting value.

Variable naming is case sensitive, so `message` and `Message` are different variable names. Be careful with this; it is a common cause of errors and confusion as you start to program.

The syntax of declaring a variable is:

```
variablename = startingValue
```

Number variables

There are two types of number variable. Note - these are two important programming keywords:

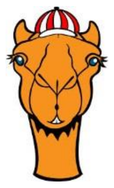
• **integer**	A positive or negative **whole** number (no decimal points), e.g. 200
• **float**	A number that has a fraction, e.g. 7.35

Some example **integer** variables being declared:

```
score = 0
lives = 5
level = 0
gamerPoints = 1000
missilesFiredToday = 0
```

Some example **float** variables being declared:

```
taxRate = 15.5
approximateDistance = 900.89
```



Good programming style is to follow the 'camel notation' when naming your variables.

Always start with a lower case letter, then each new word is in capitals (remember no spaces)

String variables

A **string** in programming is any collection of text, spaces, symbols or numbers. Declare string variables with a single quote.   Whatever is inside the quotes is known as the ‘string’:

```
name = 'Jimmy'
avatarName = 'The Killing Machine'
planetStart = 'Earth'
password = '12345XXX4$'
bestConsole = 'Sony Playstation 4'
```

Boolean variables

A **Boolean** is a True or False value.

- Note: `True` must always start with `T`
- Note: `False` must always start with `F`

```
startedGame = False
playerDead = False
bonusUnlocked = True
missiles = True
```

Outputting the contents of a variable using the print command

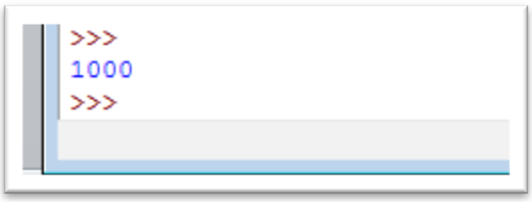
We already know the print command.   We'll now use it output the value **inside** a variable. The syntax is:

```
print (variablename)
```

For example the program:

```
score = 1000
print (score)
```

When run…

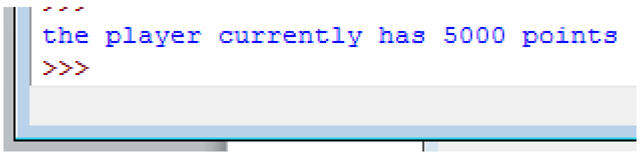


```
>>>
1000
>>>
```

You can also combine variables with text using the comma separater. For example:

```
score = 5000
print ('the player currently has',score,'points')
```

When run…



```
the player currently has 5000 points
>>>
```

Changing the value of a variable

You can change a variable's value at **any point in your program**. The syntax is just the same as declaring:

```
variablename = newValue
```

Program example:

```
# in a galaxy far, far, away...

numberPlanets = 5
numberMoons = 14
numberSuns = 3

print ('the system has',numberPlanets,'planets')
print (numberMoons,'moons')
print ('and',numberSuns,'suns')

print ('\n\n ooopps... not really!\n\n')

numberPlanets = 17
numberMoons = 34
numberSuns = 2

print ('the system actually has',numberPlanets,'planets')
print (numberMoons,'moons')
print ('and',numberSuns,'suns')
```

Practise Task – OUTPUT CHANGE [1]

Create a program where:

- at least three variables are declared
- Then the values are outputted.
- The variables are then changed.
- Then the values are outputted again.

Think you are finished? *Ok, check the following:*

1. *Variables all in camel notation?*
2. *Have you added comments to explain how your program works?*

Basic Input

We can ask the user to input something by using the `input` command.
The answer that is typed is stored in a string variable. The basic syntax is:

```
answer = input('message to display')
```

Examples:

```
name = input('please give me your name:')

whichOne = input('please choose option A or B...')

songName = input('the name of the song is? ')
```

Practise Task – IRRITATING PARROT [1]
Write a program which asks the user different questions…
… and then repeats what the user has said in annoying little phrases!!

Practise Task – CANNIBAL PROGRAM [1]
Write a program that asks the user's name, hair colour and eye colour.
Output a scary cannibal message using all three things the user typed.
Something like, "I have been watching you FRED. People with BROWN hair and BLUE eyes are very tasty in the cooking pot!"

Changing a string to a number variable

It is very important to remember that the `input` command gives back a ***string*** of what the user typed. However, what if we want to get back from the user a mathematical number so we can do some mathematical work?

The solution is to **convert** the string we receive into a number variable. We use something useful known as a ‘function’ to help us do the job:

The syntax for converting a string to an integer variable is:

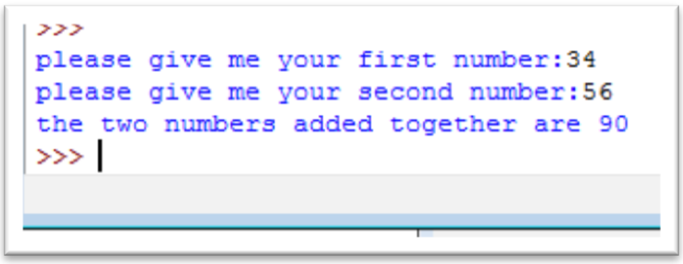
```
answer = int(stringVariableToConvert)
```

The syntax for converting a string to a float variable is:

```
answer = float(stringVariableToConvert)
```

For example this following code asks for two numbers, turns the two answers into proper numbers, adds them together:

```
# first of all ask the user for two strings
answerA = input('please give me your first number:')
answerB = input('please give me your second number:')

# convent the user typed strings into two proper numbers
numberA = int(answerA)
numberB = int(answerB)

numberC = numberA + numberB
print('the two numbers added together are',numberC)
```



```
>>>
please give me your first number:34
please give me your second number:56
the two numbers added together are 90
>>>
```

Operators

Now we know a way of storing, inputting and outputting variables, we need to know how to manipulate them. At the most basic level this is using ***operators***, which are roughly parallel to mathematical operators.

List of basic arithmetic operators

Operator	E.g.	Effect
+	a + b	*Addition* returns the value of a added to b
-	a – b	*Subtraction* returns the value of a minus b
*****	a * b	*Multiplication* return the value of a times b
/	a / b	*Divide* returns a divided by b
%	a % b	*Modulo* returns the remainder when a is divided by b

Assignment operator

We've already come across the most fundamental and important operator in the Python language which allows you to change and store the contents of a variable. The assignment operator is the **=** sign. We already know that the syntax is:

```
variableName = expression
```

However, the assignment expression could be another variable, a reference to its own value, or any combination of mathematical operators. For example

```
horsePower = donkeys * 2
happyCustomers = 200-angryCustomers
rocketLoad = missilesReady + 10
a = b * c

# the following assignments are very common in programming.
# Look closely!

# this adds 10 to the variable (i.e. it adds 10 to itself)
a = a + 10

# this subtracts 50 from the variable (i.e. it subtracts 50 from
itself)
enemies = enemies - 50

# this one adds 1 to itself
a = a + 1
```

```
# this one subtracts 1 from itself
lives = lives - 1
```

Like I said before the most important thing to remember is that the assign operator ‘=’ isn’t the same as a mathematical equality sign (what we write as ‘=’ in our Maths lessons). We learn about checking whether something is equal later in this workbook.

Order of precedence

The last thing to remember is that just like normal arithmetic, operators have precedence – in Python the precedence rules (although based on the arithmetic laws) are rather complicated, so it’s easiest to just put brackets round your expressions so you know exactly which order they execute in, for example;

```
nn = 5 * 2 + 1

# could it be?
nn = (5 * 2) + 1

#or?
nn = 5 * (2 + 1)
```

This is a simple point to grasp - remember your basic maths. Anything in brackets is calculated first. So…

```
nn = 5 * (2 + 1)
```

nn will equal 15.

PRACTISE TASKS:

SCHOOL DAZE [1]

There are 192 days in a school year. There are 6 hours at school per day. Output the total hours you spend in school from year 7 to 11.

IN THE YEAR 2050 [2]

Ask for a firstname. Ask for a surname. Ask for an age.
Then work out how old that person will be in the year 2050.
Then outputs something like "`Hi there` *`blah`*`.  You are` *`blah`* `years old now, but did you know that in the year 2050 you will be` *`blah`* `years old.`"

MULTIPLIER [2]

Create a simple multiplication calculator that allows the user to:

- type in 2whole numbers.
- Multiplies them together and outputs the answer on screen.

Tip- Try to add friendly instructions on screen as well.

AVERAGE CALC [2]

Write a program that:

- Asks the user to enter 5 whole numbers.
- Shows these 5 numbers on screen
- Then works out the average value, then displays it on screen
- Multiplies the first two numbers together and then subtracts the 3rdvalue, then displays the answer on screen.

HOW MANY MINUTES [2]

Ask the user to enter a number of days, then a number of hours. Calculate the number of minutes in these… output the number.
For an extra bonus programmer point also output the number of seconds.

TAX MAN [2]

Write a VAT Tax calculator. The program asks for a price from the user…

- It then outputs the amount the user typed,
- Then shows the amount of tax to pay (20% of that number)
- Then finally shows the PRICE + TAX amount

RECTANGLE AREA [2]

Write a program which asks the user to enter the width, then the height of a rectangle.
It then outputs the area.

REVERSE OUTPUT [2]

Ask the user to type in five individual letters. Then display the five letters in reverse order that they typed them. So what they typed in first, is outputted last etc..

KILOBYTE KING [2]
Ask for a number of kilobytes from the user. Display 2 conversions:
- number of bytes , and the
- number of bits (for the user's number)

PAY CALC [2]
Turn the following *pseudocode* into real code:
```
Ask the user to enter their hourly pay
Ask the user to enter the number of hours worked in week 1
Ask the user to enter the number of hours worked in week 2
Ask the user to enter the number of hours worked in week 3
Calculate the monthy pay
Output the monthly pay
```

CIRCLE AREA [3]
How about calculating the area of a circle by asking questions?

IMPERIAL AND METRIC CONVERTOR [between 2 - 5]
Write a program which converts between different imperial and metric amounts (e.g. asks for the user to enter a number of miles. Convert to kilometres and show the answer.)

Comparison operators

These allow two values to be compared and generates a result which is a *boolean* (i.e. True or False).

If the comparison is true, a *boolean* value of *True* is the result
If the comparison is false, a *boolean* value of *False* is the result

Syntax	In English…	Code Example	Explanation
`==`	Equal-to	`X == Y`	Returns True if X is equal to Y; Otherwise returns False
`!=`	Not-equal-to	`X != Y`	Returns True if X is not equal to Y; Otherwise returns False
`>`	Greater-than	`X > Y`	Returns True if X is greater than Y; Otherwise returns False
`>=`	Greater-than or equal-to	`X >= Y`	Returns True if X is greater than or equal to Y; Otherwise returns False
`<`	Less-than	`X < Y`	Returns True if X is less than Y; Otherwise returns False
`<=`	Less-than or equal-to	`X <= Y`	Returns True if X is less than or equal to Y; Otherwise returns False

TASK – jot down what `True` and `False` outputs you think the following program code will output (do not run it and cheat!!):

```
A = 17
B = 20

print(A==B)
print(A!=B)
print(A>B)
print(A>=B)
print(A<B)
print(A<=B)
```

Logical operators (i.e. Boolean logic)

These allow for two Booleans to be compared and generates a result which is a *boolean* (i.e. True or False)

Logical operator	Example	Explanation
`and`	`X and Y`	Returns *True* when X **AND** Y are **both** true; Otherwise returns *False*
`or`	`X or Y`	Returns *True* when X **OR** Y are true; Otherwise returns *False*

`not`	`not Y`	Returns True when Y is false;
		Returns False when Y is true;

TASK A – jot down what you think the following program code will output (do not run it and cheat!!):

```
X = True
Y = False
Z = True

print(X)
print(Y)
print(X and Y)
print(X and Z)
print(X or Y)
print(Y or Z)
print(not X)
print(not Y)
```

Compound logic

We can build up more complicated True or False statements with the use of brackets. As we know what is in brackets is worked out first.

In many programming situations we may want to do some complicated logic checks to see if something should happen or is true, for example, consider this fictional missile launch program:

```
fireButtonPressed = True
unlockSystem = False
override = True
missileSystemsOn = True
enemyInRange = True
heat = False

print( (fireButtonPressed or unlockSystem) and missileSystemsOn )

print( fireButtonPressed or (unlockSystem  or override)  )

print( (not fireButtonPressed) and (override or heat) )

print( (fireButtonPressed and override) or (heat and enemyInRange) )
```

Quick practice - jot down what you think the program code will output (do not run it and cheat!!):
Then check to see if you are correct.

Selection

Selection is another of the most fundamental concepts in programming.
It allows you to create a Boolean logial test, and depending on the `(True or False)` answer, the programmer can 'choose' different parts of the code to run.

If statement

The most important selection command is the **if** statement. The syntax is as follows:

```
if TrueOrFalsseTest:
     #these lines will only run if the test was True
     #use the TAB key to ‘indent’ the start of the line
     #indenting shows that this code is inside the IF statement
     #it is vitally important that you take care with your tabs
```

Remember - pressing the TAB key to push program in, is **very very important**. It shows that the lines of code are ‘**inside**’ the IF statement and will only run if the condition is true.

Consider this example:

```
#program line
#program line
#program line

if Test:
     #program line
     #program line
     #program line

#program line
#program line
```

Only IF the test was **true**, then the green section of code will run

The program will then carry on below after the IF section.

Program example (with comparison logic)

```
school = input(‘which school do you attend?’)

if school == ‘cavendish’:
    print('best school in Eastbourne')
    print('and Computing is the best subject')
```

Program example (with comparison logic)

```
missiles = 10

if missiles==0:
    print('you are all out of missiles')
    print('please find a storage facility')
```

Program example (with comparison logic)

```
player1score = 100
player2score = 200

if player1score > player2score:
    print('player 1 is kicking the ass of player 2!')
```

Program example (with Boolean logic)

```
ownsAPS4 = True

if ownsAPS4:
    print('What is wrong with an XBOX?')
    print('You think the PS4 is better?')

print('XBOX rules')
```

Program example (with Boolean logic)

```
inYear11 = True
collegeApplication = False

if inYear11 and collegeApplication:
    print('well done. thanks for getting your application done.')
    print('we will make sure the interview is  booked')

print('we will make sure the interview is  booked')
```

Program example (with Boolean logic and comparison)

```
player1score = 100
player2score = 200
gameOver =  True

if (player1score > player2score) and gameOver:
    print('player 1 wins the game')
```

You can start to build up complex logic, but try to **put brackets around each individual logic checks.** This will make your code clear, and prevent you from making mistakes.

```
playerMissiles = 10
onPlanet = True
rebelBaseOpen = True

if (onPlanet or rebelBaseOpen) and (playerMissiles > 6):
    print('you are ready to destroy the enemy with missiles')
```

Common errors with logic statements

Take this example, where I want to check if score A is bigger than scores B and C:

```
scoreA = 100
scoreB = 200
scoreC = 300

if scoreA > scoreB and scoreC
     print('score A is bigger than B and C!!')
```

The program above will run incorrectly. The logic test is not correctly programmed.

Each side of the `and` will be treated as a **separate** true or false results. Therefore….

```
if scoreA > scoreB and scoreC
```

This side of the `and` will give a true or false result

This side of the `and` is also a **separate** true or false result

Can you now see the problem?

The correct programming solution to test if A is bigger than B and C is:

```
scoreA = 100
scoreB = 200
scoreC = 300

if scoreA > scoreB and scoreA > scoreC
     print('score A is bigger than B and C!!')
```

This is now a proper True or False result

TECH CRITIC [3]
Ask the user for one-out-of-ten ratings for different games consoles. Depending on their answer, output different messages. Be sarcastic or wise in your output the choice is yours!

for example:

```
You rated an Xbox equal to a PS4.  Surely one is better than the other?
You also gave the Wii a rating of less than 4 out of 10.  Spot-on!
```

Else keyword

The basic if statement can be expanded using the `else` keyword. Use it to give an **alternative** to the main body of code, if the logical statement is true the first block of code is run, if it's false it's the second set. i.e.

```
if testStatement:
      # program code to run if test statement is TRUE
      # program code
      # program code
else:
      # program code to run if test statement is FALSE
      # program code
      # program code
```

Program example:

```
if race == ‘viking’:
    print('we love vikings')
else:
    print('Vikings are horrible! And you are not a viking')
```

Program example:

```
if missileLoad == 0:
    print('warning. No missiles left!')
else:
    print('you have missiles at the ready.')
```

Program example:

```
if gender == 'M':
    print('I now know that you')
    print('must be male')
else:
    print('ok, therefore')
    print('you must be a female')
```

Program example:

```
if (year10 and computingStudent) and homeworkDone:
    print('you rock!')
else:
    print('what is wrong with you?!!')
```

Program example:

```
if (friends > 10) and (age > 65) and ownsAHorse:
    print('you are an OAP cowboy with jealous friends')
else:
    print('I\’m not interested in old people without horses’)
```

PRACTISE PROGRAMMING CHALLENGES

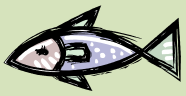
WE HATE JIM [1]

Write an unfriendly application! Ask for the person's name. If the person is called "Jim" output a rude message, otherwise tell the person they are cool! ☺

INTRUDER ALERT [2]

Write a basic password control application. Ask for two secret words. Only if the user enters the two secret words *correctly* will it output your special "secret message" on screen. Otherwise it will say "INTRUDER ALERT!!!!" and then close the program.

THREE FISHY QUESTIONS [2]



Ask three fish related questions.
For each question ask the user to type an answer.
For each answer, tell them if they are right or wrong.
At the end say how many (out of three) they got right.

THREE NUMBER SORT [2]

Ask the user to type in any three numbers. The program will then sort them into lowest to highest and output them. E.g. if the user typed 780, 12, 600. The program would output:

```
12
600
780
```

BIGGER OR SMALLER [3]

Ask the user to enter two numbers.

- If first number is bigger than the second, output a suitable message (e.g. "*X* is bigger than *Y, the difference is ___"*).
- If first number is smaller than the second, output a suitable message (e.g. "*X* is smaller than *Y by ______"*).
- If the two numbers are the same, output a suitable message (e.g. "*X* is equal to *Y"*).

USER NUMBER ENTRY [4]

Ask the user to enter a number between 1 and 20
Ask the user to enter a number between 1 and 50
If the first number isn't between 1 and 20, put an irritable message on screen and then quit. If the second number isn't between 1 and 50, put an irritable message on screen and then quit.

If all is good, now ask the user if they want to add or multiply.
If they want to add…output the two numbers added together.
If they wanted to multiply… output the two numbers multiplied together.

NUMBER 15 BLOWS THE DEATH STAR [5]

First: Ask the user the number of attempts they would like to blow up the death star (this has to be a number between 1 and 6).

Then for EACH attempt: ask the user to enter a ‘missile target number’ (which needs to be a number between 1 and 20).   If the user types 15 on any of these attempts, then tell them they’ve blown up the Death Star and the program immediately ends!

p.s. it doesn’t have to be 15 as the magic number. Change it and get your friends to play, and see if they can guess.

THE FOOTBALL LEAGUE [7]

Ask for the names of three football teams and their current league points scores.

Now output a nicely presented league table (with the teams sorted **high->low** in order of points)

Creating a random number

In certain situations we will want to create a randomly generated number. This is typical when programming games (or in any situation where the programmer wants to try to create the unexpected):

Step 1 – Getting ready for randoms…
First of all put the following right as the ***very top* line** of your program:

```
import random
```

Step 2– Putting a random number into a variable of your choice…

```
#this next line of code makes a random integer number between 1 and 10
myRandomNumber = random.randint(1,10)

#this next line of code makes a random integer number between 50 and 200
anotherRandom = random.randint(50,200)


# if you want to generate random float numbers you use UNIFORM

#this next line of code makes a random number between 3.5 and 7.5
anotherRandom = random.uniform(3.5,7.5)

#this next line of code makes a random number between 10 and 11.5
anotherRandom = random.uniform(10,11.5)
```

Waiting for a user keypress

In certain situations we will want to slow down / stop the program by getting the user to press a key. We don't need to store what they type. We **simply** want them to press a key before the program continues.   Here's how:

```
# just be saying the input statement with no variable or message
# will simply halt the program until the user presses a key

input()
```

RANDOMISER TOTAL [1]
Create three random numbers between 1 and 1000.
Output the three individual numbers to screen, then output the **sum** of all three

MOVIE MOMENTS [1]
Ask the user their name.
Randomly output **one** of four famous quips/quotes from movies (using the user's name).
E.g. `"Jim, you cannot know the power of the dark side of the force."`

LUCKY NUMBER TOTAL [2]
Ask the user to enter three numbers. Each number should be between between 1 and 5.
Now generate a random number between 3 and 15.
If the **sum** of the user's numbers equals the random number, then tell them they've hit the lucky number!

THREE STRIKE BATTLE TIME [5]
Start the battle by asking for the names of the two warriors. They both have a (randomly) generated health score of between 30 and 35.   Display these scores on screen.

Now let battle commence…
The warrors will make exactly three attacks on each other.
For **each** attack: **each** warrior's health will go down by between 1 and 10. Display their new health scores.   If **at any time** either warror has 0 health:- the program will output the winner's name and end.

SNAP CRACKLE POP [6]

Jiggle up the three words 'snap', 'crackle' and 'pop' into **any** random combination.
Now ask the user to guess the order of the words by asking three questions (i.e. 'what do you think the 1st word is?', then 'guess the 2nd word?', finally, '3rd word?')
Finally tell them which ones they guessed correctly and give a score out of 3.

Iteration (LOOPs)

Iteration is an other important programming concept. This means that a block of code can be repeated over and over again in what we call a 'loop'. There are different types of loop as you will see!

The *while* loop

A **while** loop repeats a statement block over and over again as long the *loop condition* is True.

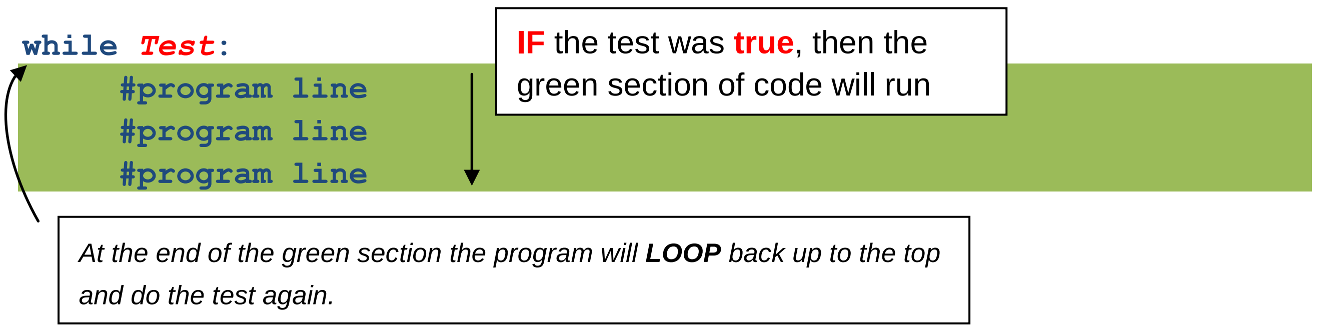
The loop condition is tested at the start.

If it is True, the statement block will run and then it will go back to the top and perform the test again. If it is True it will run again. If it is false it will break out of the loop.

The syntax:

```
while Test:
     #if the test was True
     #whatever lines of code are indented in using TAB
     #will run here as normal.
     #at the end of this block
     #it will go back and test again
     #if true,
     #and again
```

Consider this example:



```
while Test:
     #program line
     #program line
     #program line
```

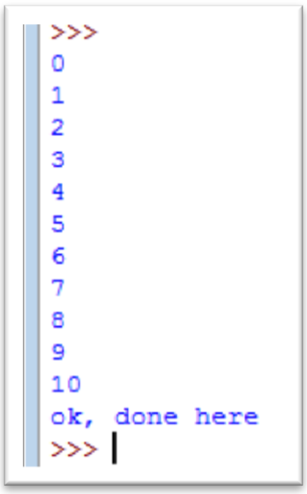
IF the test was **true**, then the green section of code will run

*At the end of the green section the program will **LOOP** back up to the top and do the test again.*

Program example A:

```
counter = 0

while counter < 11:
    print(counter)
    counter = counter + 1

print('ok, done here')
```

The output will be:



```
>>>
0
1
2
3
4
5
6
7
8
9
10
ok, done here
>>> |
```

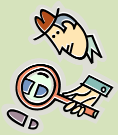
Program example B:

```
correctAnswer = False

while correctAnswer == False:
    ans = input('please select A or B')

    if ans == 'A' or ans == 'B':
        correctAnswer = True
        print('thank you for typing A or B')
```

The code will loop around and around all the time that correctAnswer is False

THOUSAND COUNTDOWN [1] – Write a program that counts down from 1000 to 1 (displaying the number on screen as it counts down). At the end the user is expected to press a key to end the program.

GO OVER A THOUSAND [2] – Write a program that keeps asking the user for a number between 1 and 100. Each time a number is entered:- add it to a running tally (and display the tally on screen). Continue doing this while the tally is less than 1000.Once the tally gets to 1001 or higher, congratulate the user on reaching over a thousand and then quit the program.



WORD GUESSING GAME [3 or 4] – Create a guess my word game.
Give the user five chances to guess a word the computer knows. Each time they guess wrong, a clue appears on screen, before they try again. If they don't guess the word after these five chances a chastising message is displayed, and the program quits.

Do this extra bit to achieve [4] points:
At the start of the program ask the user if they want to play EASY, NORMAL or HARD mode. Meaning they will get 6, 5, or 4 guess depending on their selection.

EVERRANDOM [3] – Write a program that will keep generating random numbers and displaying them on screen. After each random number is displayed, ask the user if they want another one. If they say No then the program ends, otherwise it carries on.

Extension for an extra programmer point:
At the start of the program the computer asks the user what range of random values is desired. E.g. "please enter the lowest random number you want…", "now enter the highest random number"

Extension for an extra programmer point:
Your program should watch out for a 'lucky number' coming up in the randoms. Let's say it is 7. If the number comes up, then a message will flash on screen saying something like "It is your lucky day! Did you know that 7 is a lucky number and today you will be blessed with good fortune!"

EXAM GRADER [5]
Ask the user to enter a students' exam score between 0 and 100. Keep asking them if they type something outside of the correct range of 0-100.

Once we have a score between 1 and 100. Output the grade according to these rules:
Grade A = 80+ Grade B = 70+ Grade C = 55-69 Below C = 54 or less.

At the end ask the user if they want to enter another score. If they do, run round the whole program again. Otherwise quit.

BEAT CHUCK NORRIS [6]
Let’s see if you can beat Chuck Norris (the AI player)….
You both start off with 10 points.

- The computer generates a secret random (‘X’ or ‘Y’)
- You then type your guess as ‘X’ or ‘Y’
- Chuck Norris (the AI player) then guesses ‘X’ or ‘Y’
- If you guessed right, you get 1 point added to your score. If Chuck Norris guessed right, he gets 1 point added to his score.
- If you guessed wrong, you get 1 point deducted from your score. If Chuck Norris guessed wrong, he gets 1 point deducted from his score.

Keep going with the above….. until… whoever gets to 20 points is the winner. Whoever reaches 0 points is the loser.

THREE NUMBER LOTTERY [7]

Ask the user to enter three numbers between 1 and 10.
Now generate three **different** random numbers between 1 and 10.
If the user’s three numbers matches the three random numbers, then tell them they’ve won the lottery!
Note: this task is difficult because the three random numbers all have to be different. You can’t have any of the three lottery numbers the same, e.g. 2,2,8 is not allowed!

Extension for 2 extra programmer points:
Check to see that the user’s three numbers are all different. Keep going around and around until they user has typed 3 different lottery numbers.

Lists

What is a list?

Think of a list as a collected group of variables sequenced in order. The easiest way of thinking about lists is to imagine a variable as a *box* within the computer memory; a list can then be visualised as a row of **linked** boxes.

Declaring what is in a list

Like declaring what is in a variable except we use square brackets to show we are making a list, then all the **items (sometimes known as 'elements' by programmers)** in the list are separated by commas. For example:

```
animals = ['aardvark','anteater','antelope','albert','dog','cat']
consoles = ['XBOX', 'Playstation', 'Steambox', 'Nintendo']
startScores = [0,0,0,0,0,100,0,1000,600]
planetSizes = [23.5 , 45.7, 90.56, 89.9, 12.30 ]
userChoices = [True, False, True, True, True, True, False]
```

As you can see we can make lists with all types of variable (string, integer, float or Boolean)

Accessing a single item in a list

Each list has an addressing system. We use what is known as the **index** position to access individual items.

For example, this list:

```
consoles = ['XBOX', 'Playstation', 'Steambox', 'Nintendo']
```

We take the name of the list and then use this index number in brackets to get at individual items. The index numbering starts at 0! Therefore

`consoles[0]`	contains	`XBOX`
`consoles[1]`	contains	`Playstation`
`consoles[2]`	contains	`Steambox`
`consoles[3]`	contains	`Nintendo`

Another example with a different list:

```
animals = ['aardvark','anteater','antelope','albert','dog','cat']
print('the',animals[4],'is a great friend to man')
print('the',animals[5],'purrs a lot')
print('the',animals[1],'has a snout to suck up ants!')
```

The output will be:

```
the dog is a great friend to man
the cat purrs a lot
the anteater has a snout to suck up ants!
```

Changing the value of items in a list

Basically, just think of each item in the list as an individual variable. Just assign a new value to the item.   For example, the dog in our list… needs to change to a puppy.

```
animals[4] = 'puppy'
```

Adding, removing and inserting items into a list

We need to use specialised functions to change the items in a list.

Removing an item from a list

Use the `del` function and then say which index item needs to go. For example, this line will get rid of the 3rd item in the list (which means an index reference [2]). The third item in the list is the antelope.

```
del animals[2]
```

Adding an item to the list

This is the syntax:

```
nameOfList.append( newItemToAdd )
```

For example these lines of code will add a mouse and a lion to the end of the list.

```
animals.append('mouse')
animals.append('lion')
```

Inserting an item into the middle of a list
This is the syntax:
```
nameOfList.insert( atPosition, newItemToAdd )
```

For example:

```
innerPlanets = ['mercury','venus','mars']
innerPlanets.insert(2,'earth')
```

This will insert ‘earth’ at index position 2. The new list will now be:
```
 ['mercury','venus','earth','mars']
```

Finding out how many items are in a list

The syntax is:
```
length = len( listName )
```

E.g.
```
howManyPlanets = len(innerPlanets)
```

WHICH IN THE LIST [1]
Create a list of 3 strings (storing the words “Alpha”, “Gamma”, “Beta”)
Ask the user to type in a number between 1 and 3.
Use this number as the index to display one of the 3 words.   I.e. if the user types ‘1’ it will display the 1st word in the list, ‘2’ the 2nd and so on....

SUM UP 10 RANDOMS [2]
Create a list to store 10 integer values.
Store a random number in each of the list items.
Sum up all 10 numbers.
Output the final total on screen.

VIKING NAME GENERATOR [3]

Create a two lists of strings.
Each list will contain Viking sounding *parts of names* (e.g. “alf”, “udu”, “enkson”, “ikin”, “gron”)
Randomly select two strings from the lists and mash them together to make a name (e.g. “alf”+”gron” = “alfgron”)

Iteration with the For loop

What are *for* loops?

It is quite common in programming to want to perform a certain task a fixed number of times. Although it's possible to write out the code that many times, it is a bit cumbersome and rather impractical, especially if the number of times is quite large! Also this doesn't allow you to execute the given section of code a variable number of times. The Python language contains the `for` loop which allows us to repeat a block of program code a given number of times.

Basic syntax

The syntax is as follows. The `range()` sets the number of times it goes round the loop, counting up from `startNumber` and finishing up to `finishNumber` but **not including that final number**.

You will always have a variable as part of the loop syntax.

Below it is called `currentNumber`
When the loop starts `currentNumber` will be set to `startNumber`, and each time round the loop it will increase by one, until it gets **up to** (but not including) `finishNumber` value:

```
for currentNumber in range(startNumber, finishNumber):
      # run this code
      # run this code
```

Program example:

```
for loopCount in range(1,10):
    print(loopCount)
```

`loopCount` starts with a value of 1, and each time round the loop, `loopCount` is increased by 1. The output of the program would be:

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

As you can see, `loopCount` goes **up to** 10, but doesn't include 10 for the loop.

Looping through the items in a list

In many cases we use `range` for a number to count up through the loop, however the `for` loop is a lot more flexible and allows us to step through the items in a list.

For example:

```
fruits = [ 'Apple','Pear','Plum','Orange','Peach' ]

for currentFruit in fruits:
    print(currentFruit)
```

Gives the output:

```
Apple
Pear
Plum
Orange
Peach
```

Basically, as it goes round the loop it goes through each item in the list (setting the variable `currentFruit` to the current looped item)

LOOPED RANDOMS [1]
Output 100 random numbers (use a loop).
For an extra programmer point…
Ask the user how big they want their random numbers to be.
For an extra programmer point…
Ask the user if they want to see some more random numbers. If they say **Yes**, display 5 more more, ask again,…..keep going in a loop displaying and asking until they say **No**.
And then quit.

LOOP MANIAC [1]
Ask the user if they would like to count up to 1 or 2 million. Start a loop which will count up from 1 to that many millions!

TIMES TABLE GENERATOR [2] –
Create a 'times table' program.
Ask the user to enter a number. Show a times-table for that number up to x20.

A more powerful if statement with elif

We've already learnt about using the `if`... statement to make a selection and control the flow of the program. We can now expand the power of the `if` statement with `elif` (what are known as '**else if**'s)

```
if test:
     #run this code
elif anothertest:
     #run this code instead
elif anothertest:
     #run this code instead of the others
```

Program example A:

```
x = random.randint(1,4)
if x==1:
        print('one is a good number')
elif x==2:
        print('two is a also fine')
elif x==3:
        print('three is a crowd?')
elif x==4:
        print('who likes four!?')
```

Program example B:

```
answer = input('choose A,B,C or D')
if answer=='A':
    print('you chose A')
    print('you are dead.')
elif answer=='B':
    print('you chose B')
    print('you are now a prancing unicorn.')
elif answer=='C':
    print('C was a bad choice')
    print('the potion was a potent laxative.')
elif answer=='D':
    print('great choice D')
    print('you are smiling all day.')
```

FORTUNE COOKIE [1]

Write a program to simulate a fortune cookie. The program should display one of five unique fortunes, at random, each time its run. I want you to use an ***elif*** in this program.

DICE ROLL COUNTER [2]

Write a program to tally a dice roll. A random n umber between 1 and 6 is generated **100 times**.
Finally, at the end…. display on screen how manys time 1 came up, 2 came up, 3 came up, etc.
RULE: I want you to use an ***elif*** in this program.

COMPUTER GUESSING [3]

Write a program where the user and the computer trade places in a number guessing game. The player decides a number between 1 and 100 and the computer has to guess. The computer makes a first guess, then the player then tells the computer “higher” or “lower”.
The computer has a maximum 5 chances to try to guess the number that the player has in their head!
RULE: You must use *elif* in your code at least once.

CURRENCY CONVERTOR [4].

You are going to write a currency convertor program (converting between£s and $s).

Display a menu with two options. “1. Enter a number of UK pounds 2. Enter a number of US dollars” Display an error message if they type anything other than 1 or 2. Keep asking until either 1 or 2 is pressed, then continue...
Now ask them for the amount of money (in their chosen currency). Display an error message if they type a minus number and make them choose again (and again if need be). Use a fixed conversion rate of 1.5 dollars to 1 pound. Show how them how much money in both £s and $s.
RULE: You must use *elif* in your code at least once.

Extension for 2 extra programmer points…

Ask if they want to do another conversion. If they say yes, then go do the whole thing again (taking them back to the menu).
Expand the program to cover conversion to Euros: Convert with 1EURO=£0.8, 1EURO = $1.35 show all three currencies converted as the final answer.

More on strings: useful string functions

Using string variables is very common in many programs. Therefore, it is useful to know how to trim spaces off them, find out how long they are, and so on….

Finding out the length of a string

Use the `len` function to get the number of characters in a string

```
answerInteger = len(stringVariable)
```

For example, the following program will output the number 5

```
fruit = 'apple'
howLong = len(fruit)
print(howLong)
```

Trim spaces from the start and end of a string

```
stringVariable = stringVariable.strip()
```

So basically, the variable strips itself of spaces from the start and end of itself. For example:

```
fruit = '    apple  '
fruit = fruit.strip()
print(fruit)
```

The output would be just `apple` with no spaces at start and end.

Convert all characters in the string to capitals (uppercase)

```
stringVariable = stringVariable.upper()
```

For example:

```
fruit = 'apple'
fruit = fruit.upper()
print(fruit)
```

This program will output `APPLE`

Convert all characters in the string to lower case
`stringVariable = stringVariable.lower()`

For example:

```
fruit = 'aPpLE'
fruit = fruit.lower()
print(fruit)
```

This program will output `apple`

Taking part of a string
`newString = stringVariable[startposition:endposition]`

For example, the following code takes the first three characters of a string. IMPORTANT – notice that position 0 is actually the start of the string, and it ***doesn't*** include the character in the end position.

```
fruit = 'apple'
part = fruit[0:3]
print(part)

# output will be app
# [0]  = a,  [1] = p,  [2] = p.   and remember it doesn't take element 3.
```

Another example:

```
fruit = 'pumpkin'
part = fruit[1:5]
print(part)
```

The output from this will be be `umpk`

Accessing the individual characters in the string

Think of a string as a list of individual characters. Like a list you can access the individual items by specifying the index position. E.g.

```
stringmyString = "To be or not to be"
myString[0]  // equals "T"
myString[1]  // equals "o"
myString[2]  // equals " "
```

and so on…

Converting a number to a string

Sometimes you'd want to convert an integer or float value into an actual string of characters. To do this we use the `str()` function. We give a number and get back a string.

```
answerString = str(aNumber)
```

For example, here 45 is turned into the string "45"

```
stringx = str(45)
print(stringx)
```

Also, this example converting the float value 567.45 into the string "567.45"

```
numx = 567.45
stringx = str(numx)
print(stringx)
```

WORD MUDDLE [2]

Write a program which:

- Asks for the user to enter a word. It will then be converted to upper case (and any spaces are trimmed away)
- Asks for the user to enter a second word. It will then be converted to lower case (and any spaces are trimmed away)
- Output the second letter in the 1st word
- Output the third letter in the 2nd word

BINARY TO DENARY CONVERTOR [3]

Ask the user to type in a 8bit binary number.
Convert to denary.   Display the result.

DENARY TO BINARY CONVERTOR [4]

Ask the user to type in a denary number between 0 and 255.   ***Note-***keep asking them to enter the correct number, if they type a number outside this range.

Convert to binary.   Display the result.

HEX-ELLENT [5]

Ask the user to type either 2 digits of hex OR 8bit binary. Your program will check to see whether they have typed 2 digits or 8 digits.
If they have typed 2 digits, it is therefore hex… work out the binary equivalent. Show on screen.
If they have typed 8 digits, it is therefore binary… work out the hex equivalent. Show on screen.

.

Functions

The Basic Idea behind Functions

You can divide your main program down into smaller more specific blocks of program code called '**functions**'. You give each function a name of your own choosing.

The idea is that a function should do a very particular 'job' (task) and can be run time-and-time again in your program. This will make your program code much more efficient and readable.

This code illustrates the basic syntax and use of a function:

```
def sayMyMessage():
    print('HEY DUDE!')
    print('Looking cool.')
    return
```

This is a function and it is called `sayMyMessage`

```
sayMyMessage()
sayMyMessage()
sayMyMessage()
```

The output from the program will be:

```
HEY DUDE!
Looking cool.
HEY DUDE!
Looking cool.
HEY DUDE!
Looking cool.
```

In the program I create a new function called `sayMyMessage`. I ask this method to run three times. (The programming term for this is known as "**calling**")

Any time I want to output the HEY DUDE messages I simply call `sayMyMessage()`.

Syntax for creating a very simple function:

```
def nameOfFunction():
     #program code to run
     #program code to run
     return
```

Notice that `return` statement at the end. This means the end of the function and it is time to 'return' back to where the function was called from.

IMPORTANT: any function you create in Python has to be placed at the top of your program **before** your main program starts.
E.g:

```
# functions are ALWAYS written at the top of your program file

def newFunctionHere():
      #code of the function here
      #code of the function here
      return

def anotherNewFunctionHere():
      #code of the function here
      #code of the function here
      return

def yetAnotherNewFunctionHere():
      #code of the function here
      #code of the function here
      return
```

```
#now you have the start of your program AFTER all the functions
print('ok.. now we start the main program')
```

HAPPY AND SAD FUNCTIONS [1]

Write two new functions called **`happyWords`** and **`sadWords`**.
Program **`happyWords`** to output three happy words to screen.
Program **`sadWords`** to output three sad words to screen.
In your main program call both these functions at least 3 times each.

Extension:
Adding a third function to do something of your choice.

Passing data into a function

Vital to writing useful functions is the idea of being able to pass **parameters** (data) into a function.   Have a look at the following example (in particular the highlighted code):

```python
def displayNumberMessage( numberToDisplay ):
    print('I want to show you a number')
    print(numberToDisplay)
    print('Did you like it?')
    return


displayNumberMessage( 12 )
displayNumberMessage( 14 )
displayNumberMessage( 19 )
```

The output from the program will be:

```
I want to show you a number
12
Did you like it?
I want to show you a number
14
Did you like it?
I want to show you a number
19
Did you like it?
```

Lets look at the call, `displayNumberMessage(12)`
In English think of this as "ok, I'm going to pass the value of 12 **into** the displayNumberMessage function...

… the *displayNumber* function is expecting to receive a single value and it will place it in the numberToDisplay variable.

```python
def displayNumberMessage( numberToDisplay ):
```

In English you could think of this as, "ok, so I should receive a number when I'm called. It will be called *numberToDisplay* and I can then use it in my chunk of program like any normal variable."

Syntax for passing more than one value to a function:

You can have as many parameters as you like into a function, you just have to separate each one by a comma. For example, I've just modified the code example above:

```
def displayNumberMessage( numberToDisplay, messageToDisplay ):
    print('I want to show you a number')
    print(numberToDisplay)
    print(messageToDisplay)
    return

displayNumberMessage( 12, 'Do like it?' )
displayNumberMessage( 14, 'I think 14 is a very smart number!' )
displayNumberMessage( 19, '19 was the average age of US Vietnam soldiers.')
```

Another program example:

```
def accountDetails( acNum, acName ):
    print('-------------------------')
    print('Account number',acNum)
    print('Account name',acName)
    print('-------------------------')
    return

accountDetails( 1034, 'Jim Smith')
accountDetails( 1934, 'Sally McChild')
accountDetails( 49834, 'Imran Khan')
```

Can you guess what it will output?

Parameter data is killed at the end of the function

```
def displayNumberMessage( numberToDisplay, messageToDisplay ):
    print('I want to show you a number')
    print(numberToDisplay)
    print(messageToDisplay)
    return
```

Only this function will be able to see these two variables. The rest of the program cannot use them.

They are only 'alive' for the time the function is running, whatever in them is wiped when the function is finished.

ADDER FUNCTION [1]
Write a function to add two numbers and output the result. So…
Pass two integer numbers **into** the function, add them, then print out the result.

WHAT IS IN A NAME? [2]

1. Ask the user for their name.

2. Write a function called `nameComment` and pass the user's input string **into it**.
Convert the string to upper case first of all…
… If the string is "JIM" say a suitable message.
… If the string is "SALLY" say a suitable message.
… If the string is "MARY" say a suitable message.
… If the string is "KIM" say a suitable message, etc, etc…

SORT IT FUNCTION [3]
Write a function that will accept three separate integer numbers passed into it.
The function will then sort the numbers in size order, and display a message such as:

```
The biggest number was XX
The middle number was XX
The smallest number was XX
```

Extension task (for one extra programmer point):
Call the function using three *random* numbers each time. Pass the three random numbers into the function.

The return keyword

Functions can also return a value **back** to the calling program. A normal function looks like this with the `return` statement at the end:

```
def functionName():
      #code of the function here
      #code of the function here
      return
```

However, we can return back a single value from the function easily using this syntax. Just put a value/ variable after the `return` keyword:

```
def functionName():
      #code of the function here
      #code of the function here
      return passBackValue
```

Program example:

```
def bonusCalculator( a, b ):
    bonus = (a-30)+(b+200)
    return bonus

customerA = bonusCalculator( 100, 50 )
print('customer A bonus is',customerA)

customerB = bonusCalculator( 400, 150 )
print('customer B bonus is',customerB)

customerC = bonusCalculator( 90, 80 )
print('customer A bonus is',customerC)
```

This program outputs:

```
customer A bonus is 320
customer B bonus is 720
customer A bonus is 340
```

Ok, lets take a particular look at the function call:

```
customerA = bonusCalculator( 100, 50 )
```

In English you could say this as “I’m going to be getting something returned **back** from *bonusCalculator*. When I get it, I will put that returned value into the *customerA* variable.”

FUNCTION TO TIMES SEVEN [1]
Write a function. Pass into it a decimal number. It will times that number by 7 and return it back. The calling code will use the returned number to display on screen.

DOUBLE IT FUNCTION [2]
Ask the user to enter a number. This number is passed into a function called **`doubleIt`**, the function doubles the number and returns it. This doubled number is then displayed on screen.
Extension for an extra programmer point…
Ask the user if they want to do it again (and enter another number). Keep looping, doing it over and over again, until the user says no.

FIVE GAMER NAMES [3]
Create a function called **`AskForPlayerName`** which asks for a player's name. Return a string from this function (which contains the name that the user entered).

In the main program, create a loop that runs 5 times (call the function inside the loop). Store the resulting player names in a list called **`PlayerNames`**.

Extension – ask the user if they are sure the names are correct.
- If they say "N" – loop round and get them to enter the five names again.
- If they say "Y" the program outputs the five player names, and then quits.

SMALLEST AND LARGEST [4]
Create a function called `largest`. It takes three integers and returns back the largest number.
Create a function called `smallest`. It takes three integers and returns back the smallest number.

Ask the user to enter three numbers. Use the functions to output the smallest and largest numbers.
Ask the user if they want to go again. If not, quit.

ALIEN WORLD CREATOR [5]
Create a function called `alienWorldCreator`. It randomly generates a planet size, type of planet, and other randomly generated planet based facts of your choice.

Ask the number how many planets are in the alien solar system:
Call the function that many times. I.e. if there are 6 planets in the solar system, 6 randomly generated planets will be outputted.

File input/output

A common programming requirement is to read (retrieve) and write (store) to text files.

Creating a file and then writing to it

Look at this sample code:

```
file1 = open("testfile.txt","w")

file1.write("this has been written out to the file\n")
file1.write("and also this line as well")

file1.close()
```

The first line **creates** a file called **testfile.txt** The `“w”` parameter means that we want to write to the new file. File1 is what is called the file handle. It is a direct link to the file which has been created.

Then `file1.write(“something”)` is to actually write something out to the file. Whatever is in quotes will be written to the file.

Finally `file1.close()` actually closes the file once we've done with it. This is an important last step. *Note - if you forget this line, you'll be surprised to see that your output file may be blank!*

Another program example:

```
answer1 = input('Please enter the name of the 1st Amigo:')
answer2 = input('Please enter the name of the 2nd Amigo:')
answer3 = input('Please enter the name of the 3rd Amigo:')

file1 = open("amigos.txt","w")

file1.write("These are the three amigos\n")
file1.write(answer1+"\n")
file1.write(answer2+"\n")
file1.write(answer3+"\n")

file1.close()
```

Note - the `\n` forces the output to go down onto a new line.

JOKER OUTPUT FILE [1]
Create a new file called *joke.txt* and output the text of a short joke to it.

POEM TIME [2]
Ask the user for to write a few lines of poetry
Store what the user typed into a file called *poem.txt*

DIB DAB FILLER [3]
Create a new file called *dibdab.txt*
Randomly fill the file with many lines saying ‘dib’ or ‘dab’ (in any combination)
Note – don’t go too mad!!!

- Output between 1 and 200 ‘dib’s
- Output between 50 to 70 ‘dab’s

Reading from a file

The following codes opens a file called ‘tester.txt’ and reads the contents of the file into a list called fileContent. Each item in the `fileContent` list should be a line from the file that is being read from.

```
with open("tester.txt") as f:
    fileContent = f.readlines()
```

READ A FILE [1]
Read the contents from a small text file and then output what was in the file to the screen.

READ AND COUNT [4]
Prepare for this challenge by typing into a text file some numbers (separated by spaces)
Ok…
Read the file. Count up all the numbers.
Output the total on screen.

IRRITATING PEOPLE [10]
You will create a text file called *people.txt*   This will contain a list of names (of irritating people).
You program will have 3 options to choose from:
1. View the list of irritating people
2. Remove a person from the list
3. Add a person from the list

Important: If you add or remove a person, then the file people.txt will be modified and kept up to date.

Basic error trapping

Run time errors can occur at almost any stage when a program runs, so how do you detect them and recover from them? If you don't catch errors as your program runs, your program will crash.

You can try to stop this by using something known as handling an **exception**. The word 'exception' is another word for an error.

The syntax is as follows:

```
try:
   Program code runs as normal
   .......................
except:
   BUT if there is ANY error in the try block, then this block of code will then run
   .......................
```

For example, look at this code below.
I've put a string to integer conversion inside the *try* block. This is a common cause of error (e.g. the user typing 'FRED' will crash the program because "FRED" cant be converted to an integer number), therefore I want to try and catch the error and put a suitable message on screen.

```
enteredNumberString = input('please enter a number:')

try:
    actualNumber = int(enteredNumberString)
    print(actualNumber)
except:
    print('an error happened when trying to convert to an integer')
```

ENTER A BLEEDIN NUMBER [2]:
Take the example above and add to the code:-
Add a loop so that the user has to keep entering again if they typed something unacceptable. They are told to enter a number, so that is what they have to do until they get it right!

ANY FILE MISSING? [3]:
Note – prepare for this challenge by typing a single word into 3 text files (called 1.txt   2.txt   3.txt)
Ok…

Write a program to read each words in the three text files. Output the three words to screen.
HOWEVER – what happens if you deliberately delete one of the text files?
Make sure that your program can catch this. Your program should output the phrase `[missing word]` if that text file is missing.

Extra programming challenges

GUESS IT GAME [4]
Create a number-guessing game. Your program should first generate a random number less than 1000 and then prompt the player to start making guesses.

After each guess, the player should be informed whether his guess was too low, too high, or correct.

BINARY CONVERTOR [4]
Ask the user to enter a number between 0 and 255.
The program converts to 8bit binary and displays on screen.

Extra programmer point: Ask the user if they'd like to enter again. Keep going until they say no.

INPUT ANALYSIS [5]
Write a program that tells the user to type a message on the keyboard. Your program should then:

- Show how many characters were typed by the user,
- Show the total number of vowels ("a", "e", "i", "o", or "u") that were in the message,
- What the very first character was,
- What the very last character was.

HEX CONVERTOR [5]
Ask the user to enter 2 digits of hex. Convert to denary and display.
Extra programmer point: Also convert to binary and display.
Extra programmer point: Ask the user if they'd like to enter again. Keep going until they say no.

WORD COMBINATIONS [6]
Write a program to display all possible permutations of a given input 3 letter string. Input should be a single word from the user. For example the user enters "CAT". The program outputs:

```
cat    cta    act    atc    tac    tca
```

CODED MESSAGE [7]
Write a program that tells the user to type a coded message on the keyboard (using upper case letters only).
Your program should then display that same message in lower case letters... but with each letter in the message replaced with the letter that is three places further back in the alphabet. (I.e., "D" becomes "a", "C" becomes "z", "B" becomes "y", etc.)

Programming challenge – BANK ATM MACHINE [25]

Northern Frock needs you to write a program for their new ATMs (or Automatic Teller Machines). In this version you will need to use a customer database (text file) that will initially look like this (ignore the top row):

```
ID Title First Name Surname Balance
1057 Mr. Jeremy Clarkson £172.16
2736 Miss Suzanne Perry £15.62
4659 Mrs. Vicki Butler-Henderson £23.91
5691 Mr. Jason Plato £62.17
```

Write a program that will do the following:

• Read the above data file.
• Prompt the user for their ID number, validating against the list of known users.
• If the ID number 99999 (5x 9s) is given the machine should shut down.
• Print a menu:

```
 Welcome to Northern Frock
  1 - Display balance
  2 - Withdraw funds
  3 - Deposit funds
  9 - Return card
  Enter an option:
```

• If '1' is entered, display the current balance and also the maximum amount available for withdrawal (must be a multiple of £10)
• If '2' is entered, provide another menu with withdrawal amounts of £10, £20, £40, £60, £80, £100, Other amount, Return to main menu & Return card.
• Check that the requested withdrawal is allowed, print a message to show that the money has been withdrawn and calculate the new balance. Note: make sure the new balance is outputted back out to the database text file.
• If 'Other amount' is selected then prompt the user for an integer value. Check this number is a multiple of 10 and that the withdrawal is permitted (as above).
• If '3' is entered, provide another menu that will allow the user to enter an amount to deposit(does not need to be a multiple of £10), return to main menu or return card. If funds are deposited, provide appropriate feedback and update the balance. Note: make sure the new balance is outputted back out to the database text file.
• If '9' is entered, print a goodbye message and return to the initial prompt (for the next customer).
• If another value is entered, print an error message and print the menu again.

Extension [3]: Add an extra field to the database for a 4 digit PIN which should be prompted for and checked following the entry of the ID number. The user should also have the option to change their PIN.

Extension [4]: Add another field to record whether the card is blocked. Three incorrect PIN attempts should permanently lock the card. PIN attempts should only be reset by correctly entering the PIN. Simply removing the card and trying again should not work.

Extension [4]: Create an encoding algorithm that will store the PIN in a format that cannot be read by just looking t the file. Only by running the PIN attempt through the same algorithm can the match be found.

Programming challenge – BATTLE MAP GENERATOR [30]

We are going to make a random 'battle map' for a warfare game.
It is a 10x10 grid of squares.
Each square in the grid will be either a **grassland**, **forest**, **mountain**, **hill**, or **lake**

Part 1 – randomly generate a map containing grassland, forests, mountains, hills and lake squares. Display the map to the user.

Part 2
Ok.. that was the easier bit. Now for the big challenge: You need to change your program so that your randomly generated map will always match these rules:

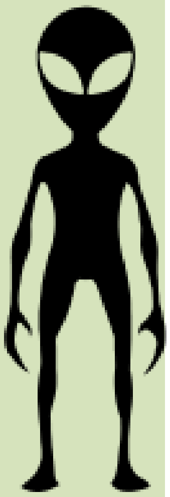
- Mountains must always be a minimum of 4 squares **together** on the map (any direction).
- Forests will always be a minimum of 2 squares **together** on the map (any direction).
- A lake square must always touch a mountain square (in any direction)
- A forest must always touch a grassland square (in any direction)

Programming challenge – SUDOKU MAKER [35]

Write a program to generate a full random Sudoku grid and show it on screen in full.

GLAXON GLADIATORS [30]

The Glaxons are an irritating alien species who are intimidating the human race. They insist on a robot gladiator battle every year.   The gladiator game is in three stages:



Stage 1. Human player equips the battle robot
Stage 2. Glaxon equips their battle robot (randomly generated AI)
Stage 3. Turn by turn fight.

Stage 1
The player has 20 credits to spend on weapons for the robot.
The player can choose from any multiples of:

1 credit - power punch. Inflicts 1 damage on opponent.
2 credit - heatmissile. Inflicts 3 damage on opponent.
4 credit - plasma punch. Inflicts 7 damage on opponent.
2 credit - force field. Protects against all attacks in the round (excluding nuke).
3 credit - hydrogen force field. Protects against all attacks this round and next (including nuke).
16 credit - Nuke. Will destroy the enemy robot outright.

Stage 2
Program some random/AI to select the Glaxon's weapons. Spend all 20 credits.

Stage 3
This is now the battle phase. Both the human and Glaxon robot's start with 10 hit points each. If the robot's hit points reaches 0, then the robot is destroyed.
For each round of the battle phase, the human player decides on one of the weapons to use, and the Glaxon chooses one too (random/AI).
Note – a weapon is a one-off attack. Once it is used it is gone from the robot's inventory.

Keep going - round by round using up weapons - until a robot is destroyed and a winner declared.

If both players run out of weapons and the robots are still alive, it is a draw.

Gain extra programmer points:

A. Put the entire game into a loop. The human player can keep going year-after-year until they are beaten by the Glaxons. How many years can the human player draw or win on the trot!?
B. Create a high-scores table in a text file. Program an option to view the high scores table (at the start or end of the game). If a player has a new high score, don't forget to output the new score to the file.

TEXT ADVENTURE GAME [points depends on complexity]

Build a text adventure game. (Think of a computer based version of the "Fighting Fantasy" books)

Some suggestions for you:

- Create variables to store your character's attributes (e.g. health score, damage ability, and luck rating). These numbers should be randomly generated.

- At the start of the adventure your character will be carrying a few items (e.g. water bottle, sword, backpack). Program a **list** to store these items. The list can change as the game progresses.

- Use random numbers to add uncertainty, e.g.
  Generate random place names
  Generate random enemy names
  Generate random treasure amounts

- Build a score variable into gaming choices.
  Add score for making good choices
  Remove score for bad choices

- Write a basic combat (or magic) system

Other challenges:

So, do you want to test your brainpower against the programmers from around the world? If you are extremely confident in what you've learnt so far, and love a mental challenge, then look at this website dedicated to programming challenges:

https://www.hackerrank.com/

(Note: you'll need to sign up for an account)

Below is another list of programming challenges which you may want to look at:
http://simeonpilgrim.com/blog/2010/05/28/programming-challenges-problems-easist-to-hardest/

If you solve ANY of them from these sites,
I'll be **VERY PLEASED** and a prize, plus a big stack of programmer points will be coming your way.

Appendix A – Math functions

Rounding a float number

You use the `round()` function on a float number to round up or down to a given number of decimal places. For example:

```
>>> round(1.923328437452, 3)
1.923
```

To round up or down (cutting away the fraction) use zero as the number of decimal places.

```
>>> round(2.7237452, 0)
3.0
```

Cutting the fraction off a float number

First of all at the very top of your program include this line (to add in the specialised maths functions)

```
import math
```

Then to cut off the fraction off any float you use `math.floor()`

```
answer = math.floor(numberToCut)
```

For example:

```
import math

full = 4.524543
fractionRemoved = math.floor( full )
```

The code above will give a result of 4.0

Appendix B – Multi dimensional lists

NOTE: You must have an understanding of Lists before tackling this topic.

What is a multidimensional list?

Basically a single item in the list is actually a list in itself (and not a single value). For example:

```
y = [ ['apples','tangy',23],
      ['pears','sharp',100],
      ['oranges','sweet',23],
      ['plums','firm',5]]
```

Accessing the individual values in a multidimensional list:

In the above example, think of `y[0]`

What is in `y[0]` ? It is actually a list… `['apples','tangy',23]`
What is in `y[1]` ? It is actually a list… `['pears','sharp',100]`
What is in `y[2]` ? It is actually a list… `['oranges','sweet',23]`
What is in `y[3]` ? It is actually a list… `['plums','firm',5]`

Let's take `y[0]` again. To get at the individual list items inside it, you then need another index position immediately after. For example:

```
y[0][0]       #  is 'apples'
y[0][1]       #  is 'tangy'
y[0][2]       #  is 23
```

So for example, this code…

```
print (y[0][0],y[0][1],y[0][2])
print (y[1][0],y[1][1],y[1][2])
print (y[2][0],y[2][1],y[2][2])
print (y[3][0],y[3][1],y[3][2])
```

… will output:

```
apples tangy 23
pears sharp 100
oranges sweet 23
plums firm 5
```

Think of a multi-dimensional list as being a bit like a co-ordinate system…"position down… position across".

Looping through the characters in a string

If you have a string and you want to access each character; Python has a nice variation on the `for` loop. Look at this example:

```
exampleString = "HI THERE JIM"

for currentLetter in exampleString:

    print("This is the current letter in the string:",currentLetter)
```

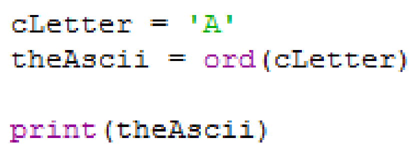
It basically goes round a loop accessing each character in `exampleString` , the individual character is placed in the `currentLetter` variable.

The output is:

```
This is the current letter in the string: H
This is the current letter in the string: I
This is the current letter in the string:  
This is the current letter in the string: T
This is the current letter in the string: H
This is the current letter in the string: E
This is the current letter in the string: R
This is the current letter in the string: E
This is the current letter in the string:  
This is the current letter in the string: J
This is the current letter in the string: I
This is the current letter in the string: M
>>> 
```

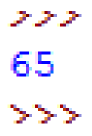
Checking the ASCII value of a character

If you have a string variable that contains a **single** character, we can find out the ASCII value by using the `ord()` function. In the example below. The `ord()` function returns back a single number. I place it in the variable `theAscii` , then print it out on screen:



```
cLetter = 'A'
theAscii = ord(cLetter)

print(theAscii)
```

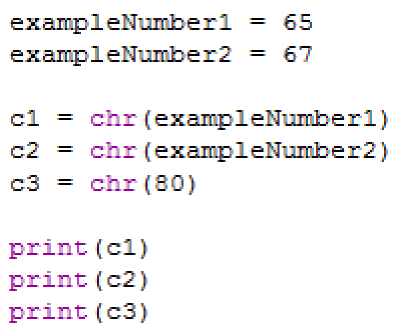
The output from the program is 65. This is the ASCII value of capital A:



```
>>>
65
>>>
```

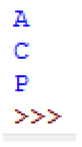
Creating a character from an ASCII number

We can use the `chr()` function to create a single character. We just pass into the `chr()` function a single number (i.e. an ASCII value). In the following example I turn the ASCII numbers 65, 67, and 90 into the characters ‘A’, ‘C’ and ‘P’:



```
exampleNumber1 = 65
exampleNumber2 = 67

c1 = chr(exampleNumber1)
c2 = chr(exampleNumber2)
c3 = chr(80)

print(c1)
print(c2)
print(c3)
```

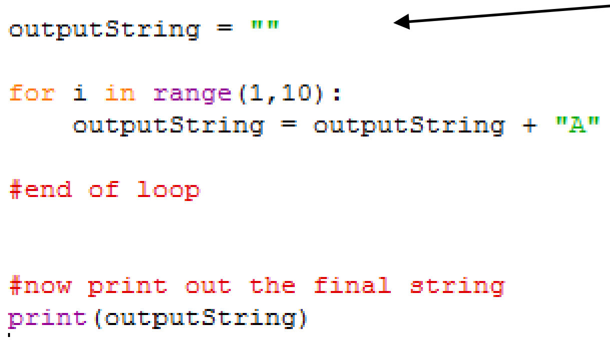
The output from the program:



```
A
C
P
>>>
```

Adding to an empty string as you go through a loop

It may not be obvious how you add characters to a string as you go through a loop. Here's a code snippet to show you the approach most programmers take:

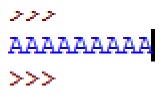


```
outputString = ""

for i in range(1,10):
    outputString = outputString + "A"

#end of loop


#now print out the final string
print(outputString)
```

You start off with an **empty** string (**before you go into the loop**)

As you **go through the loop you add** characters to the string

outputString now contains many A's. This is output after the loop:



```
>>>
AAAAAAAAA
>>>
```