Variables

Variables are where data is stored within a program. Each variable has a 'data types', that is the type of data it can contain. If you try to put the wrong sort of data into a variable you'll get an error.

There are four main types of data that can be stored in a variable -

- Integer - e.g. 3,5, any whole number
- Float - e.g. 0.2, 2.5, any decimal number
- String - e.g. "hello", "goodbye", "JDce8dk", any collection of characters
- Boolean - e.g. true or false

In Python we create variables by writing the name of a variable and setting it equal to a value,
E.g.
Total = 0
Name = "Bob"
This creates the variable and sets its data type.

Constants

A constant is like a variable, in that you use a word to represent a value, but the value cannot be changed within the code. Python doesn't really use constants, but they can be a good way of making your code more readable. E.g. rather than writing

Area = 3.14159 * radius * radius

your code will be clearer if you write

Pi = 3.14159
Area = Pi * radius * radius

Output

The basic output command is print. E.g.

print ("Hello world")

This will print Hello world in the console.
Print takes a string and outputs it to the console.

Text contained within the quotation marks is literally placed on the screen. However, if we print a variable name not in quotation marks it will print the value of the variable, e.g.

Total = 15
print(Total)

This will print 15 to the console.

Sometimes we want to combine together different things in a print statement. In order to join them together we use a comma to prevent errors. E.g.

Total = 15

print("The total is:",Total)

This will output

The total is:15

print("The total is: Total")

This will output

The total is: Total

We can combine together multiple items using the commas.

E.g.

Total = 20

Average = 4.5

print("The total is: ",Total," and the average is: ",Average)

This will work fine as the different parts are stuck together using commas.

This will output

The total is: 20 and the average is: 4.5

Input

Input means taking a value from the user. The command we use for this is input. We have to Python where to store the value that has been given to use by the user.

E.g.

NewMark = input("Please enter the next mark:")

This will display on screen:

Please enter the next mark:

When the user enters a value, it will be stored in the variable NewMark.

However, all values entered by the user will be strings. If we want to use it as a number - i.e. we want to do calculations with it, we have to convert it to a number first. There are two commands for doing this -

int converts a string into an integer

float converts a string into a float

So the above instruction would more usually be:

NewMark = int(input("Please enter the next mark:"))

Sequence

A sequence is simply a series of lines of code, where all lines are executed, in order, once. Any part of a program that is note a loop or selection will be a sequence.

Selection

A selection structure allows us to choose which code is executed, and which is skipped over. The most commonly used Selection structure is If..then...else. An If… then… else is made up of a 'condition' - a question we ask to decide which block of code to execute - followed by a first block of code (between the 'if' and the 'else'), and a second block of code (after the 'else'). If the condition is evaluated to true then the first block of code is executed, if it is found to be false then the second block of code is executed.

E.g. in Python -

```
if x<10:
        print("Less than ten")          #First code block
else:
        print("Bigger than ten")        #Second code block
```

Here, the condition is 'x<10', meaning 'is x less than 10'. If this is true then the first block of code is executed - print("Less than ten") - if 'x<10' is not true then the second block of code is executed - print("Bigger than ten").

Selections are often 'nested' to allow for more options - i.e. an If...then...else is put inside another if...then...else. To evaluate them you consider the first condition, then move to the correct code block and simply evaluate the statement within that. E.g.

```
if x<20:
                if x<10:
                        print("Less than ten")
                else:
                        print("Between 10 and 20.")
else:
                if x<30:
                        print("Between 20 and 30")
                else:
                        print("Bigger than thirty")
```

So in the above example, if x was 12 we'd first ask the question 'is x less than 20', which it is. So we'd go to the first code block. We then reach the line "if x<10:" This is false, so we go to the second code block, and run the line "print("Between 10 and 20.")"

Some programming languages also allow the use of an 'Else if' statement then introduces a new condition. E.g. in Python:

```
if x<10:
        print("Less than ten")              #First code block
elif x==10:
        print("Exactly ten")          #Second code block
else:
        print("Bigger than ten")     #Third code block
```

This gives us three code blocks. If x is less than ten we execute the first block of code. If x is exactly equal to 10 we perform the second block of code. In all other cases, we perform the third code block.

An 'else if' statement is never strictly necessary, as you can write the same code without it, e.g.

```
if x<10:
        print("Less than ten")
        if x==10:
                print("Exactly ten")
        else:
                print("Bigger than ten")
```

However, sometimes elif makes code more readable, as it doesn't require so much indentation and nesting.

Q1. Write a piece of code which -
Asks the user to enter a number, and stores that number in a variable.
If the number they enter is less than 50 output 'Small number'
If the number is 50 or larger, output 'Big number'.

Q2. Write a piece of code which -
Asks the user to enter two different numbers, and stores them both.
It then outputs the larger of the two numbers, saying 'This is the larger number.'

Q2a.
Remember, in Python, % is the mod function. That is, it gives you the remainder when you divide one number by another - e.g. 9%4 gives the answer 1, as when you divide 9 by 4 you get two, with the remainder 1. Likewise, 8%2 gives the answer 0, as when you divide 8 by 2 you get 4, remainder 0.
Use the mod function in code which asks the user to enter a number, then outputs whether it is even or not. (Hint: take their number and do %2 to it)

Q3. Write a piece of code which -
Asks the user to enter three different numbers, and stores all of them.
It then outputs the largest of the three numbers, saying 'This is the larger number.', then outputs the smallest number, saying 'This is the smallest number.'

(Hint: to do this you'll have to compare numbers together in turn - e.g. if a>b and a>c then a is the biggest number, but if a is less than c but a is bigger than b, c is the largest number)
You will have to write nested if...then...else statements. The format will resemble the following (you'll have to work out what goes in place of the *s):

```
If *>*:
        If *>*:

        else:

else:
```

Q4. Write a piece of code which -

Asks the user for two numbers, and then outputs -

- If one is bigger than the other, or
- If one is twice as big as the other.

Iteration

An iteration is the correct term for a loop - i.e. when code repeats.

There are two categories of iteration, conditional controlled and count controlled.

Conditional controlled

These loops repeat until a particular condition is met. The most commonly used loop of this kind is the while loop. For example, in Python:

```
X = 2
While X < 1000:
    X = X**2
    print (X)
```

A while loop contains a condition - or question - and a block of code following it. Whilst the condition gives the answer ‘true’, then the block of code will be repeated, over and over. At the end of the block of code, we check the condition - if it is still true, we run the block of code again, if it is false, we stop repeating the block.

E.g. 1.

```
Y = 5
While Y < 25:
    print(Y)
    Y = Y + 5
```

Output to screen is:
5 10 15 20

E.g. 2.

```
Z = 30
While Z > 0:
    print(Z)
    Z = Z - 5
```

Output to screen is:
30 25 20 15 10 5

E.g. 3.

```
Z = 30
While Z > 0:
    Z = Z - 5
    print(Z)
```

Output to screen is:
25 20 15 10 5 0

The following code is a simple program which asks the user, over and over again, to enter a number, and adds that number to a total. When the user enters -1 (or any number less than 0) then program stops running, and outputs the total of the numbers they entered.

```
Total = 0
While Num1 >0:
        Num1 = int(input("Enter a number, enter -1 to end"))
        Total = Total + Num1
Total = Total + 1
Print("The total of your numbers is:",Total)
```

(Adding 1 to the total to compensate for the -1 is a bad bodge - what would be a better way of writing this program?)

Q1. Write a program that outputs 10, 20, 30, 40, 50, by using a While loop.

Q2. Write a program that outputs 1,2,4,8,16,32,64,128, by using a While loop.

Q3. Write a program, using a while format, that outputs the three times table in the following format:

```
                1 x 3 = 3
                2 x 3 = 6
                3 x 3 = 9
up to           12 x 3 = 36
```

Hint: You'll need two variables - one to go 1, 2, 3, the other to go 3,6,9. You'll also need to format your 'print' statement correctly so that the output looks right.

```
Num1 = 1
Num2 = 3
While Num1<13:
        print(Num1,"x 3 =",Num2)
        Num1 = Num1 +1
        Num2 = Num2 + 3
```

Count controlled Iteration

In a count controlled iteration we can specify exactly how many times the loop is performed. This involves the For loop.

In Python:

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

We can also control the 'step size' in a loop, e.g. in Python:

```
    for i in range (1,20,2):
        print(i)
```
This will count up in 2s.

Finally we can use the 'For each' construct to traverse a string or an array. In Python:

```
    MyArray = [1,3,5,7,9]
    for i in MyArray:
        print(i)
```

Q1. Write a program that outputs 0,5,10,15,20,25 using a for loop.
Q2. Write a program that outputs 21,18,15,12,9 using a for loop.
Q3. Write a program, using a for loop, that outputs the four times table in the following format:

```
        1 x 4 = 4
        2 x 4 = 8
        3 x 4 = 12
up to   12 x 4 = 48

    For i in range(1,13):
```

Q4. Write a program using a for loop that asks the user to input 5 numbers, and then outputs the total of the numbers they have entered.

```
Total = 0
For i in range(0.6)
    Num = int(input
    Total =
Print (
```

Q5. Finish the following code:

```
    for i in range
        for j in range
            print("*", end = '')
        print()
```

So that it prints out the following:

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

Arrays

An array is a **data structure**. It is a way of storing multiple pieces of data under a single variable name. (In Python Arrays are strictly called 'lists', as Python doesn't have proper arrays in it - so some of the rules that apply to Arrays in other programming languages don't apply to lists in Arrays).

An Array is always referred to by a **name**, and an **index**. The name follows the same rules as the name of any other variables. However, an array name is then followed by a number in two square brackets, which tells you which part of the array to look in to find the data we are interested in. The data contained in an array is called the **'elements'** of the array. The following array contains five elements, each of which is a string.

E.g. MyArray = ["red","blue","green","yellow","orange"]

index: 0 1 2 3 4

This Array contains five elements, each of which is a string. "red" is element 0, "blue" is element 1, "green" is element 2, and so on.

This means, if we wrote the line -

```
print(MyArray[2])
```

then "**green**" would be printed.

Likewise, print(MyArray[4]) will print "orange" on the screen.

Often, we use a variable to control the index within an array. An integer variable is used in the brackets to control which element of the array is accessed.

E.g.

```
Num1 = 4
print(MyArray[Num1])
```

This will print "yellow" on the screen.

The most common use for this is putting an Array into a for loop, e.g. using the same array again we can print all of the Array contents by doing the following:

```
for i in range(0,5):
    print(MyArray[i])
```

This would print:

red
blue
green
yellow
orange

This is because the variable i is taking each value - 0,1,2,3,4 - as the loop repeats. We therefore print MyArray[0], then MyArray[1], MyArray[2], and so on.

Loops can also be used to read data into Arrays. E.g.

```
for i in range(0,5):
    MyArray[i] = input("Please enter a colour:")
```

This code would ask the user to enter 5 colours, and store them in MyArray.

A useful function built into Python is len - this returns the length of an Array. So, using the above example, len(MyArray) would return 5. Therefore, the above code could be re-written as:

```
for i in range(0,len(MyArray)):
    print(MyArray[i])
```

This code is more versatile, as it works no matter how big MyArray is.

We can also use the 'for each' construction of the for loop, as mentioned above:

```
for i in MyArray:
    print(i)
```

This will loop through each location in the array and then stop, without us having to mention the index at all.

In most programming languages, every element in an array must be of the same data type. In Python, as arrays are really lists, each element can be a different data type.

Q1. Write a piece of code which uses this array

MyNumbers = [2,4,6,8,10,12,14]

Write code that outputs these numbers in order using a loop.

Q2. Using the array

MyNumbers = [2,4,6,8,10,12,14]

Write a piece of code that outputs the number in reverse order.

Q3. Starting with the array MyNumbers = [0,0,0,0,0,0,0,0]

wite a piece of code that asks the user to input 8 numbers, stores them in the array, and then outputs them again in the order they were entered.

Advanced Problems

Thief!
A thief has managed to find out the four digits for an online PIN code, but doesn't know the correct sequence needed to hack into the account. Design and write a program that displays all the possible combinations for any four numerical digits entered by the user.
The program should avoid displaying the same combination more than once.
Ensure the output is formatted in an easy to read and understand way.

Fruit Machine
Write a program to simulate a Fruit Machine that displays three symbols at random from Cherry, Bell, Lemon, Orange, Star, Skull (or whatever you can best represent them as using the Python character set).
The player starts with £1 credit, with each go costing 20p. If the Fruit Machine "rolls" two of the same symbol, the user wins 50p. The player wins £1 for three of the same and £5 for 3 Bells. The player loses £1 if two skulls are rolled and all of his/her money if three skulls are rolled. The player can choose to quit with the winnings after each roll or keep playing until there is no money left.

Arithmetic test
A primary school teacher wants a computer program to test the basic arithmetic skills of her students. Generate random questions (2 numbers only) consisting of addition, subtraction, multiplication and division. The system should ask the student's name and then ask ten questions. The program should feed back if the answers are correct or not, and then generate a final score at the end.
Extensions:
1. Extend your program so that it stores the results somewhere. The teacher has three classes, so you need to enable the program to distinguish between them.
2. The teacher wants to be able to log student performance in these tests. The teacher would like the program to store the last three scores for each student and to be able to output the results in alphabetical order with the student's highest score first out of the three. (The extension here is where it gets tricky/interesting - you'll have to look into how to output data to a text file you save on your computer. We had a brief look at this last year, but it's worth knowing how to do to... pretty easy, but it'll just take a bit of research)