Learning to Program with Python

In order to start Python you need to: <ask your teacher>. Before you begin make sure you know what these mean: prime, factor, multiple, square number, lowest common multiple, highest common factor.

Python the Calculator

Python can be used just as a calculator. You type the calculation you want to do in, press Enter and it will give you the result.

You need to know that instead of a x sign for multiplication you use *.
For division instead of ÷, you use /.

Try these calculations:
a) 192 x 39 **b) 782 x 3 – 292** **c) -32 + 29 x 2**

Variables

Sometimes we want to use "variables", which are just a name for something so we don't have to keep typing it in. For example, I might want to tell Python that x is equal to 39292. That way I can just type x instead of 39292. I do this by typing:

```
>>> x = 39292
```

When I want to say that x is equal to a number, I call this assignment.

It is not fixed what number x is. I can change it just by setting it to a different number:

```
>>> x = 293
```

Use Python to assign these values to each variable:
a) a is 39 b) b is – 7 c) c is 2.5 d) d is 0.394

I can now use these variables in calculations. Use Python to find out the answers to:

a) a*b b) b + c – d c) a*b + c*d

I don't have to use just one letter for my variables, and they don't have to be numbers. They can be what we call "strings":

```
>>> first_name = "bob"
>>> surname = "smith"
```

Using Python, set two variables, first_name and surname equal to your first name and surname.

Printing

If I want Python to display those then I'd have to use the "print" function. If I wanted to print first_name I would type in:

```
>>> print(first_name)
bob
```

The brackets are very important so make sure you use them. If I wanted to print my full name I could just ask Python to print both names:

```
>>> print(first_name, surname)
bob smith
```

This print function is really useful and you can use it to print numbers as well.

Mod Operator – Finding the Remainder

We have looked at doing things like * and / with numbers, but there is also a very useful operator called "mod" which tells us the remainder when we do a division.

To use "mod" I use the symbol "%". If I want to find out the remainder when I do 29 divided by 7, I type in:

```
  >>>    29%7
```

And Python answers:

```
1
```

If I wanted to know the remainder when 2932023 was divided by 3, I would type in and get the response:

```
  >>>    2932023%3
0
```

That means the remainder is 0, and so I know that 3 is a factor of 2932023.

Use Python to work out the remainder of these calculations:
a) 293 / 4   b) 981 / 3   c) 1284 / 4   d) 2932932 / 5

Now use Python and the % operator to work out which of these numbers 3 goes into:
92, 922, 9222, 92222, 922222

Find out if 7 is a factor of these numbers:
a) 2 933   b) 8 746   c) 20 461

Primes

Before you continue, just make sure that you remember what a prime number is.

In order to check if a number is prime, we need to make sure that it only has two factors.

We can use the % operator to work this out.

For example, if I want to see if 8 is prime, I could do:

```
>>> 8%1
0
>>> 8%2
0
```

This immediately tells me that 8 is not prime because 2 goes into it!

Use Python and the % operator to show that 17 is prime.

What number do you have to check up to to see if 89 is prime? All the way to 89? Lower?

Now I want you to use Python to see if 89 is prime.

If and Else

Sometimes you want to do different things depending on the answer to a calculation. For example, if 7 is a factor of a number then you might want to print the sentence “7 is a factor!”, but if 7 is not a factor then you might want to print “7 is not a factor”.

We use the if statement to do this. So if I wanted to print "yes" if 7 is a factor of 21 then I would write:

```
>>> if ((21%7) == 0) : print("yes")
```

and Python would answer:

```
yes
```

Let's look at that statement in more detail:

- The bit inside the first brackets checks if 7 is a factor of 21.
- The 21%7 will be 0 if 7 is a factor of 21.
- There are brackets around 21%7 because it is a calculation and we must brackets round it in case we confuse Python.
- The == 0 just checks if it is equal to 0. Make sure you use == not just one = as they mean different things.
- The colon (:) tells Python what to do if the bit inside the brackets is true. In this case it will print yes.

If I wanted to print "Correct" if 12 * 8 is 96 then I would type in:

```
>>> if ((12*8) == 96): print("Correct")
Correct
```

Write some if statements in Python that:

a) print "Yes" if 8 is a factor of 248
b) print "Correct" if 420 divided by 14 is 30

Sometimes we want to do one thing if the condition is true, and another if it is false. For example, I might want to print "Yes" if 7 is a factor of 21 and "No" if it is not.

```
        >>> if ((21%7)==0) : print ("Yes")
        else : print ("No")
```

and Python responds:

```
        Yes
```

If I wanted to check if 112 was a factor of 203 208, I could type in:

```
        >>> if ((203208%12)==0) : print ("Yes")
        else : print ("No")

        Yes
```

Write if statements that print "Correct" or "Incorrect" depending on the result of these:

a) is 7 a factor of 92 852 **b) is 9 a factor of 65 952**

We can use variables in our if statements.

```
        >>> x = 81
        >>> if ((x%7)==0) : print("Yes")
        else : print ("No")
```

Use Python to do the following:

Set x = 293293
Write if statement that will print "Yes" if 7 is a factor of x

Write if statement that will print "Yes" if 7 is a factor, "No" otherwise

Write if statements to print "Yes" if 2,3,4,5,6,7,8,9,10 are factors of x

While Loops

So far we have had to repeatedly type in the same thing. But programming is supposed to make things easier and the ***While*** loop is the answer. It allows us to repeatedly do a task.

The way it works is to have some test condition, and we keep doing the task until that test is false.

Here is an example loop:

```
>>> n = 0
>>> while (n < 5) :
        print(n)
        n += 1
```

Let's go through this loop:

- n = 0 just sets a variable n equal to 0. We often call it the "loop variable".
- while (n < 5) : means that we will keep doing the instructions below as long as the variable n is less than 5. As soon as it is 5 or more we will stop.
- print(n) just prints the number n
- n += 1 adds 1 to the value of n, so each time we do the instructions n gets one bigger.

Python will respond by printing:

```
0
1
2
3
4
```

So for the last thing you were asked to program:
we could have just written the loop:

```
>>> n = 2
>>> while (n < 11) :
        if((x%n)==0) : print ("Yes")
        n += 1
```

This would have made life much easier!

We could have made it much more obvious which ones were the factors by writing:

```
>>> n = 2
>>> while (n < 11) :
        if((x%n)==0) : print (n, " is a factor of ", x)
        n += 1
```

Python would respond:

```
2 is a factor of 2939292
3 is a factor of 2939292
4 is a factor of 2939292
6 is a factor of 2939292
9 is a factor of 2939292
```

Here are some challenges for you: (HINT: make sure your loop variable is set to 1 to start with)

Write a while loop to check if the numbers 1 to 15 are a factor of 87832

Write a while loop to check if the numbers 1 to 10 are a factor of 89

Write a while loop to print the first 10 multiples of 8.

Function Definitions

We are still repeatedly typing in the same code! There is a way of giving a name to a piece of code so you can just use that name instead of typing it in. This is called function definition.

For example, I might want to write a function to say if 7 is a factor of a number. I would type in:

```
>>> def factor7(n) :
        if((n%7)==0) : print("7 is a factor of ", n)
        else : print("7 is not a factor of ", n)
```

In the first line "def" just tells Python we are going to define what a new word means. The name of the new word is factor7. The (n) means that we will need to be told a number n that we will use in our function. Here, n is called an "argument" to the function. We can have as many "arguments" as we need.

Then to check if 7 was a factor of some numbers, I would type this in and get some answers:

```
>>> factor7(42)
('7 is a factor of ', 42)
>>> factor7(12)
('7 is not a factor of ', 12)
```

Write a function to check if 12 is a factor of a number n, and then try it on the numbers 8000, 288, 312, and 4202

Write a function to check if n is a factor of 2000, and then try it to check if 8, 22, and 400 are factors of 2000. (HINT: read it very carefully, it is different to the question above in that the factor is the variable here)

We can make our function even more general, by having two arguments. We can write a function that will check if a is a factor of n:

```
>>> def factor(a,n):
        if((n%a)==0) : print("Yes, ", a, " is a factor of ", n)
        else : print ("No, ", a, " is not a factor of ", n)
```

To use the function to check if 8 is a factor of 1272 we would just need to type in:

```
>>> factor(8,1272)
```

And Python would respond:

```
Yes, 8 is a factor of 1272
```

Type the function definition in and then use it to check the following:

a) is 12 a factor of 28824 **b) is 19 a factor of 1929382**

Now use the function to see if the numbers 2 to 10 are factors of 89.

Did you use a while loop for that? If not, go back and do it again with a while loop.

Now write a function that takes an argument n, and uses a while loop to see if the numbers 2 to 10 are factors of n. It should print any numbers that are a factor.

This is very tricky, but try experimenting and seeing if you can write a function to check if a number is prime. You will need to take one argument, n, the number to check if it is prime. You will need to keep track of how many factors the number has between 1 and n. Name a variable numberoffactors, and everytime a number is a factor, increase it by 1 (numberoffactors += 1). At the end of the function, you will need to check and see if it has more than 2 factors before you print if it is prime or not.

Lists

Sometimes we don't want to just have one number in a variable, but have a whole list of numbers. For example, I might want to have a variable that holds all the factors of a number in a list.

Python makes this really easy. If I wanted to store the numbers 2, 3, and 5 in a variable called x, I would just type in:

```
>>> x = [2,3,5]
```

If I then wanted to add another numbers to the list, 7 say, then I would type in:

```
>>> x.append(7)
```

Append just means add 7 to the end of the list. If I then typed in:

```
>>> print(x)
```

Python would respond:

```
[2,3,5,7]
```

If I wanted to know how many numbers were in a list, I would type in:

```
>>> len(x)
```

And Python would respond:

```
4
```

The function len() means what is the length of this list?

The items in the list are numbered starting with 0. To pick out a particular number from the list is really easy. If I wanted the first item I would type in:

```
>>> x[0]
2
```

If I wanted the third item, I would type in:

```
>>> x[2]
5
```

Use Python to do the following:
a) Create a list called squares with the first 5 square numbers in it.
b) Print the list
c) Print the first, third, and fifth numbers from the list.
d) Append the number 36 onto the list.
e) Find the length of your new list.

It is really easy to combine both lists and while loops. If I had a list called mylist with the numbers [1,2,3,4,5,6] in it, I would type in:

```
>>> mylist = [1,2,3,4,5,6]
```

Then if I wanted to go through the list and double all the numbers I could create a while loop. I'll start by making a variable x that starts at 0.

```
>>>  x = 0
```

Now I'll write my loop:

```
>>> while (x<len(mylist)) :     (* this line loops through the length of the list*)
        mylist[x]  *= 2          (* this line doubles the xth number in the list*)
        x += 1                   (* this line moves us on to the next number *)
```

If you type this in and then type in print(mylist), you will get the new list:

```
>>> print(mylist)
[2,4,6,8,10,12]
```

Can you create a list called mylist, with numbers [1,3,5,7,9,11], and then write a while loop that will add 1 to each number in the list?

Can you now set the variable x = 1, the variable factors = [], and the variable mynum = 120?

Now write a while loop that will check for all the numbers from 1 to mynum, to see if they are factors of mynum? If they are factors then they should be appended to the list factors.

Can you now turn this into a function definition, called factorlist(n) that takes the number n and then puts all of its factors into a list called factors? Before you start, make sure you have set the list factors = []:

```
>>> factors = []
>>> def factorlist(n): (* now write your code *)
```

Prime Factors

You should have learnt in your lessons about how to do the prime factorisation of a number. Here are some examples:

30 = 2 x 3 x 5
100 = 2 x 2 x 5 x 5
4000 = 2 x 2 x 2 x 2 x 2 x 5 x 5

We want to write a function, primefact, that takes a number n, and creates a list, primes, that computes the prime factorisation of n.

For example, primefacts(30) would create a list [2,3,5]. primefacts(20) would create the list [2,2,5].

You will want to start your code with:

```
>>> def primefact(n) :
        x = 2                (* we will start looking for factors at 2 *)
        primes = []          (* list of prime factors is empty to start *)
        ???
        ???           (* put your code in now *)
        ???
        ???
        print(primes)   (* print the list of prime factors *)
```

You need to write the correct code in. If you are really stuck then think about how you would do this with pen and paper using a tree diagram. HINTS: we will need some sort of while loop; when doing a tree we keep using 2 until we can't divide by 2 anymore, then move on to 3 etc.

When you have finished, test your function on a few examples. Try primefact(200), primefact(4000), and primefact(39283202).

Further challenges:

1) Write a function hcf(a,b) that finds the highest common factor of a and b. You could use your primefacts() function, or do it some other way.
2) Write a function lcm(a,b) that find the lowest common multiple of a and b.
3) Write some code to find the largest common factor of 4200 and 8950.
4) Write some code to find the first number that is one more than a square number and a multiple of 3.
5) Write some code to find all the numbers that are factors of 4000 and a square number.
6) Write some code to find the lowest common multiple of 220 and 4800.
7) Write a function, isprime(n), that prints "Prime" if n is prime, and "Not prime" if n is not a prime.