

## Introduction To Programming Logic

### Using Python

#### SUMMARY

#### ***Introduction To Python:***

When you start learning any language it's a basic tradition to start with the 'Hello World' program i.e. printing some text on the screen.

In python the program is as follows:

```
print("Hello World ! ")
```

That's it. It's just a one line code in python. We can divide the above statement into 2 parts

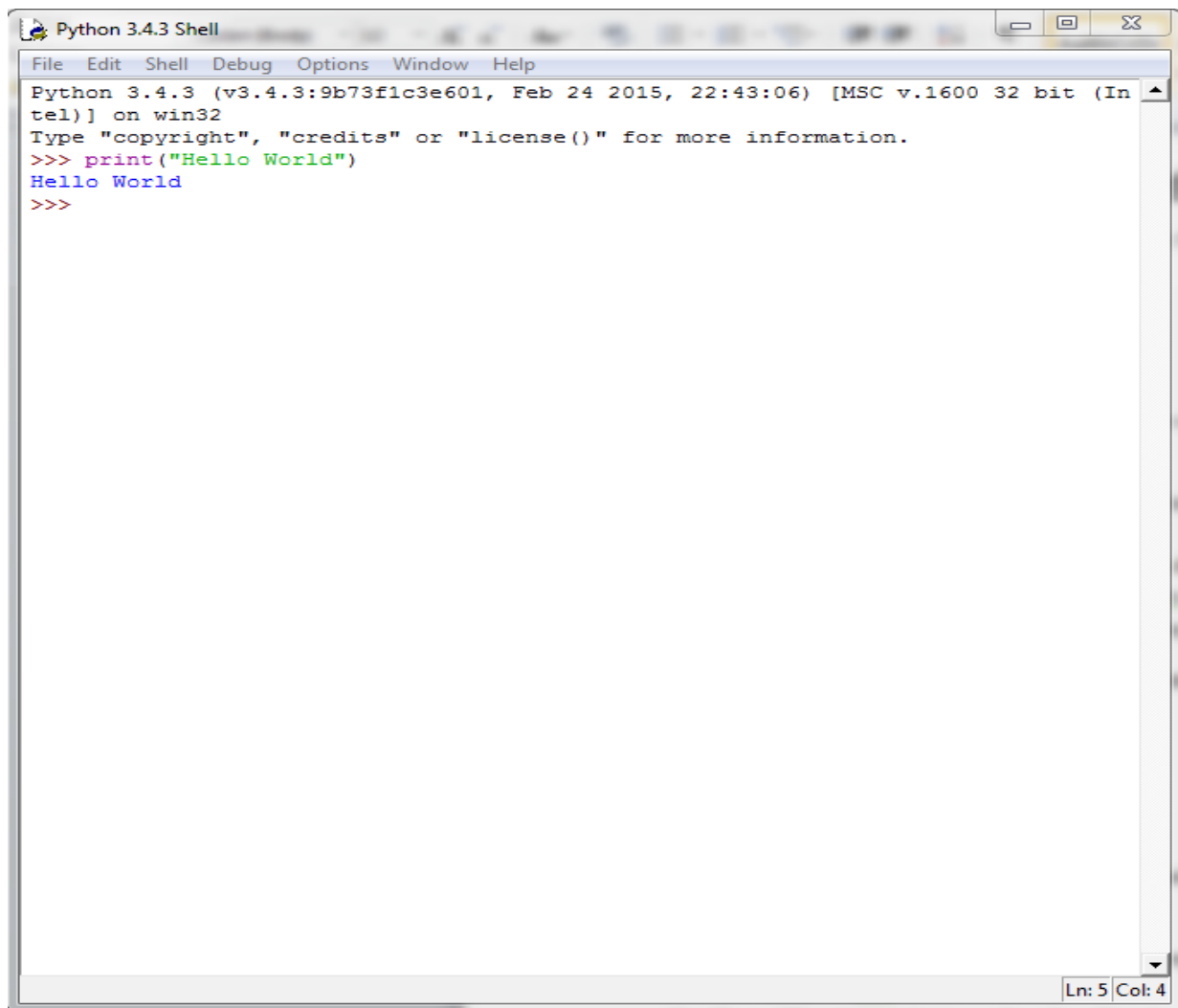
1. `print()` function: `print()` is a built-in function in python which helps in displaying the content to the output screen.
2. Content: It is something that you wish to display on the output screen. In the above program we

have written Hello World ! in double quotes. The double quotes indicate that the content is a string and will display as it is.

**NOTE:** Strings can even be written in single quotes

Example: `print('Hello World ! ')`

Now try to print something of your own.

A screenshot of a Python 3.4.3 Shell window. The window has a menu bar with 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Window', and 'Help'. The main text area shows the following: 'Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32', 'Type "copyright", "credits" or "license()" for more information.', '>>> print("Hello World")', 'Hello World', and '>>>'. The status bar at the bottom right indicates 'Ln: 5 Col: 4'.

```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> print("Hello World")
Hello World
>>>
```

Basic print statement in python console

**Data types:** It specifies the type of data that is being used. We will look on 4 basic data types here:

- *int* (Integer) – These are nothing but simple integers that we use in normal maths like 1,2,3,4.. and so on.
- *Float* (Floating point numbers) – Floating point numbers are the fractions that we deal in maths or simply numbers with decimal point.
- *Strings* – The English language is basically made of letters, words and sentences. But in Python all these things come under one category that is strings. Moreover, numeric values like 1,2,2.5 etc. can also be used as strings when used with double/single quotes.
- *bool* (Boolean values) – This data type can just hold two values and they are either True or False.

---

In programming there are many more data types than the above mentioned. So it becomes difficult to learn them all as you need to explicitly mention the data type in most of the languages .

But Python relieves you from remembering all these data types. HOW??

Well in python you need not explicitly mention data types. In fact python automatically detects the type of data. For example just enter the value 2 and python will automatically understand that it's an integer and allocate only that much memory space required by int data type.

**Operators:** There are 3 types of operators that we will be looking upon:

1. *Arithmetic:* The operators that come under this category are:

- + (Addition) – Will add two or more numbers. Eg 2+3 will give 5
- - (Subtraction) – Will subtract two or more numbers. Eg 5-4 will give 1
- \*(Multiplication) – Will multiply two or more numbers. Eg 2\*5\*5 will give 50
- / (Division) – Will divide and return the quotient. Eg 5/2 will give 2.5
- // (Floor division) – This will divide and return the quotient but it will be rounded off to the nearest lower integer. Eg 5//2 will give 2
- % (Modulus) – This will return the remainder of the division. Eg 5%2 will give 1
- \*\* (Exponent) – We can find the exponent values using this function. Eg 2\*\*3 will give 8

Note: Only by using arithmetic operators we can get some absolute value. The remaining two will only return either True or False and not any definite value.

2. *Conditional:* These are operators that you may have seen in Maths (inequalities). We will just list them and give u some examples because they are self understood:

- > (Greater than) – 5>3 will return True
- < (Less than) – 2<3 will return True
- >= (Greater than or equal to) – 3>=3 will return True
- <= (Less than or equal to) – 5<=3 will return False
- == (Equal to) – 1==1 will return True

Note: Here we use two equal to sign in order to check the equality.

3. *Logical:* We will see two types over here:

- and – Lets consider an example. Suppose if we want to check whether a number is between 0 and 10

and let's assume that the number is stored in variable x. Then in maths we write as follows:

$$0 < x < 10$$

---

---

But in programming we cannot write as above instead we need to write 2 separate conditions and join them using logical operator i.e. :

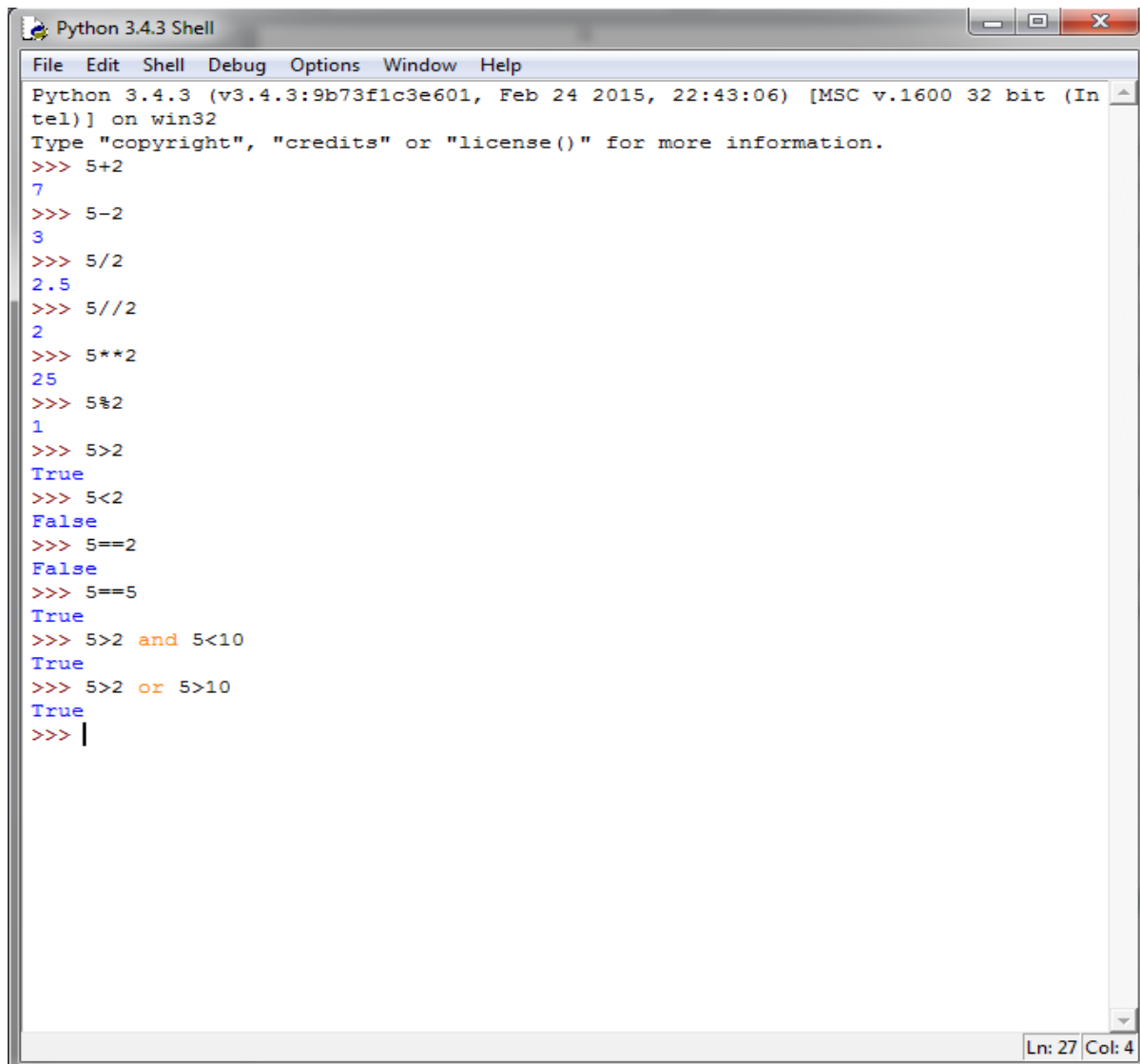
$(x > 0) \text{ and } (x < 10)$

That's it!. A little difference right? If your x is 5 then the above statement will return True.

- Or – Consider another example if we want to check whether a number is either negative or positive and let's assume that the number is stored in variable x. Then you can write it as follows:

$(x > 0) \text{ or } (x < 0)$

If your x is 5 or -5 then the above statement will return True. But if its 0 then it will return False.



```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> 5+2
7
>>> 5-2
3
>>> 5/2
2.5
>>> 5//2
2
>>> 5**2
25
>>> 5%2
1
>>> 5>2
True
>>> 5<2
False
>>> 5==2
False
>>> 5==5
True
>>> 5>2 and 5<10
True
>>> 5>2 or 5>10
True
>>> |
```

Ln: 27 Col: 4

me


---


## Use of various operators

**Variables:** What are variables in standard mathematics?. Can we say that it is a name given to some value which can be manipulated or changed? For eg,  $a=10$  means we are giving the name “a” to value 10, thus “a” is a variable.

Same is the case with python! You simply write  $a=10$  and python defines a memory location, stores value 10, and refers the location as “a”. The name given to the value may be just one character (like a,b,c etc) or it may be a combination of characters i.e a word (like Myvar ).

The following represents the same.

$A=10$  -----means----- 

 is a memory unit storing value 10 and referred using name A

 **codeCELL**

A variable may store value of any data type.

- $a=10$  means “a” stores integer value 10
- $a=10.5$  means “a” stores float value 10.5
- $a=\text{“hello”}$  means “a” stores a string hello.
- $a=\text{True}$  means “a” stores a Boolean value true.

The basic property of a variable is that its value may vary. In other words, the value stored by a variable can be altered. This can be done by simply reinitialising (reassigning) some other value to the same variable. For eg, consider the following code:

```
Myvar=20
```

```
print(Myvar)
```

```
Myvar=30
```

```
print(Myvar)
```

Can you guess the output?? You are right if you guessed the output as

---

---

20

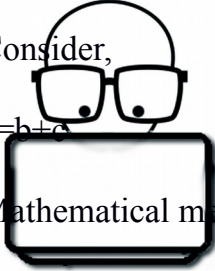
30

How did this happen? As Explained earlier  $\text{Myvar}=20$ , defines a variable "Myvar" and stores value 20 in it which is printed first. Then value 30 is assigned to Myvar which is simply overwritten on the previous value 20. This means the memory location which stored 20, now stores the value 30. Thus 20 is lost and is replaced by 30.

*Note:* The above print statement does not have single/double quotes over Myvar. This is because we want to print the *value* stored in Myvar and *not the text* "Myvar" as it is.

However, the meaning of '=' symbol in programming is not the same as that in standard school mathematics. Mathematics defines '=' as a symbol for *equality*, whereas the same symbol in programming means *assignment*. What's the difference??

Consider,  
 $a=b+c$



# codeCELL

Mathematical meaning: value of "a" is EQUAL to the value of sum of "b" and "c" i.e. LHS = RHS  
Changing the world one bit at a time  
Meaning in Programming: value of  $b+c$  is evaluated and the result is ASSIGNED ( stored ) to "a".

Still not clear?? Have a look at this statement:

$a=a+1$

Is this valid in maths? Well definitely no! The value of "a" can never be equal to "a+1". However, this is very much valid in programming. The above statement would evaluate "a+1" and store the result in "a". If suppose "a" initially stored value 9, then  $a+1$  is evaluated and the result 10 is stored in the same variable "a". Thus, "a" initially stored value 9 which is then replaced (overwritten) by its new value i.e. 10.

We can summarize the significance of "=" in programming as:

- The LHS of "=" defines the memory storage location where the result is to be stored.
-

- 
- The RHS of “=” defines an expression (i.e. combination of variables and operators and constant values) whose value is to be evaluated.

Note: The LHS must always be a single variable and cannot be an expression as it defines a storage unit for storing the result which cannot be generated by evaluating an expression.

**Input from user:** By now, it is very well understood, how to assign a constant value to a variable. But, how can we initialise these variables to a value given by the user? How can the user be prompted to input a value?? The answer to this question will be , using the input() function.

For eg,

```
a=input(" Enter a value ")
```

Executing the above code will print the text “ Enter a value ” and the cursor will wait for the user to provide with an input. The inputted value will then be stored into variable “a” Easy! Isn’t it?

However, it is important to note that the input function considers the inputted value as string by default. Thus, when the cursor waits for an input from user and the user gives value 10, the “10” is interpreted as a string and not as an integer. If we wish to perform any arithmetic operation with this inputted value then it is necessary to convert this string into an integer or float whichever is appropriate. This can be done using type casting.

**Type Casting :** Type casting can be seen as assigning a role to some variable or value. In other words it can be thought of as asking a variable of some data type to act like a variable of another data type.

As mentioned above, we can convert the user input from string datatype into integer or float datatype. This can be done using the following syntax:

```
a=int(input("Enter a integer"))
```

Here, even though user input is treated as a string, it is converted into an integer value before assigning it to variable “a”.

Similarly,

---

---

```
a=float(input("Enter a value"))
```

The above statement converts the input string into float value and then assigns it to variable a.

Going the reverse way,( i.e integer/float to string ) is also achievable using str() function. The following explains the same:

```
Myvalue=10
```

```
a=str(Myvalue)
```

In the above code, Myvalue holds integer value 10, whereas variable a holds string "10"

**Comments:** Comments are the part of the code which are not executed or interpreted. They are simply ignored while running of a program. Why are they used then? They are used just for programmer's reference. In case of huge codes, it may be a good idea to write small descriptions for various statements so that it is easier for the programmer to read and recall the purpose of writing certain lines of the code without actually going through the entire code again.

In python, a single line comment can be made using "#" (hash) symbol. Whatever text that follows the # symbol in the same line is simply ignored while running of the program. For example,

```
# This is a single line comment.
```

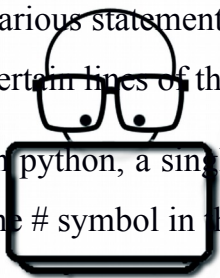
```
print("hello") #This is a print statement.
```

Similarly, a multiline comment can be made using *triple single quotes* i.e

```
'''-----comment line 1-----
```

```
-----comment line 2-----
```

```
-----comment line 3-----'''
```



codeCELL  
Changing the world one bit at a time



---

# Conditional and Looping Statements

## Conditional Statements:

Suppose that sometimes we need to check for a condition and based on the output of the condition we need to perform a set of different tasks. In programming language, we can achieve this using the

**if-else** statement. Syntax:

**If condition :** Changing the world one bit at a time

```
    statement1
```

```
        statement2
```

```
            statement3
```

**else :**

```
    statement1
```

```
        statement2
```

Note:

1. Please remember to use proper **indentation** (one tab or four spaces) while using conditional and looping statements. For example, as you can see in the
-

---

above block of code that all the statement in the **if** as well as the **else** block have an indentation.

2. The **condition** can be specified with or without braces as per one's wish. 3. Do not forget to use the colon ":"

Example:

**Write a program to check whether a number is even or not.**

```
x=int(input("Enter a number:"))  
if(x%2):  
    print("the number is even")  
else:  
    print("the number is odd")
```

Note:

# codeCELL

Changing the world one bit at a time

1. Do not simply copy paste the code from here in Python as some characters (like " and ') might not get identified properly.
2. You can have just an **if** block without an **else** block but not vice versa.

Now, suppose if we had to check various conditions and not just one. We understand this using an example of

## if-elif-else statement

Syntax:

**if** condition1 :

statement1

statement2

---

---

statement3

**elif** condition2 :

statement1

statement2

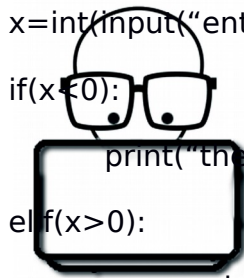
**else :**

statement1

Note: elif is the keyword for “elseif”

Example:

**Write a program to check whether a number is positive negative or zero.**



```
x=int(input("enter a number:"))
if(x<0):
    print("the number is negative")
elif(x>0):
    print("the number is positive")
else:
    print("the number is zero")
```

Note: You can have any number of **elif** blocks as per your requirement.

## • Looping Statements:

Now suppose that we want to execute a set of instructions until a condition is met or until a condition holds true then we make use of looping statements.

There are **two** looping statements as explained below;

### • for loop

---

---

Statements inside the for loop are repeated for a number of times depending on the counter.

We understand **for** loop with the following examples.

**Example 1:**

**Write a program to print first 'n' whole numbers.**

```
x=int(input("how many numbers to print? "))
for i in range(1,x+1):
    print(i)
```

In the above example,

1. "i" is a counter whose value is initiated as the first value(x) in the "range(x,y)".
2. On writing "range(x)", the default value of the counter will be taken as zero and "x" will be the upper limit; it is not included.
3. By default the counter is incremented by 1.
4. In "range(1,x+1)" the value of "i" goes from 1 to x, (x+1) is the upper limit; it is not included.
5. To increment the counter by 2 the change will be "range(1,x+1,2)". The output would then be 1,3,5.....

 **codeCELL**  
Changing the world one bit at a time

Note: Do not forget to use colon ":"

**Example 2:**

**Write a program to print the characters of a string input by user.**

```
x=input("enter a string")
for i in s:
    print(i)
```

In the above example,

1. Suppose if the input string is "codecell".  
Then the index goes on from 0 to 7.
  2. The value of counter "i" is initialized as zero by default.
  3. The statement "for i in s:" checks whether the index "i" is present in the string, if yes then the next line "print(i)" print the character at index "i".
  4. By default the value of i is incremented by 1.
-

- 
5. In the above example, after printing the character at index 7 the value of `i` is incremented by 1 so it becomes 8. Now since there is no index 8 in the input string the for loop ends.

## while loop

Statements inside the while loop are repeated until the while condition holds true.

We understand **while** loop with the following example.

### Example:

Write a program to input a number and print it if it is even. If an odd number is input then don't print it and ask for next number. Stop the program if -111 is input.

```
while(1):
    x=int(input("Enter a number: "))
    if(x== -111):
        break
    else:
        if(x%2==0):
            print(x)
        else:
            continue
print("end of program")
```

in the above example,

1. The syntax is "while(condition):". The use of brackets around the condition is as per one's wish.
  2. "while(1)" means repeat forever as "1" means true and "0" means false.
  3. "break" is a keyword used to break the currently running looping statement.
-

---

Here since we have just the one while loop, the execution will come out of the loop and print “end of program”.

4. “continue” is a keyword used when nothing needs to be done but just move further in the loop. So the execution goes to the start of the loop.
5. In the above example even if you do not write the “else” block it will work the same as we are just using the “continue” keyword in it.

DAY 2 – 2<sup>nd</sup> September 2015

## **Functions**

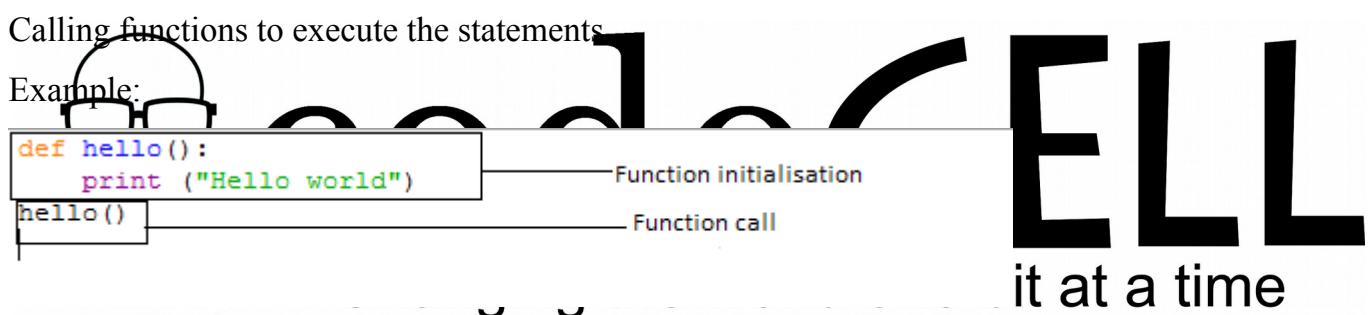
Consider executing a block of statements again and again in the program. In functions we can assign a block of statements a name and call it any number of times we want.

Writing functions :

Function initialisation: declaring the statements in the function and give it name and parameters

Calling functions to execute the statements

Example:



```
def hello():  
    print ("Hello world")  
hello()
```

ELL  
it at a time

Output :

```
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (tel)] on win32  
Type "copyright", "credits" or "license()" for more information.  
>>> ===== RESTART =====  
>>>  
Hello world  
>>> |
```

Here we have created a sample function named hello. We write statements after giving a colon and then indenting statements that follow.

To execute a function or call a function, we simply say <functionname>(<parameters>).e.g.hello()

*Note:*

1.def is colored in orange..its a keyword

---

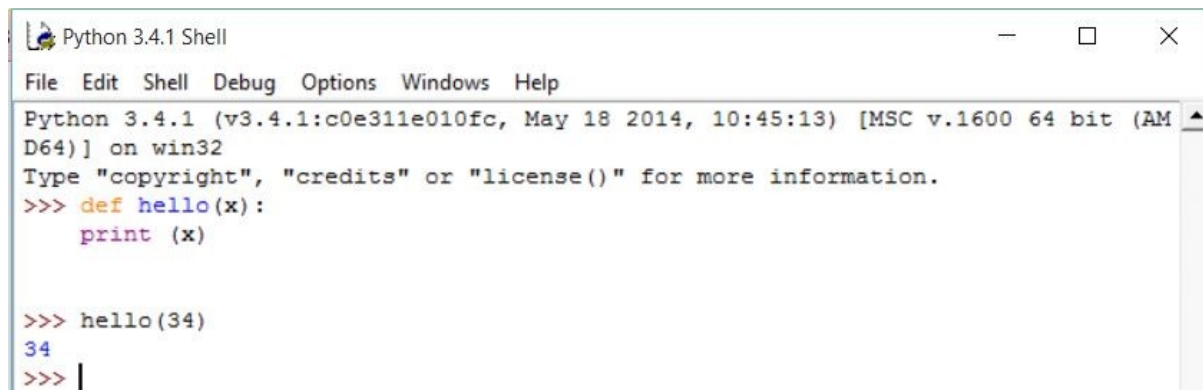
---

2. After colon, we have to indent statements that are to be executed

In addition to this we can also specify arguments in the function, these arguments can be used as values in statements of function.

e.g.

```
def hello(x):  
    print (x)  
hello(34)
```

A screenshot of a Python 3.4.1 Shell window. The window has a title bar "Python 3.4.1 Shell" and a menu bar with "File", "Edit", "Shell", "Debug", "Options", "Windows", and "Help". The main text area shows the following code:

```
Python 3.4.1 (v3.4.1:c0e311e010fc, May 18 2014, 10:45:13) [MSC v.1600 64 bit (AMD64)] on win32  
Type "copyright", "credits" or "license()" for more information.  
>>> def hello(x):  
    print (x)  
  
>>> hello(34)  
34  
>>> |
```

Output:

Changing the world one bit at a time

```
>>>34
```

Add two numbers?

```
def hello(x,y):  
    print(x+y)  
hello(1,2)
```

```
Python 3.4.1 Shell
File Edit Shell Debug Options Windows Help
Python 3.4.1 (v3.4.1:c0e311e010fc, May 18 2014, 10:45:13) [MSC v.1600 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> def hello(x,y):
    print(x+y)

>>> hello(1,2)
3
```

output:

```
>>>3
```

Now, if I want to call a function and want it to return a value, we use the keyword return.

e.g.

```
def hello():
```

```
    return 7;
x=hello()
print(x)
```

```
Python 3.4.1 Shell
File Edit Shell Debug Options Windows Help
Python 3.4.1 (v3.4.1:c0e311e010fc, May 18 2014, 10:45:13) [MSC v.1600 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> def hello():
    return 7;

>>> hello()
7
```

output:

```
>>>7
```

Can we do better? Can we use them in loops? Yes!

```
def hello(x):
```

```
    for i in range(1,x+1):
        print("hello world")
```



hello(5)

output:

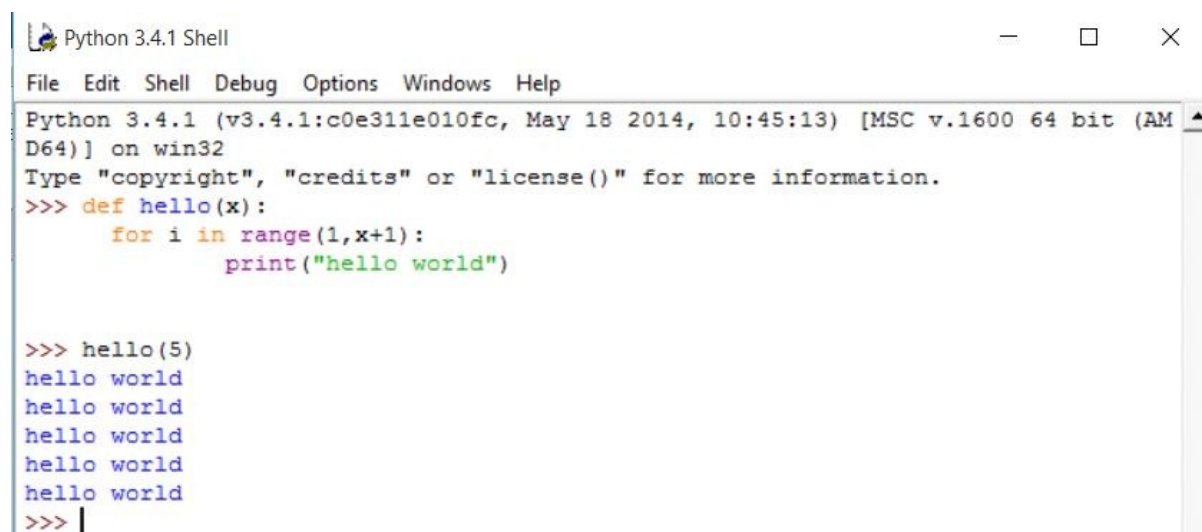
```
>>>hello world
```

```
hello world
```

```
hello world
```

```
hello world
```

```
hello world
```

A screenshot of a Python 3.4.1 Shell window. The window title is "Python 3.4.1 Shell". The menu bar includes "File", "Edit", "Shell", "Debug", "Options", "Windows", and "Help". The status bar shows "Python 3.4.1 (v3.4.1:c0e311e010fc, May 18 2014, 10:45:13) [MSC v.1600 64 bit (AMD64)] on win32". The main text area shows the following code:

```
>>> def hello(x):  
    for i in range(1,x+1):  
        print("hello world")  
  
>>> hello(5)  
hello world  
hello world  
hello world  
hello world  
hello world  
>>> |
```

Note: take care of indentation, if you don't intend properly statements won't execute as you expect.

Better applications:

```
def hello(x):  
    y=0  
    for i in range(1,x+1):  
        y+=i  
    return y  
x=hello(3)
```

Changing the world one bit at a time

```
Python 3.4.1 Shell
File Edit Shell Debug Options Windows Help
Python 3.4.1 (v3.4.1:c0e311e010fc, May 18 2014, 10:45:13) [MSC v.1600 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> def hello(x):
    y=0
    for i in range(1,x+1):
        y+=i
    return y
>>> hello(3)
6
print(x)
```

output:

```
>>>6
```

What do you think the above function does?

Can u predict hello(4),hello(5)?? (Easy.)

What about hello(345678)?(try Python :)



# codeCELL

Changing the world one bit at a time

*Sample exercises:*

- 1.Add all numbers within a range say 4 to 78 using functions.(Cake walk)
- 2.Try printing factorial of a number. (good)
- 3.Write a function to check whether a given number is prime. (Awesome! )

## ***Lists***

Python provides us a structure to store variables at an index.

Consider I want to save my data as

1-“smit”,2-“amit”,3-“rohit” and so on.

We can store all of them in a list.

---

How?

listed=["smit","amit","rohit"].

To access a value in list, simply say <listname>[index]

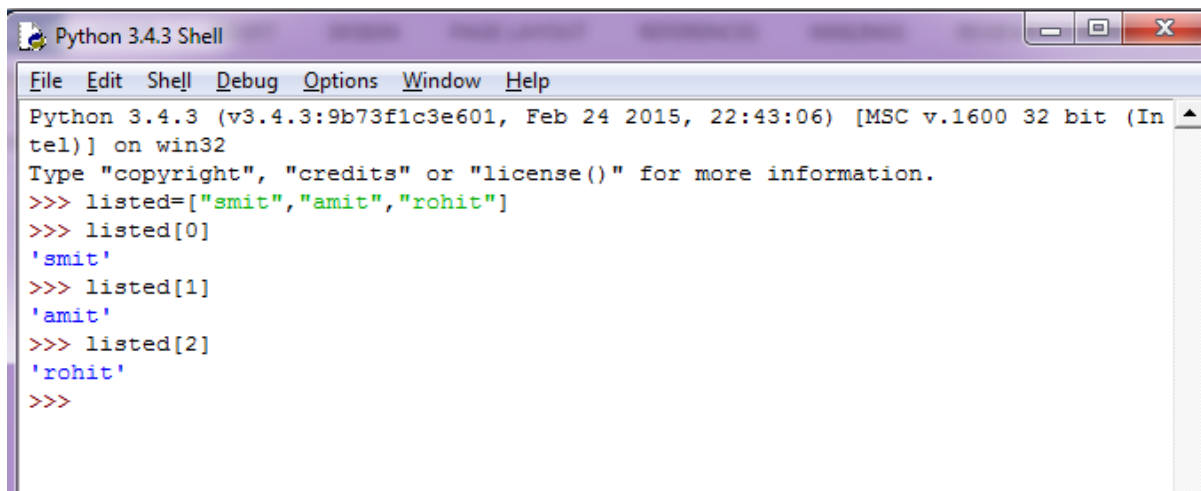
## *Indexing*

In lists the start index is 0.

Thus the data would be stored as

listed=["smit","amit","rohit"]

index ->    0            1            2

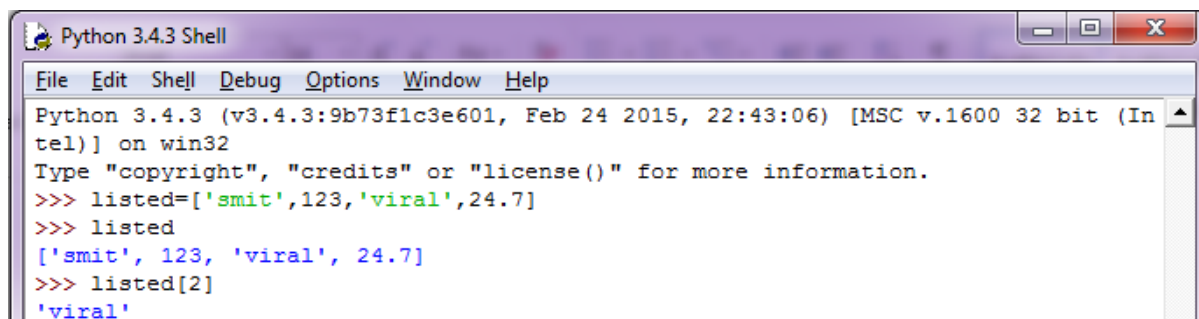
A screenshot of a Python 3.4.3 Shell window. The window has a menu bar with 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Window', and 'Help'. The main text area shows the following code and output:

```
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> listed=["smit","amit","rohit"]
>>> listed[0]
'smit'
>>> listed[1]
'amit'
>>> listed[2]
'rohit'
>>>
```

me

Thus,

A list can even store values of different data types.

A screenshot of a Python 3.4.3 Shell window. The window has a menu bar with 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Window', and 'Help'. The main text area shows the following code and output:

```
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> listed=['smit',123,'viral',24.7]
>>> listed
['smit', 123, 'viral', 24.7]
>>> listed[2]
'viral'
```

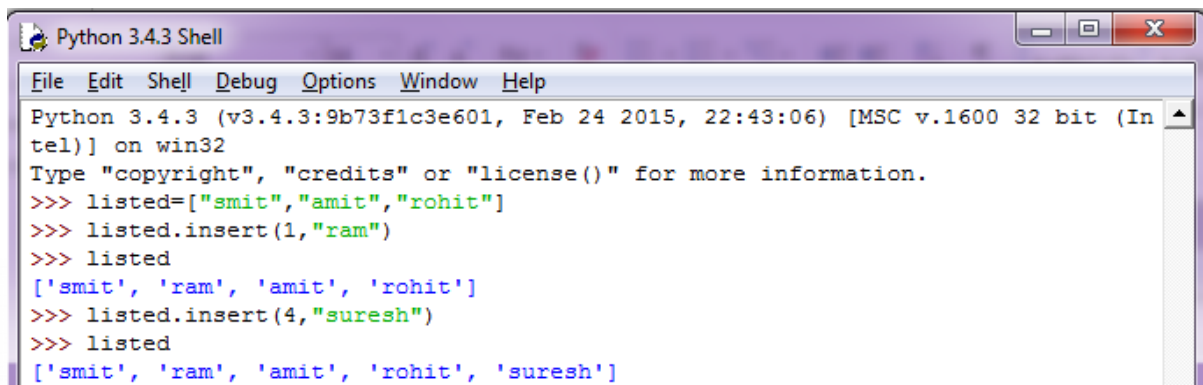
---

## List operations:

### 1.Insert

We can insert items in a list at any position in the list.

The general syntax is `<listname>.insert(<index>,<data>)`

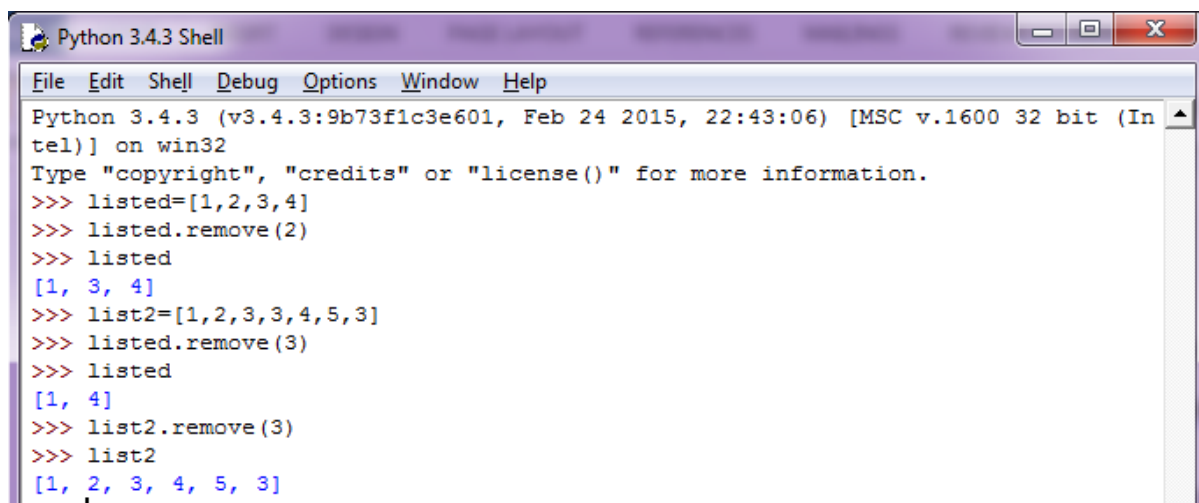
A screenshot of a Python 3.4.3 Shell window. The window has a menu bar with 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Window', and 'Help'. The text inside shows the following commands and output:

```
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> listed=["smit","amit","rohit"]
>>> listed.insert(1,"ram")
>>> listed
['smit', 'ram', 'amit', 'rohit']
>>> listed.insert(4,"suresh")
>>> listed
['smit', 'ram', 'amit', 'rohit', 'suresh']
```

### 2.Remove

We can remove data from a list. However, it removes the first occurrence of the data in the list.

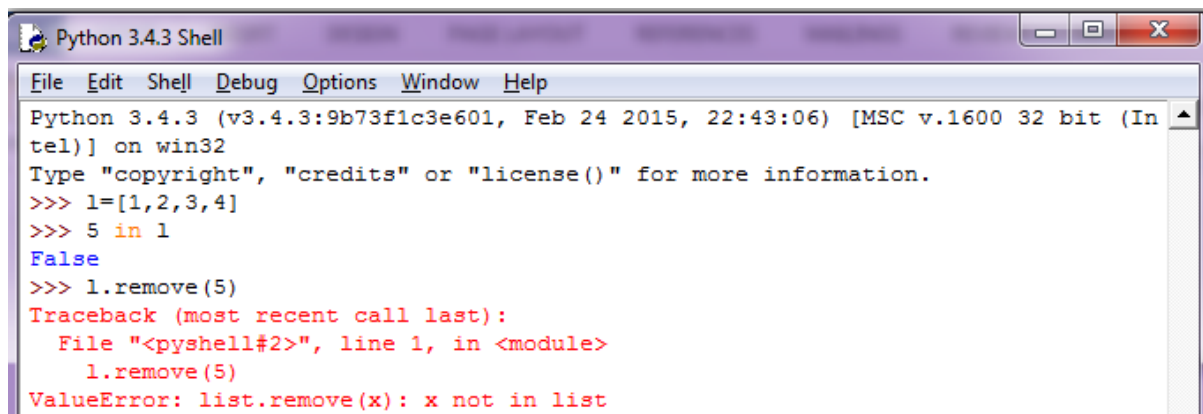
The general syntax is `<listname>.remove(<data>)`

A screenshot of a Python 3.4.3 Shell window. The window has a menu bar with 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Window', and 'Help'. The text inside shows the following commands and output:

```
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> listed=[1,2,3,4]
>>> listed.remove(2)
>>> listed
[1, 3, 4]
>>> list2=[1,2,3,3,4,5,3]
>>> listed.remove(3)
>>> listed
[1, 4]
>>> list2.remove(3)
>>> list2
[1, 2, 3, 4, 5, 3]
```

Changing the world one bit at a time

If data is not found then it gives an error. To check if a value is present in the list we say `<data> in <listname>`.

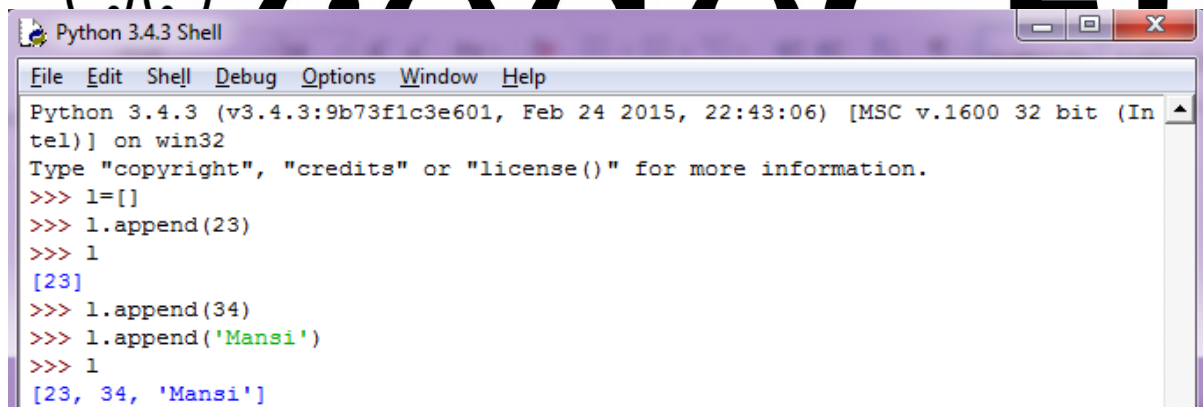
A screenshot of a Python 3.4.3 Shell window. The window has a menu bar with 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Window', and 'Help'. The text inside shows the following commands and output:

```
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> l=[1,2,3,4]
>>> 5 in l
False
>>> l.remove(5)
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    l.remove(5)
ValueError: list.remove(x): x not in list
```

### 3. Append

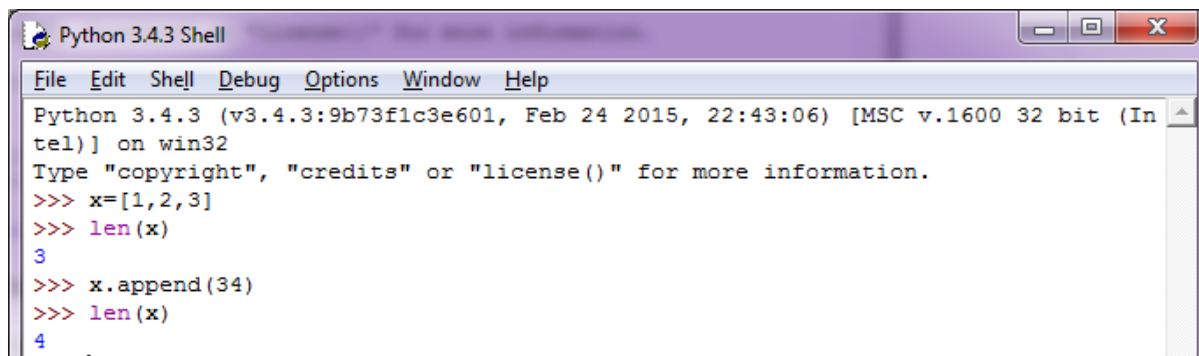
To append means to add in the end. In lists we can append data at the end of the list.

General syntax `<listname>.append(<data>)`

A screenshot of a Python 3.4.3 Shell window. The window has a menu bar with 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Window', and 'Help'. The text inside shows the following commands and output:

```
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> l=[]
>>> l.append(23)
>>> l
[23]
>>> l.append(34)
>>> l.append('Mansi')
>>> l
[23, 34, 'Mansi']
```

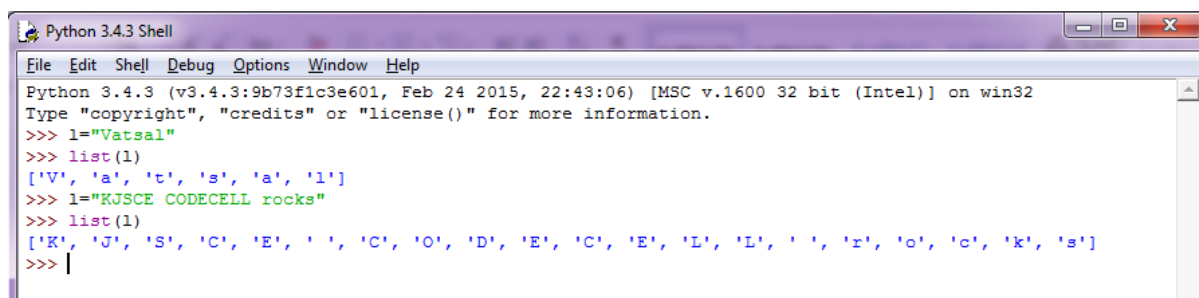
Now suppose we want to find the length of a list, we can do that by `len(<listname>)`



```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> x=[1,2,3]
>>> len(x)
3
>>> x.append(34)
>>> len(x)
4
```

Suppose we want to list a string

We can use `list(<string name>)`



```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> l="Vatsal"
>>> list(l)
['V', 'a', 't', 's', 'a', 'l']
>>> l="KJSCE CODECELL rocks"
>>> list(l)
['K', 'J', 'S', 'C', 'E', ' ', 'C', 'O', 'D', 'E', 'C', 'E', 'L', 'L', ' ', 'r', 'o', 'c', 'k', 's']
>>> |
```

Note: even ' ' or space is also an element.

me

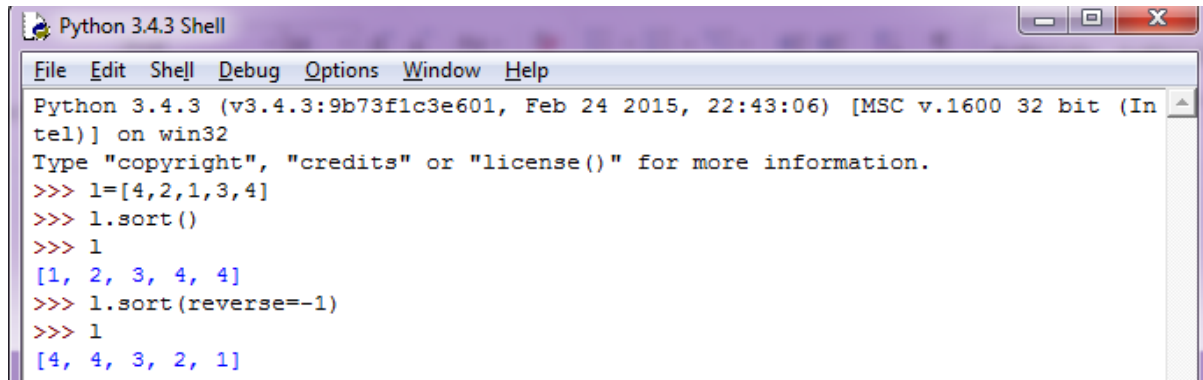
---

Apart from basic operations we have inbuilt features to manipulate lists

Sort

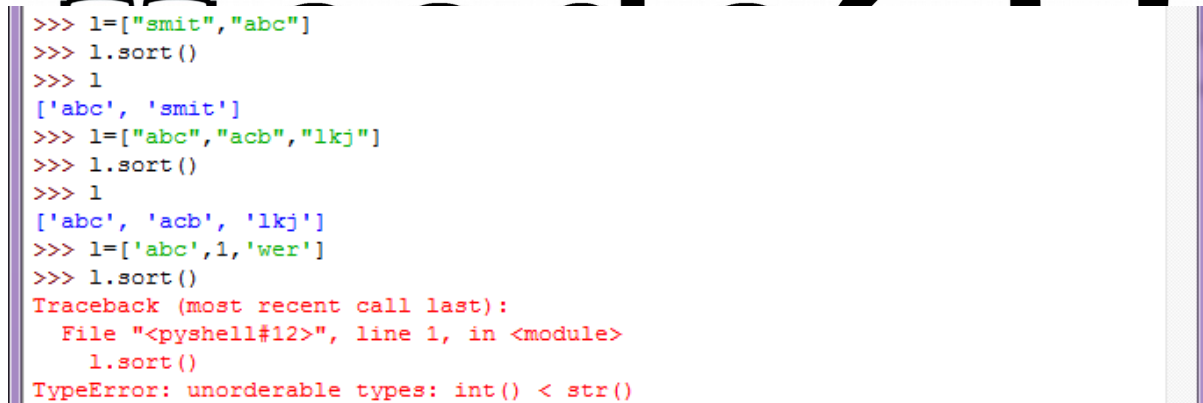
Ascending : <listname>.sort()

Descending : <listname>.sort(reverse=-1)



```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> l=[4,2,1,3,4]
>>> l.sort()
>>> l
[1, 2, 3, 4, 4]
>>> l.sort(reverse=-1)
>>> l
[4, 4, 3, 2, 1]
...
```

List of strings can be sorted. However, lists of strings and numbers cannot be sorted.



```
>>> l=["smit","abc"]
>>> l.sort()
>>> l
['abc', 'smit']
>>> l=["abc","acb","lkj"]
>>> l.sort()
>>> l
['abc', 'acb', 'lkj']
>>> l=['abc',1,'wer']
>>> l.sort()
Traceback (most recent call last):
  File "<pyshell#12>", line 1, in <module>
    l.sort()
TypeError: unorderable types: int() < str()
```

## **Strings**

---

---

String is a continuous sequence of terms(character, symbol, digits) it can be a single character or a sentence or a paragraph strings are denoted by using single(')or double (") quotes.

*Example :*

a]"Hello world"

b}"s"

c}"dsfdsf545664566#%#@#%"

d}'i love coding it is great'

*Note:*

1]Even a blank space (" ")is considered as string character

2}Python does not support a character type: these are treated as strings of length one



*Declaring a string*

s="hi"

print(s)

*Output:*

hi

There are various string functions

1]String length

2]Character from a string

3]Concatenation of 2 or more strings

4]Substring

---



---

Lets get more technical

1]String length

*Syntax:* len(string name)

This gives the number of characters present in the string

for example:

1]

```
x="codecell"
```

```
y=len(x)
```

```
print(y)
```

*Output:*

8

Thus, the string 'codecell' has 8 letters



codeCELL

Changing the world one bit at a time

2]

```
x=len("hi python")
```

```
print(x)
```

*Output:*

9

*Note:*

Indexing:

---

---

In any programming language start counting from 0.

for example,

0 1 2 3 4 5 6 7

c o d e c e l l

Which implies character 'c' is stored at 0 position, and character 'o' at 1 position and so on.

## 2]Character from a string

Suppose I want to take out character 'd' from my above string example. For this, first I need to check the index of d i.e 2



```
s="codecell"
```

```
x=s[2]
```

```
print(x)
```

*Output:*

“d”

## 3]Concatenation of strings

We can combine 2 or more strings

---

---

Suppose a and b are 2 strings then a+b is their concatenation

For example,

```
a="hello"
```

```
b="world"
```

```
print(a+b)
```

*Output:*

“helloworld”

4]Substring

What if we want only part of the string i.e suppose i want only cell from codecell

*Syntax:*

```
substring=stringname[starting position:ending position +1]
```

*Note:*

In end if [2:4] then it takes out till 4 position the last char is not considered



codeCELL

Changing the world one bit at a time

For example,

```
x="codecell"
```

```
y=x[4:8]
```

```
print(y)
```

*Output:*

“cell”

There are two more types of substrings:

```
a]
```

---

---

```
y=x[2: ]
```

```
print(y)
```

Here from position 2 to end of string is considered

*Output*

“ecell”

b]

```
y=[ :5]
```

```
print(y)
```

Here from start of string to n-1 position is considered

*Output :*



***Introduction to CodeChef***

---

---

---

So by now, you all know that we are CodeCell, the KJSCE Campus Chapter of CodeChef. What really is CodeChef?

CodeChef comes under a category what we call as an “Online Judge”. An online judge is a portal to test programs in competitive programming. Too many buzzwords! Let’s break it down. You are provided with a problem statement, similar to word problems we used to solve in maths in school. But these test your programming concepts, logic, data structures and algorithmic skills. You have to devise a program to solve this given problem keeping in mind the mentioned constraints. Along with the problem, you are provided with a sample input and output as an example to help you understand the problem and also, to check whether your program yields the same. Now, once you submit your code to this online judge, it will compile and run your code over a particular number of test cases. These test cases are hidden from the viewers. If your program fails to give a correct output to any ONE of these, your program will be marked as wrong answer. It is very important to optimize your code in terms of time and space complexity.

What do you learn from solving these questions? Different programming methods and approaches like recursion, divide and conquer, greedy methods, dynamic programming, graph theory, string matching algorithms, Mathematical concepts like combination with repetition, zeckendorf's theorem, probabilistic analysis, amortized analysis and much more!

Now, imagine there are thousands of participants solving the same problem as you in a given time limit. That makes it competitive. You have to provide the best solution and that too, fast! To make it a little more interesting and rewarding, CodeChef organizes monthly contests and distributes prizes to top performers. Apart from that, there are plenty of competitions taking place. You can think of competitive programming as a sport. We have our own Olympics and Grand Slam. ACM ICPC is considered as the Olympics of competitive programming and is the most prestigious contest hosted annually. Apart from that, there are IOI (International Olympiad in Informatics), Google Code Jam, Facebook Hacker Cup, Top Coder Open and many many more.

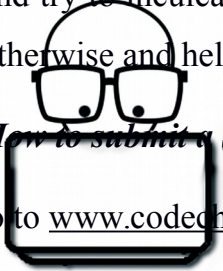
So what is the point of all this now? Most engineers go either for placements after their B. Tech or go for MS. Let's understand how it helps in either:

---

- 
3. *Placements*: Why do big shot companies like Google, Facebook, Microsoft, Amazon, Flipkart, Morgan Stanley pick students from IITs, NITS, IIITs? We both learn the same syllabus, almost. The reason is problem solving – These companies don't want people who can write in a 1000 languages (Although that would be lauded) but problem solvers. People who know how to find optimal solutions which take the least time. This is where competitive programming comes in. It teaches you a lot of these skills and helps you actually apply all you've learn in realtime.
4. *MS*: Going to the National Finals of the ICPC means you're one of the top 500 computer scientists in the country and going to the International Finals means you're one of the top 500 computer scientists in the world. Top colleges like MIT, Stanford want students who are the best at what they do. Need I say more?

So the main aim of CodeCell is not only to create awareness about the importance of programming and try to inculcate the passion in students but also to help them learn concepts they wouldn't learn otherwise and help them reach the level of world-class programmers.

*How to submit a solution on CodeChef.com*

- 
- # codeCELL
- Changing the world one bit at a time
- Go to [www.codechef.com](http://www.codechef.com)
  - Make an Account or Login
  - Go to Practice and select a problem to solve. For this example we select the problem “Life, The Universe and Everything” or Problem Code: TEST.
  - Read the problem statement alongside the input/output constraints and format and formulate a solution.
-

---

## Life, the Universe, and Everything ✓

Problem code: TEST

91 people like this.

---

All submissions for this problem are available.

For help on this problem, please check out our tutorial [Input and Output \(I/O\)](#)

Your program is to use the brute-force approach in order to find the Answer to Life, the Universe, and Everything. More precisely... rewrite small numbers from input to output. Stop processing input after reading in the number 42. All numbers at input are integers of one or two digits.

**Example**

**Input:**

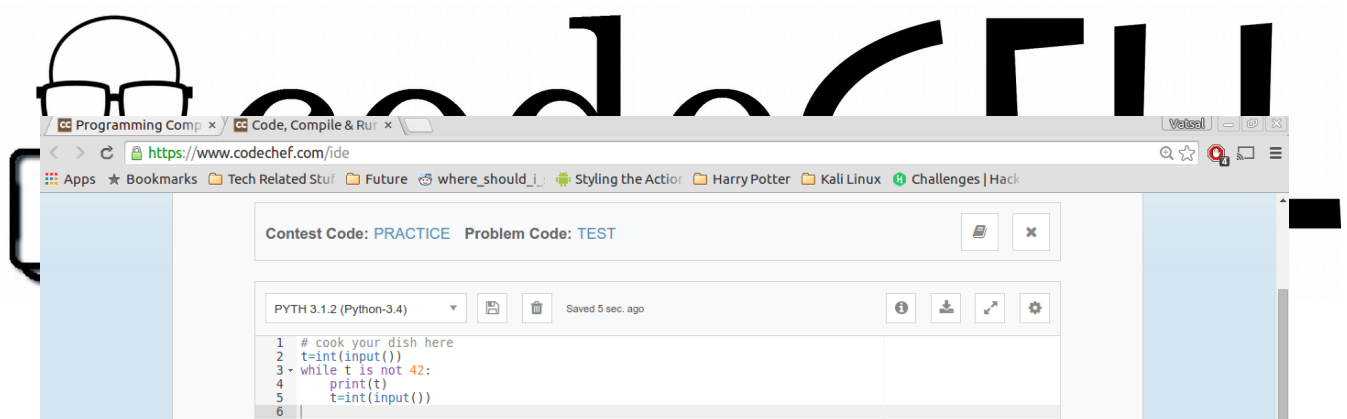
```
1
2
88
42
99
```

**Output:**

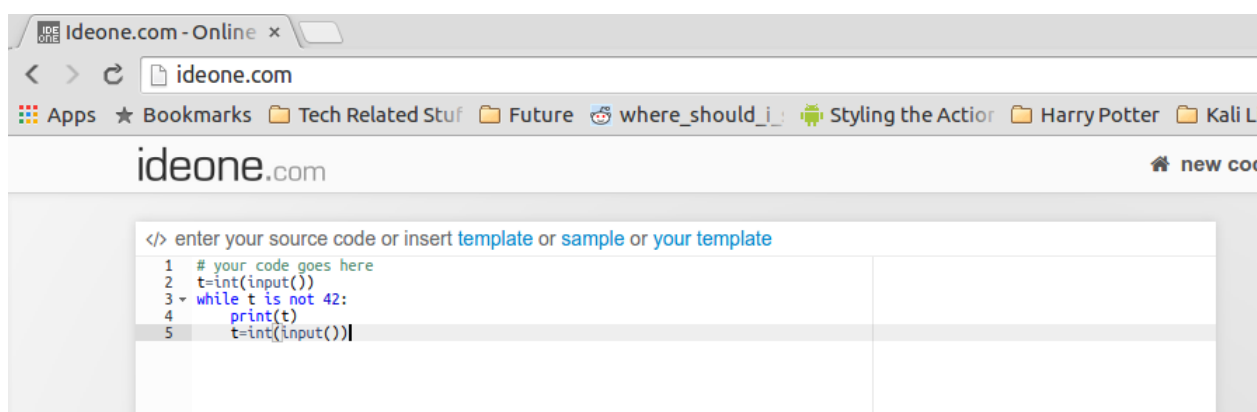
```
1
2
88
```

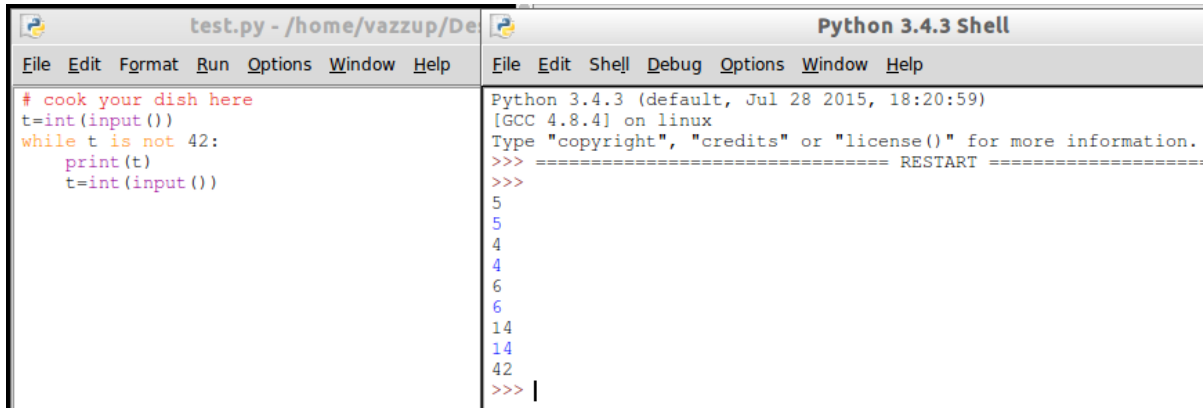
*Life, The Universe  
and Everything*

- Convert the solution into a code format into a language of your choice in a local ide, or online ide like ideone.com or “Code, Compile and Run” ([www.codechef.com/ide](http://www.codechef.com/ide)) Select your language and write down your code. For this case, we select python 3 as our language



*Code, Compile and Run*





The screenshot shows the Ideone.com interface. On the left, a file named 'test.py' is open, containing the following Python code:

```
# cook your dish here
t=int(input())
while t is not 42:
    print(t)
    t=int(input())
```

On the right, the 'Python 3.4.3 Shell' is open, showing the output of the code:

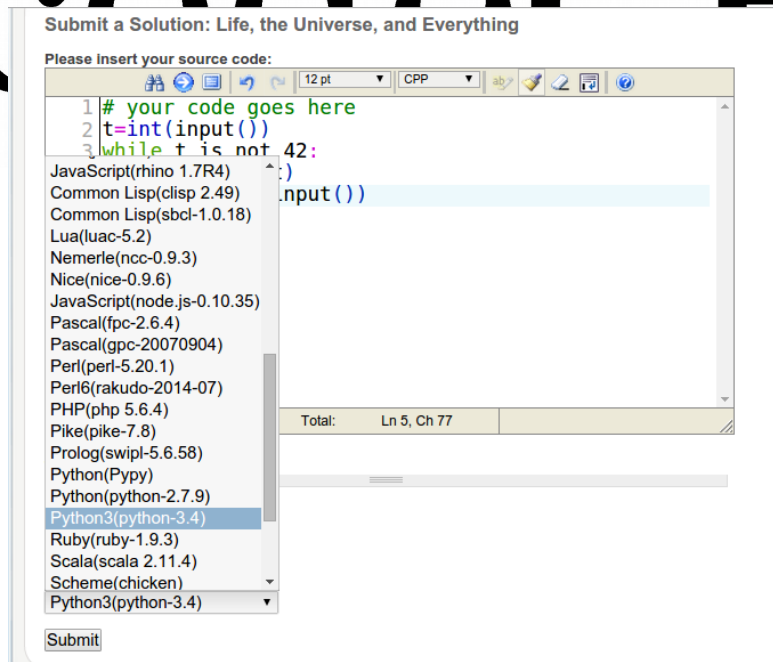
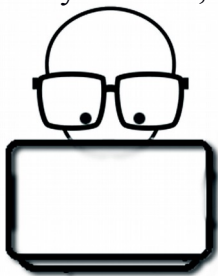
```
Python 3.4.3 (default, Jul 28 2015, 18:20:59)
[GCC 4.8.4] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
5
5
4
4
6
6
14
14
42
>>> |
```

Local Ide  
(Idle 3)

- Check with multiple input test cases. If we

get expected output for all testcases, we submit the problem.

- To submit, click the submit button on the top right of the question's page. In the editor that opens, paste your code, choose your language and press submit.



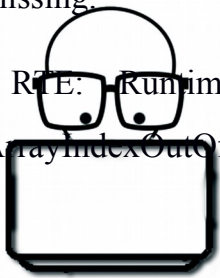
at a time



- 
- Congratulations! If it's correct, you've just solved your first problem on CodeChef.com!

✓  
**Correct Answer**  
**Execution Time: 0.01**

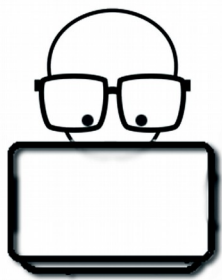
- There are not only correct answers but also:
  - WA: Wrong Answers
  - TLE: Time Limited Exceeded. Your Program is too slow
  - CLE: Compile Time Error. There is some fundamental error in your code. Example: Semi Colon missing.
  - RTE: Runtime Error. Error encountered when the program is running. Example: ArrayIndexOutOfBoundsException.



codeCELL

Changing the world one bit at a time

---



# codeCELL

Changing the world one bit at a time

---

---