**Pyton**

I have been quite eager to learn this language .. so i thought why not LEARN and SHARE simultaneously  
  
So all the stuff that you'll see below is not Written completely by me.. I've taken reference from Google  
  
So let's Start from Basic.  
  
**What is Python?**  
  
Wiki says --- "**Python** is a widely used general-purpose, high-level **programming language**. Its design philosophy emphasizes code readability, and its syntax allows programmers to express concepts in fewer lines of code than would be possible in **languages** such as C++ or Java."  
  
**What is Python Used for?**  
  
StackOverflow says -- "**Python** can be **used** for any programming task, from GUI programming to web programming with everything else in between. It's quite efficient, as much of its activity is done at the C level. **Python** is just a layer on top of C."  
  
**Why Should We learn Python only?**  
  
SkilledUp notes down --  
  
1. Easy to learn  
2. Stepping Stone Python can be your stepping stone into the programming universe  
3. Get a Raspberry with Pi done -- Making programming fun is no easy task, whether your students are beginners, seasoned veterans, or children. Heck, even the most enthusiastic teachers struggle to get anyone excited about writing code. But miracles do happen once in a while: bridging the gap between abstract computing and real world applications is now*as easy as Pi*.  
4. Works online too    

**From where to download Python?**

**If you're running Windows**: the most stable Windows downloads are available from the [Python for Windows](https://www.python.org/downloads/windows/) page.

**If you're running Windows XP**: a complete guide to installing [ActivePython](https://wiki.python.org/moin/ActivePython) is at [Python on XP: 7 Minutes To "Hello World!"](http://www.richarddooling.com/index.php/2006/03/14/python-on-xp-7-minutes-to-hello-world/). [ShowMeDo](https://wiki.python.org/moin/ShowMeDo) has [two videos](http://showmedo.com/videos/series?name=pythonOzsvaldPyNewbieSeries) for downloading, installing and getting started with Python on a Windows XP machine - this series talks you through the Python, [ActivePython](https://wiki.python.org/moin/ActivePython) and [SciPy](https://wiki.python.org/moin/SciPy) distributions.

**If you are using a Mac**, see the [Python for Mac OS X](https://www.python.org/downloads/mac-osx/) page. MacOS 10.2 (Jaguar), 10.3 (Panther), 10.4 (Tiger) and 10.5 (Leopard) already include various versions of Python.

**For Red Hat**, install the python2 and python2-devel packages.

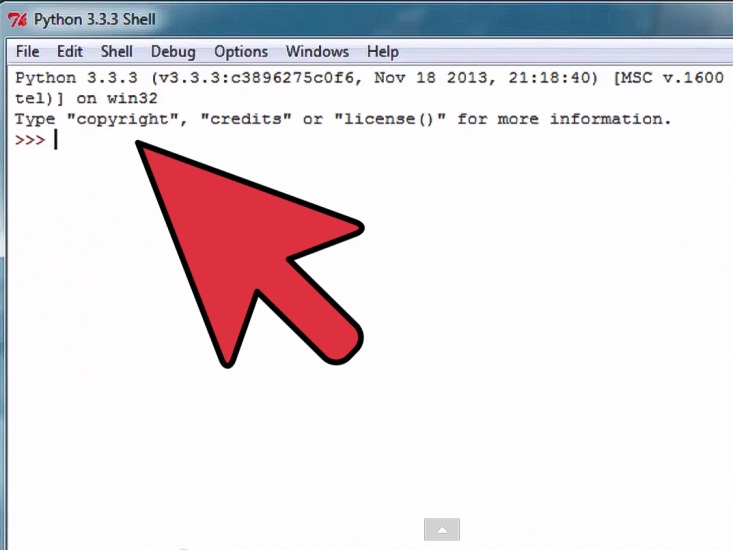
**For Debian or Ubuntu**, install the python2.x and python2.x-dev packages.

**For Gentoo**, install the '=python-2.x\*' ebuild (you may have to unmask it first).

**For other systems**, or **if you want to install from source**, see the [general download page](http://www.python.org/download/).   
  
   
**Using Python. How?**  
  
**First of all install the python from the above links then follow below:**  
  
Install a text editor. While you can create Python programs in Notepad or TextEdit, you will find it much easier to read and write the code using a specialized text editor. There are a variety of free editors to choose from such as Notepad++ (Windows), TextWrangler (Mac), or JEdit (Any system).

**Test your installation.** Open Command Prompt (Windows) of your Terminal (Mac/Linux) and type python. Python will load and the version number will be displayed. You will be taken to the Python interpreter command prompt, shown as >>>.

* Type print("Hello, World!") and press ↵ Enter. You should see the text Hello, World! displayed beneath the Python command line.

   
  
This is how the shell looks like  
  
  
  
   
  
  
  
  
  
  
  
**SYNTAX::**  
  
Important point please read...  
  
Python has no mandatory statement termination characters and blocks are specified by indentation. Indent to begin a block, dedent to end one. Statements that expect an indentation level end in a colon (:). Comments start with the pound (#) sign and are single-line, multi-line strings are used for multi-line comments. Values are assigned (in fact, objects arebound to names) with the \_equals\_ sign ("="), and equality testing is done using two \_equals\_ signs ("=="). You can increment/decrement values using the += and -= operators respectively by the right-hand amount. This works on many datatypes, strings included. You can also use multiple variables on one line. For example:

>>> myvar = 3

>>> myvar += 2

>>> myvar

5

>>> myvar -= 1

>>> myvar

4

"""This is a multiline comment.

The following lines concatenate the two strings."""

>>> mystring = "Hello"

>>> mystring += " world."

>>> print mystring

Hello world.

# This swaps the variables in one line(!).

# It doesn't violate strong typing because values aren't

# actually being assigned, but new objects are bound to

# the old names.

>>> myvar, mystring = mystring, myvar

**NOW google help ends.. Below all stuff is written by me**

**Declaring variables:**

var=5

**var prasad=5 ....** this is not correct

python do not allow spaces to be der in the variable n**ame**

**Math OPERATORS:**

**+ -**

**\* /**

**() % .... used for getting the remainder**

**NOTE: the best operator is \*\* .. known as power operator**

eg. 12\*\*2 = 144

**and 16\*\*0.5= 40**

**Float and INT difference**

eg. int are like 0,4,515

and float are lije 2.3, 6.5454

eg 3/2=1

**and** 3/2.0=1.5

**Overwriting:**

float(3/2) .... -> 1.0

**and**  float(3)/2 -> 1.5

**OTHER FUNCTIONS:**

**1. abs() ... absolute value**

**2. sin() ... sin of**

**3. cos() ... cos of**

**4. floor() ... round down**

**5. ceil() ... round up**

**6. pow() ..... power also known as \*\***

**STRINGS**

**T**hey are bunch of characters. we can directly input strings in python as follows:

eg. x='ham' or x="ham" .... does the same job so go for single or double quotes any one will do

ADDING STRINGS::

eg. y=x +"prasad"

o/p will be y=hamprasad

now if we want to ad a space in between them we can have as follows:

eg. y=x+" prasad"

o/p will be y=ham prasad

**NOTE: STRING AND INTEGER CANNOT BE Merge directly in python so we need to do the following**

e.g z=10

**y=x+str(z)**

**o/p is y="ham10**

**we can use special format specifiers to do the same**

**e.g.** y="same %d" %z

**o/p is y="same 10"**

**NOTE:: %d for integer %f for float and %.3f or any value before an f rounds up the number of decimal**

**numbers after decimal point**

e.g y="same 10.00000"

and then put y="same %.3f" %z ( z=10)

so o/p is y=10.000

**'\n' --> this is new line character adds a new line dat's it**

**Use of print keyword**

e.g. x="te \n pap"

so we write print x

o/p is te

pap

**'\t' --> this is tab character adds a tab dat's it**

**'IN' keywoerd:**

Used to find anything within any other thing

e.g. if you directly write in IDE

"ham" in "hamper"

o.p will be TRUE

**LISTS:**  
  
Data structures used for storing all the data types ...  
  
**eg.**x=['ham',4,2.2]  
  
**Adding to a list:**  
  
**Append keyword:**  
  
**e.g.**x=append(5) ..... here 5 gets added at end of the lists by default  
 **so o/p is**x=['ham',4,2.2,5]  
  
**Adding to a list:**   
 **Insert Keyword**:  
  
syntax: x.insert(licn,value)  
  
**e.g.**x.insert(1,3.14)  
so op is  x=['ham',3.14,4,2.2,5]   ...... **note LISTS  start with 0 location**  
  
**Removing a value from lists:**  
  
**POP keyword:**  
syntax: pop(location)  
  
e.g. x.pop(1)  
so op is .... x=['ham',4,2.2,5]  
  
**LEN keyword:**  
finds the length of items in a string or list  
  
**e.g.**len("words")  
so op is ... 5  
  
**Converting a item to list**  
  
syntax... list(item)  
  
e.g  list("ham")  
op is ['h','a','m']  
    
**TUPLES**   
  
just like Lists but they are not flexible or unadjustable  
  
e.g x=['ham',4]  
  
**TUPLES vs LIST**  
  
1. Tuples take less memory than list  
2. Cannot be adjusted, lists are more adjustable  
  
**DICTIONARIES**  
  
also known as Hash tab,map  
used for binding keys to values  
  
e,g sam={} ... creating a dict  
sam["weapon"]="chainsaw"  
sam["health"]=10  
  
then press enter,, it gets created after this if we write sam we get op as follows:  
  
o.p. {'weapon':'chainsaw','health':10}  
  
**Retrieving from a dict**  
  
**e.g.**sam["weapon"]  
op is 'chainsaw'  
  
**Removing from a dict**  
  
**syntax:**del sam["item"]   
e,g del sam["health"]  
op is {'weapon':'chainsaw'}

Exercises:

A=str(int(2.23)+float(14))+” pap”

Op is …” 16.0 pap “

“ham Ham”.upper()

Op is … ‘ HAM HAM’

“SUPER Baby”.lower()

Op is .. ‘super baby’

b=” I am ham’

b.split()

Op is…[ ‘I’,’am’,’ham’]

b.split(“m”)

Op is.. [‘I a’,’ ha’,’’]

b.join(“p”)

Op is ..[‘I ap’,’ hap’,’p’]

L=[1,2,3,4,5,6,7]

L[:] à [1,2,3,4,5,6,7]

L[ : 2] à [1,2]

L[: : 2] à [1,3,5,7]

L[1: : 2] à [2,4,6]

**Conditonals:**

1.       IF CONDITION

Syntax

If(condition is true):

                Do this

2.       ELSE condition

Syntax

If():

…..

else:

---

3.       Elseif Condition

Syntax

elif():

EXAMPLES::

e.g1 mail=5

if mail:

                print “Mail TIME!”

**enter again**

**OP is .. mail TIME!! ….>  Since mail has some value op is what is written in the print statement**

e.g2 mail=0

if mail:

                print “Mail time”

else:

                print “No mail”

**2 enters here to get the output**

**Op is  …. No mail**

**COMPARISON OPERATORS::**

< less than

<= less than equal to

> greater than

>= greater than equal to

== equal to

!= not equal to

ALWAYS USE PARENTHESIS To write conditions

e.g if (4<6):

print “Pap”

Op à Pap

e.g if(7) and (6):

print “YUP”

**Op .. YUP because there is some value present in the parenthesis**

e.g if (7) or(0)

print  “wahahah”

OP… No output is displayed beacause there is no value in one parenthesis

**LOOOPS: way to repeat the actions**

1.       while (condtition is true ):

     do this

e.g x=0

while (x<0):

x+=1

Op.  x=10

2.       break … Used to stop the loop

syntax …

while (true):

                if(something):

                                break

e.g x,y=0,0

while(True):

                x=x+1

                y=y+2

                if(x+y>10)

                                break

Op is …. X=4 y=4

3.       for loop

e.g. x=[1,2,7]

for i in x:

                print i

OP is … 1

                2

                7

**RANGE a Creates a list of sequential numbers**

e.g for i in range(30):

                print i

Op is…. 0 ,1 ,2 ,3 …… 29

**CONTINUE à Start loop over, skip a value**

e.g.  Print non multiples of 3

for I in range(30):

                if not( i%3):

                                continue

                                print i

OP is  …. All the non multiples of 3 till 29

**EXCEPTIONS:**  
  
Prevents the codes and scripts from breakaways  
**We'll look at 2 new keywords**  
  
**1. try -->**used to test if input is correct or errorfree  
**2. except**--> catches all the errors  
e.g. x=5+'pap'  
**o/p> you will recive an error here**  
  
e.g solved.  
try:  
           x=5+'pap'  
     except:  
           print 'U have an error above'  
op is .... U have an error above  
  
**3. pass**--> says to ignore and move on, mostly used in the except loop shown above.  
  
**e,g,**try:  
            x=9+'pap'  
       except:  
           pass  
op.   no output will occur, so it passes the statement  
  
**4.raise**--> force an error to occur  
  
**e.g.**raise TypeError("PAP")   
in op you will get the traceback error msg PAP passed  
  
**5. finally**--> used to specify which is the last action to perform if any exception occurs  
**e.g**  
try:  
x=5+'pap'  
except ZeroDivisionError:   **--> NOTE here you can specify any specific error you want, if left blank it will catch all the errors**  
            print 'nothing'  
finally:  
      print 'the final statement'  
  
op will be ...... the final statement **and then you'll see the error traceback msg.**  
  
  
**FUNCTIONS:**  
  
**way to combine many actions together**  
  
**syntax:**  
  
**def doesnothing():**  
  
here. def -->declares the function  
doesnothing --> name of the function  
  
**to run the function:**  
  
**run doesnothing()**  
  
**1. return -->**way to output the data from a function  
**e.g.**  
  
def makeone():  
**return 1**  
x=makeone()  
**print x**  
  
**op will be ...**1  
  
**ARGUMENTS:**  
  
**way to input pass the data to a function**  
  
**2 types, 1. regular 2. keyoword**  
  
in the regular arrgument, we have basic input variables  
while in the keyword argument we declare variables with a value within the function.  
  
**e.g.**  
  
def myfn(var1,var2=4)  
**here var1 --> regular argument and**  
**var2 --> is keyword argument**  
  
egs...  
  
1. def add10(myint)  
**mayint+=10**  
**return mayint**  
  
x=12  
**y=add10(x)**  
**print x,y**  
  
**op is...**12,22  
  
**SPECIAL keyword called ... dir() --> gives all the functions and files that are stored in the current directory**  
  
**LOCAL vs GLOBAL variables**  
  
**1. local -->** variables created andstored within a function that will be deleted from memory when a function is completed  
  
**2. global -->**variables that  are accessible anywhere within the program  
  
e.g.  
var=5  
def myfn();  
global var ---> **this declare variable var as global**  
  
**COMMENTS & DOCUMENTS:**  
  
comments and documents are part of documenting your code so it can be easily understood by others.  
  
**doc syntax .... use triple quotes as follows: `"**  
**comment syntax... used # symbol in front of line and it won;t be executed**  
 **e.g.**  
def myfn():  
    `"  
        this is what i documented  
     `"  
#only seen in code window  
pass  
  
print myfn.\_\_doc\_\_      **NOTE there are 2 underscores back to back**  
  
**op is  ...**this is what i documented