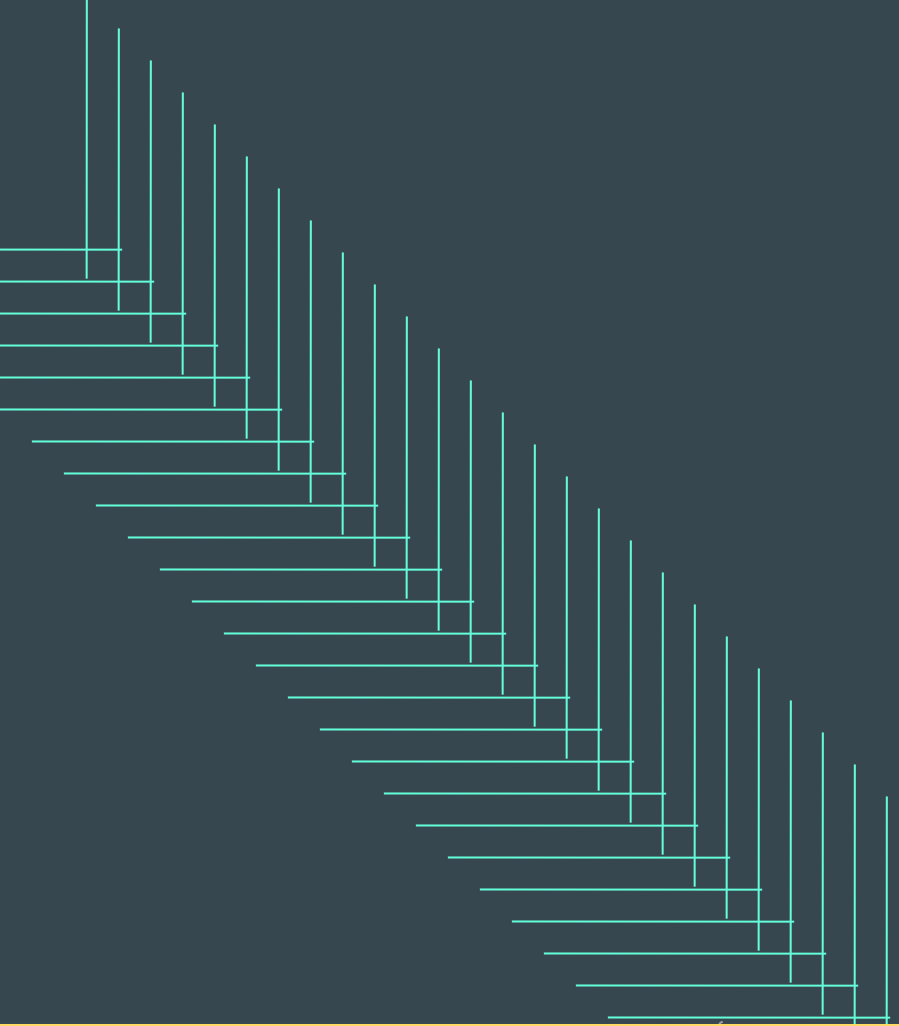


Python

The Complete Course

TELCOMA

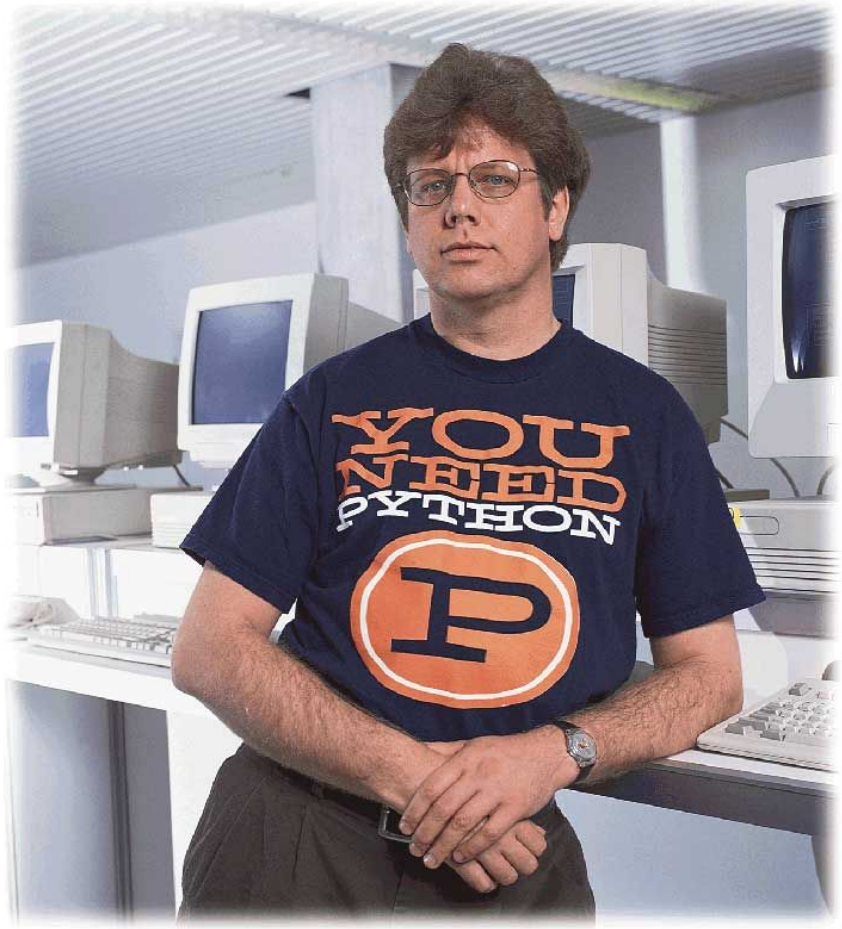
Introduction

What is Python ?

- Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language.
- Python is designed to be highly readable.
- Python is copyrighted. Like Perl, Python source code is also available under the GNU General Public License (GPL).

Python's Creator

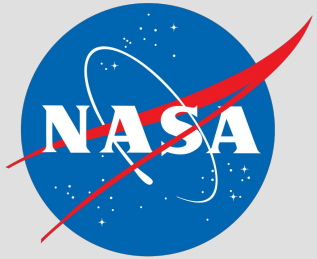
- Guido van Rossum is the creator of Python, one of the world's most popular programming languages
- Python was conceived in the late 1980s, and its implementation began in December 1989



Why named Python?

- Named after Monty Python
- Monty Python were a British comedy group who created their sketch comedy show Monty Python's Flying Circus, which first aired on the BBC in 1969.

Companies using Python



Python Feature

- **Easy-to-learn:** Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.
- **Easy-to-read:** Python code is more clearly defined and visible to the eyes.
- **Easy-to-maintain:** Python's source code is fairly easy-to-maintain.
- **A broad standard library**: Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.

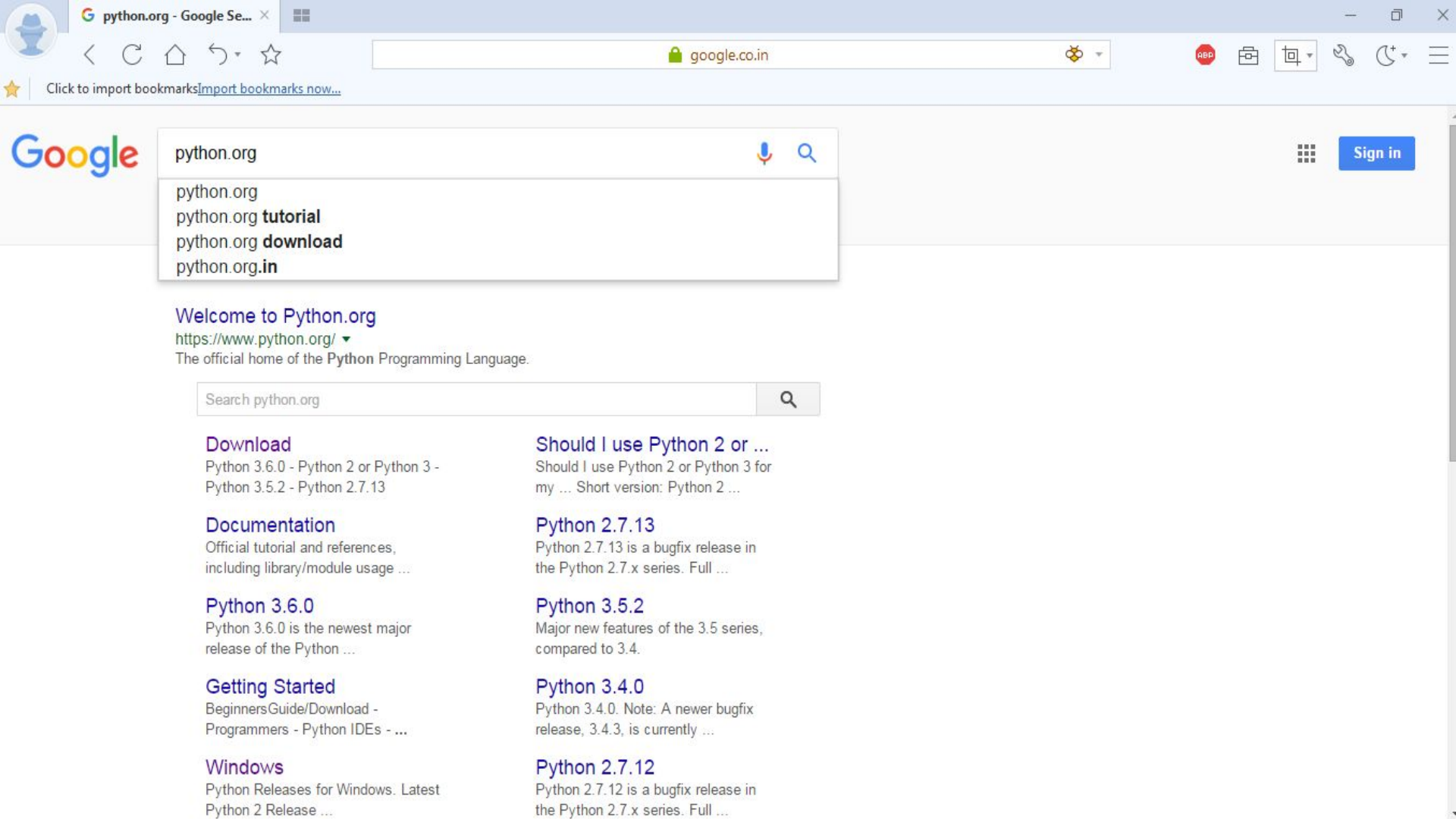
Python Feature

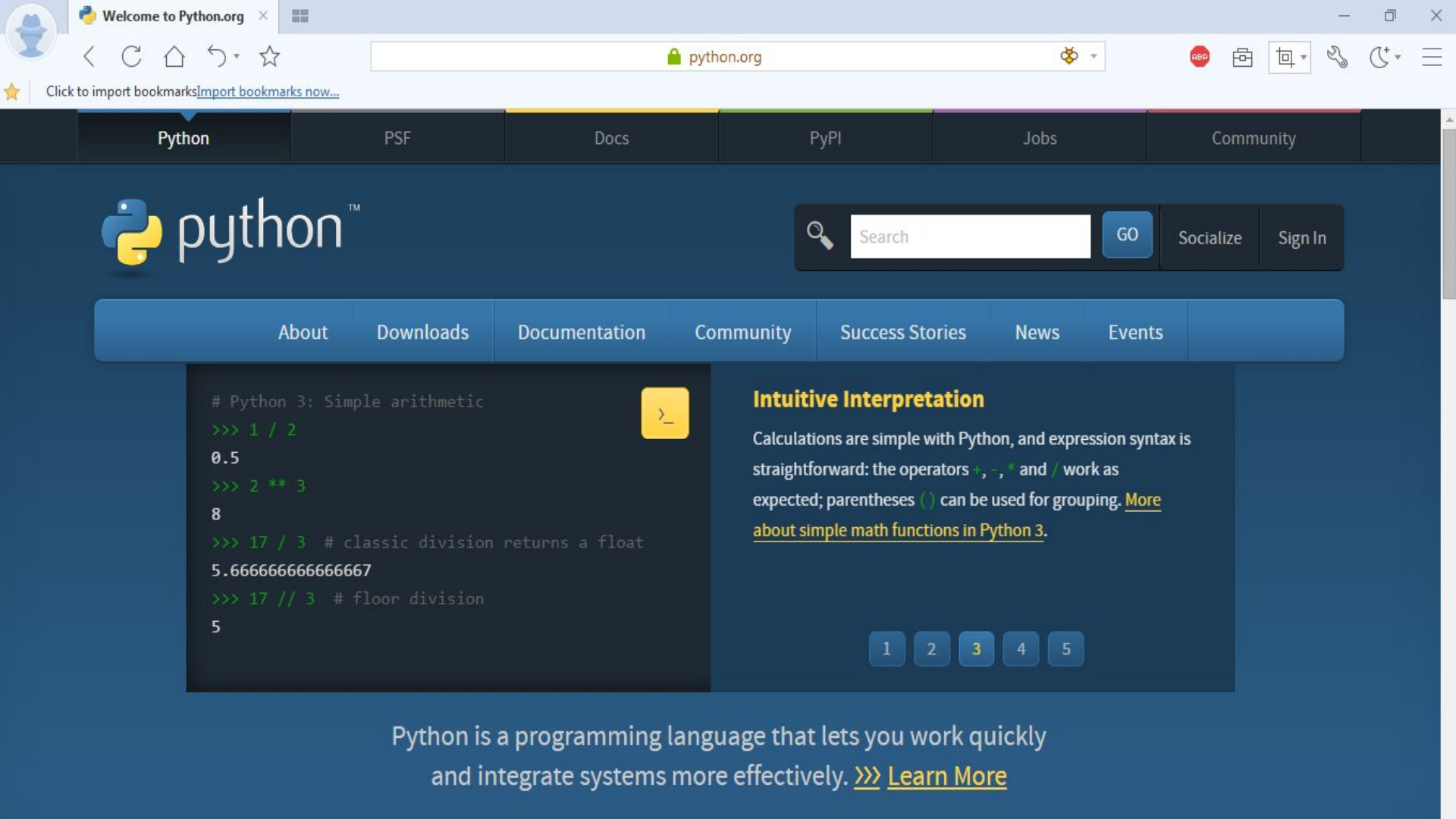
- **Interactive Mode:** Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.
- **Portable:** Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- **Extendable:** You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
- **Databases:** Python provides interfaces to all major commercial databases.

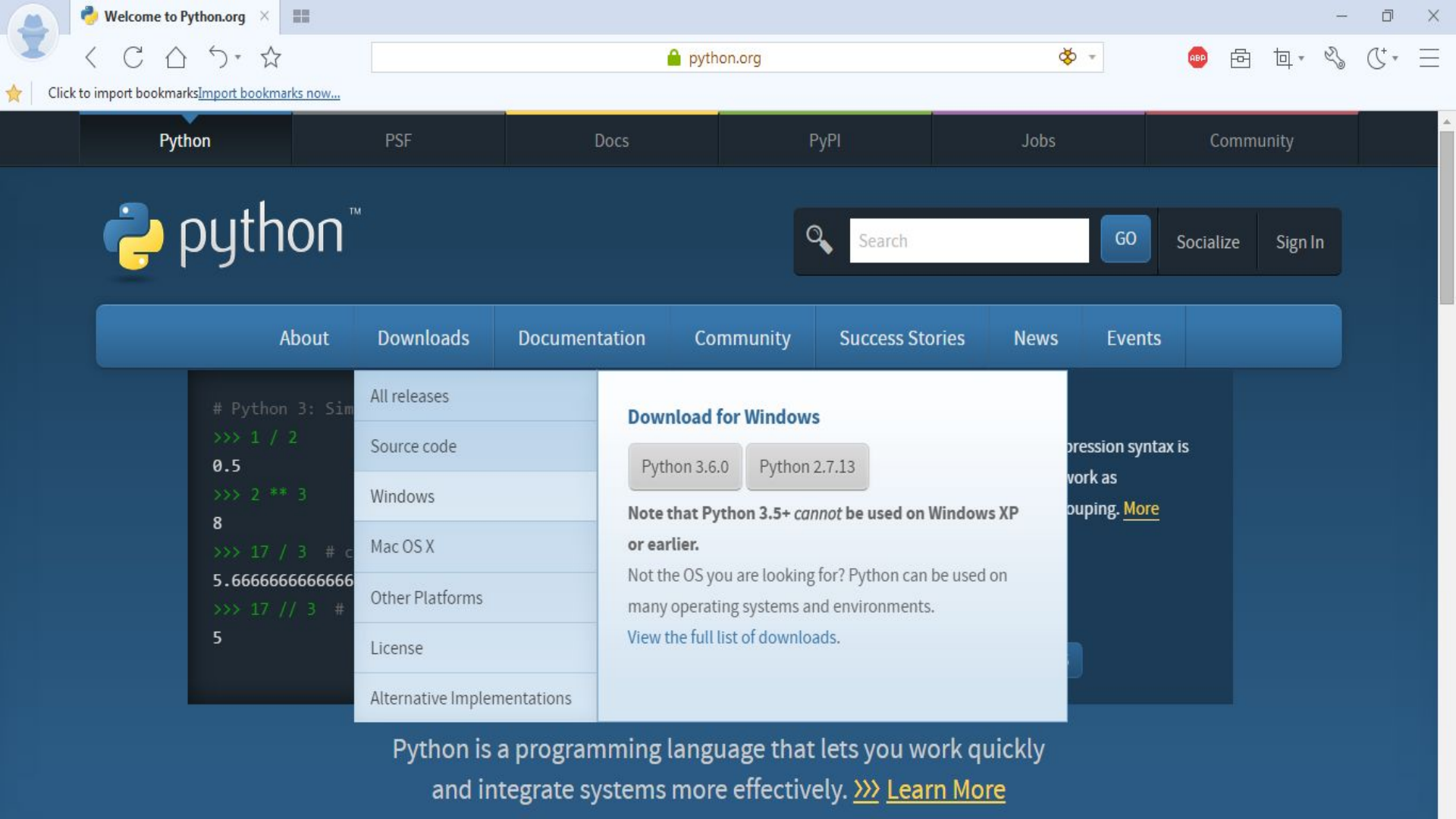
Python Feature

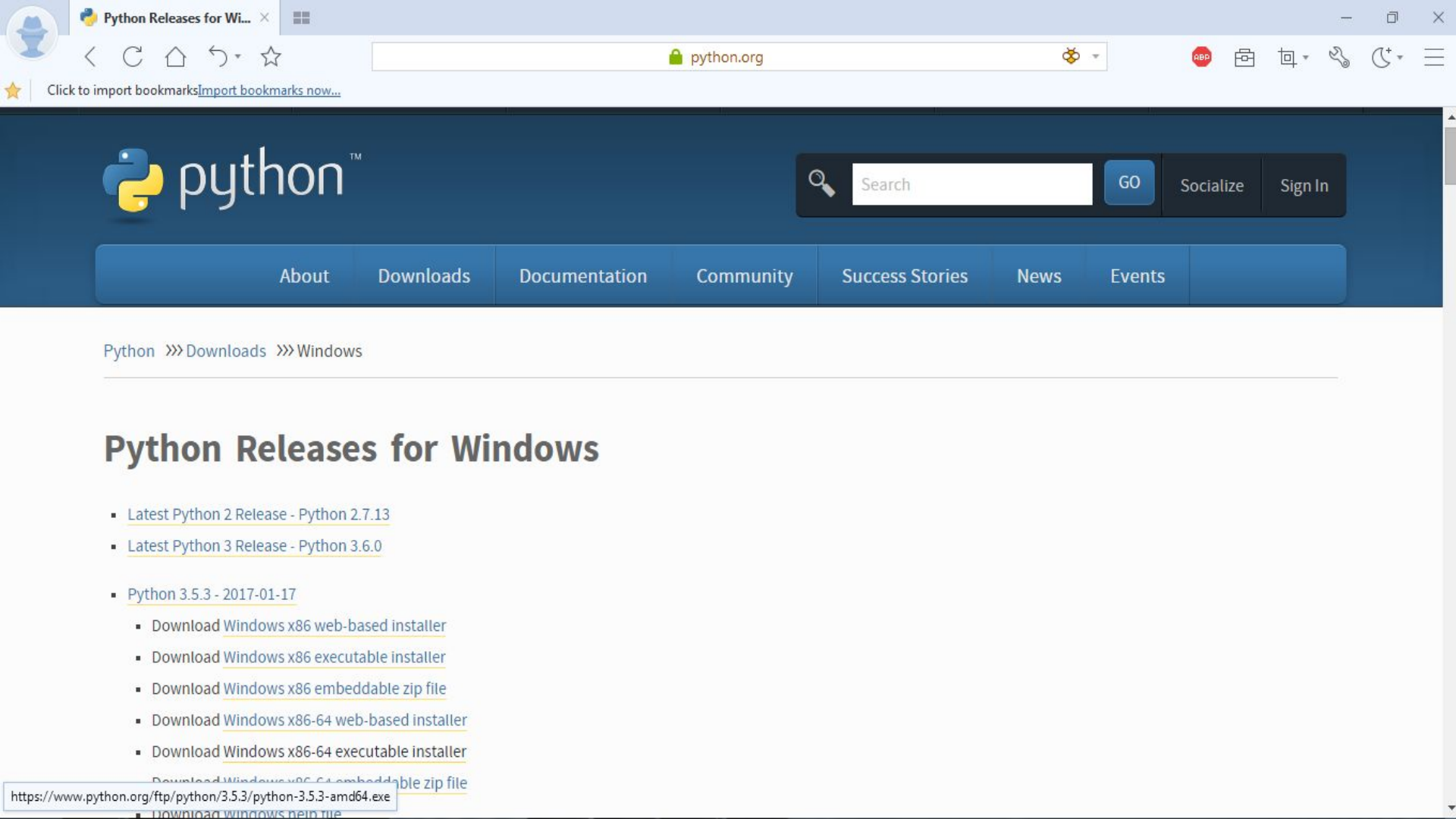
- **GUI Programming:** Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.
- **Scalable:** Python provides a better structure and support for large programs than shell scripting

Getting Python


python.org - Google Se...
google.co.in
Click to import bookmarksImport bookmarks now...
Google
python.org
Sign in
python.org
python.org tutorial
python.org download
python.org.in
Welcome to Python.org
https://www.python.org/
The official home of the Python Programming Language.
Search python.org
Download
Python 3.6.0 - Python 2 or Python 3 - Python 3.5.2 - Python 2.7.13
Should I use Python 2 or ...
Should I use Python 2 or Python 3 for my ... Short version: Python 2 ...
Documentation
Official tutorial and references, including library/module usage ...
Python 2.7.13
Python 2.7.13 is a bugfix release in the Python 2.7.x series. Full ...
Python 3.6.0
Python 3.6.0 is the newest major release of the Python ...
Python 3.5.2
Major new features of the 3.5 series, compared to 3.4.
Getting Started
BeginnersGuide/Download - Programmers - Python IDEs - ...
Python 3.4.0
Python 3.4.0. Note: A newer bugfix release, 3.4.3, is currently ...
Windows
Python Releases for Windows. Latest Python 2 Release ...
Python 2.7.12
Python 2.7.12 is a bugfix release in the Python 2.7.x series. Full ...

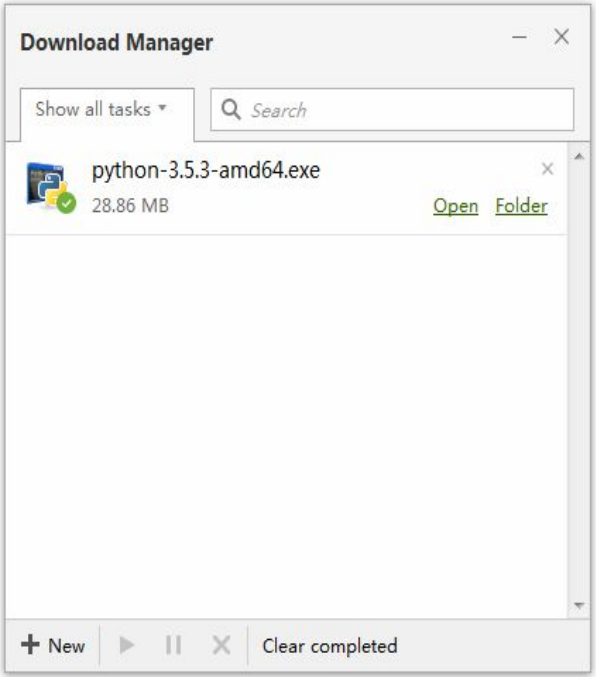

Welcome to Python.org
python.org
Click to import bookmarks Import bookmarks now...
Python
PSF
Docs
PyPI
Jobs
Community
python
Search
GO
Socialize
Sign In
About
Downloads
Documentation
Community
Success Stories
News
Events
# Python 3: Simple arithmetic
>>> 1 / 2
0.5
>>> 2 ** 3
8
>>> 17 / 3 # classic division returns a float
5.666666666666667
>>> 17 // 3 # floor division
5
Intuitive Interpretation
Calculations are simple with Python, and expression syntax is straightforward: the operators +, -, * and / work as expected; parentheses ( ) can be used for grouping. More about simple math functions in Python 3.
1
2
3
4
5
Python is a programming language that lets you work quickly and integrate systems more effectively. >>> Learn More


Welcome to Python.org
python.org
Click to import bookmarksImport bookmarks now...
Python
PSF
Docs
PyPI
Jobs
Community
python™
Search
GO
Socialize
Sign In
About
Downloads
Documentation
Community
Success Stories
News
Events
# Python 3: Sim
>>> 1 / 2
0.5
>>> 2 ** 3
8
>>> 17 / 3 # c
5.666666666666
>>> 17 // 3 #
5
All releases
Source code
Windows
Mac OS X
Other Platforms
License
Alternative Implementations
Download for Windows
Python 3.6.0
Python 2.7.13
Note that Python 3.5+ cannot be used on Windows XP or earlier.
Not the OS you are looking for? Python can be used on many operating systems and environments.
View the full list of downloads.
pression syntax is
work as
ouping. More
Python is a programming language that lets you work quickly and integrate systems more effectively. >>> Learn More


Python Releases for Wi...
python.org
Click to import bookmarks Import bookmarks now...
python
Search
GO
Socialize
Sign In
About
Downloads
Documentation
Community
Success Stories
News
Events
Python ››› Downloads ››› Windows
Python Releases for Windows
Latest Python 2 Release - Python 2.7.13
Latest Python 3 Release - Python 3.6.0
Python 3.5.3 - 2017-01-17
Download Windows x86 web-based installer
Download Windows x86 executable installer
Download Windows x86 embeddable zip file
Download Windows x86-64 web-based installer
Download Windows x86-64 executable installer
https://www.python.org/ftp/python/3.5.3/python-3.5.3-amd64.exe

- Latest Python 2 Release - Python 2.7.13
- Latest Python 3 Release - Python 3.6.0

- Python 3.5.3 - 2017-01-17
  - Download Windows x86 web-based installer
  - Download Windows x86 executable installer
  - Download Windows x86 embeddable zip file
  - Download Windows x86-64 web-based installer
  - Download Windows x86-64 executable installer
  - Download Windows x86-64 embeddable zip file
  - Download Windows help file
- Python 3.5.3rc1 - 2017-01-03
  - Download Windows x86 web-based installer
  - Download Windows x86 executable installer
  - Download Windows x86 embeddable zip file
  - Download Windows x86-64 web-based installer
  - Download Windows x86-64 executable installer
  - Download Windows x86-64 embeddable zip file
  - Download Windows help file
- Python 3.6.0 - 2016-12-23
  - Download Windows x86 web-based installer



Python 3.5 (64-bit)

```
Python 3.5.3 (v3.5.3:1880cb95a742, Jan 16 2017, 16:02:32) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

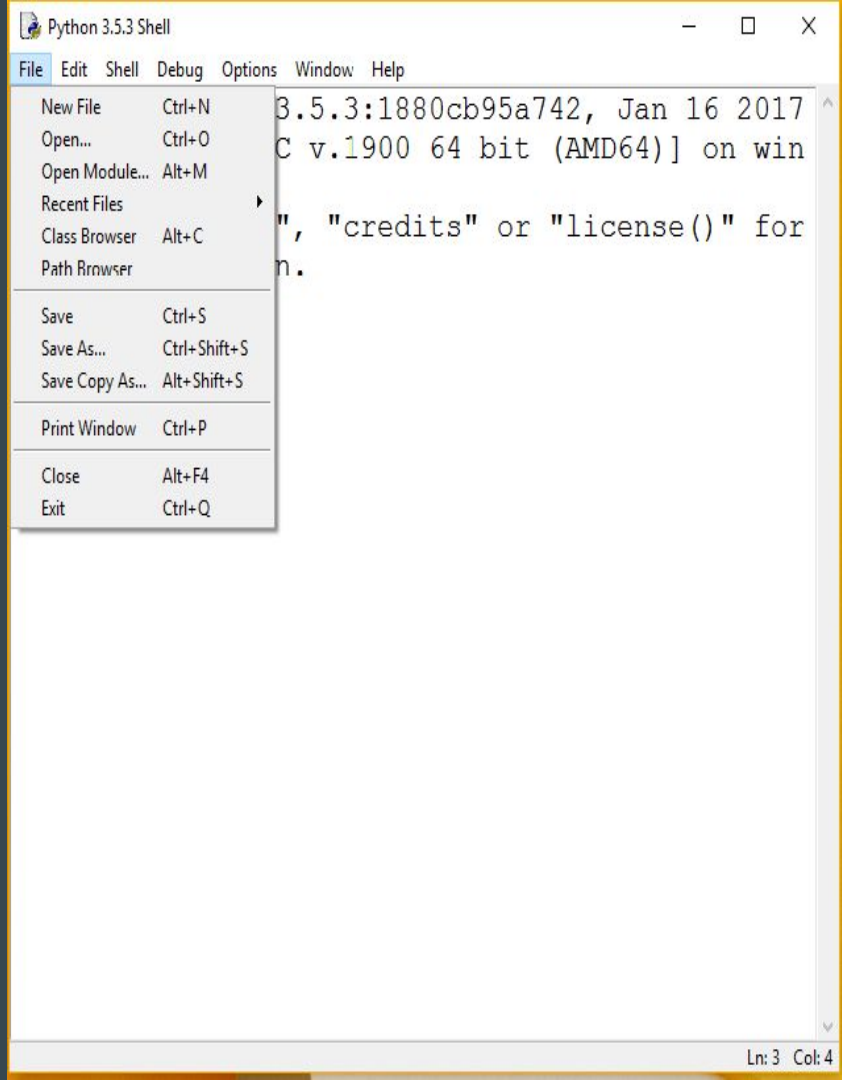
Python 3.5.3 Shell

File Edit Shell Debug Options Window Help

```
Python 3.5.3 (v3.5.3:1880cb95a742, Jan 16 2017, 16:02:32) [MSC v.1900 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
```

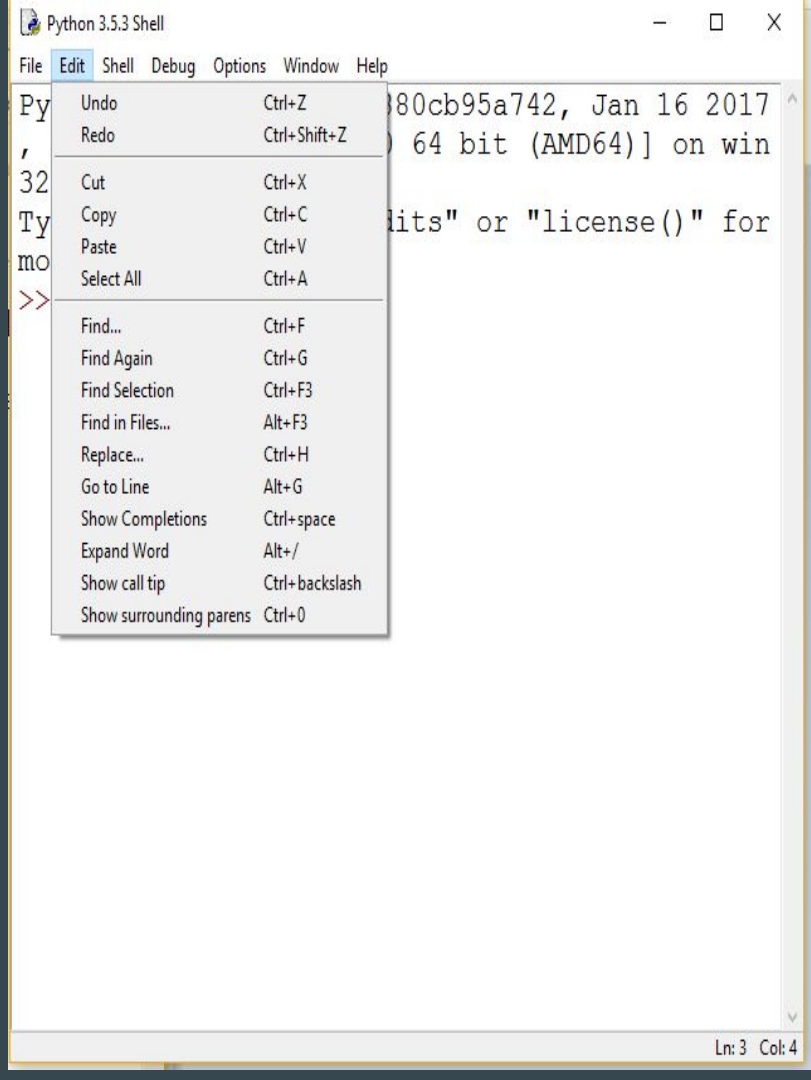
IDLE SHELL (FILE MENU)

Entity	Function
New File	Create New File
Open	Open Existing File
Recent File	Open list of Recent File
Open Module	Open Existing Module
Class Browser	Show functions, classes
Path Browser	Show sys.path Directories,Module etc
Save	Save Current Window
Save As	Save with New Associated File
Save Copy As	Save Current window to different file
Print Window	Print current window to default Printer
close	Close Current Window
Exit	Close all Window and Quit



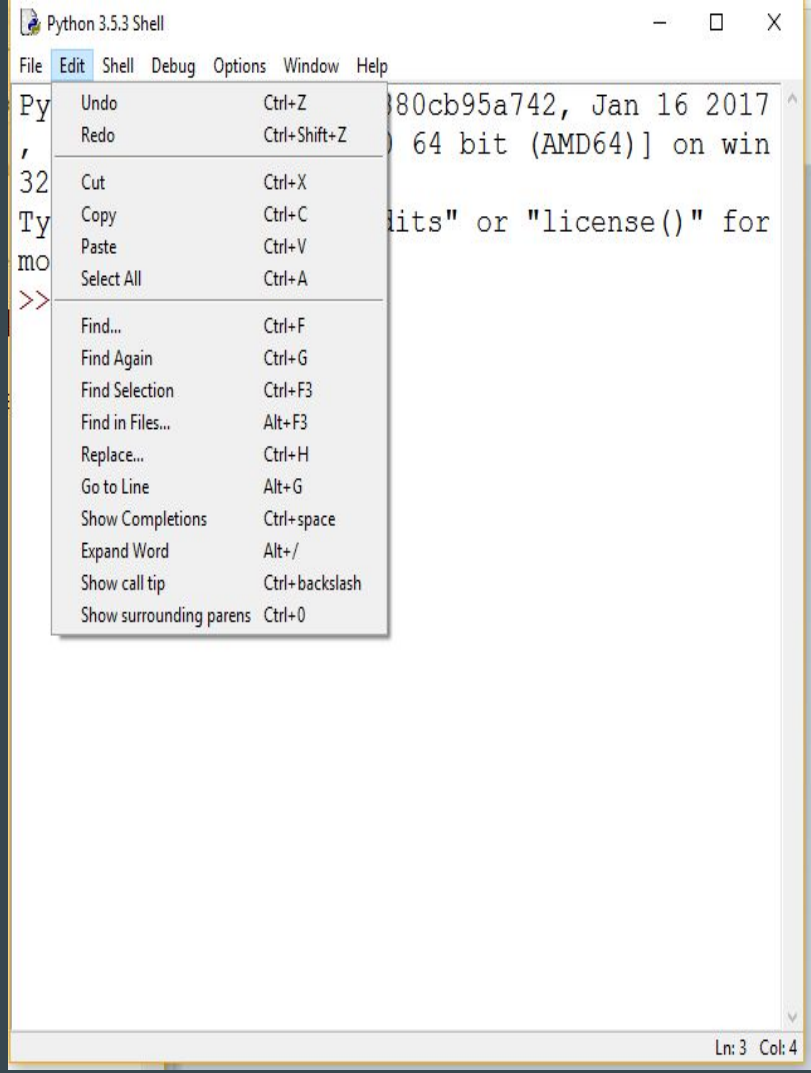
IDLE SHELL (Edit MENU)

Entity	Function
Undo	Undo the last change of current window
Redo	Redo the last undone change to current window
Cut	Copy selection than delete selection
Copy	Copy selection into system-wide clipboard
Paste	Insert contents of system wide clipboard
Select all	Select entire contents of current window
Find	Open search dialog with many option
Find selection	Search for the currently selected string
Find in files	Open a file search dialog
Replace	Open a search and replace dialog
Go to line	Move cursor to the line number requested make a line visible



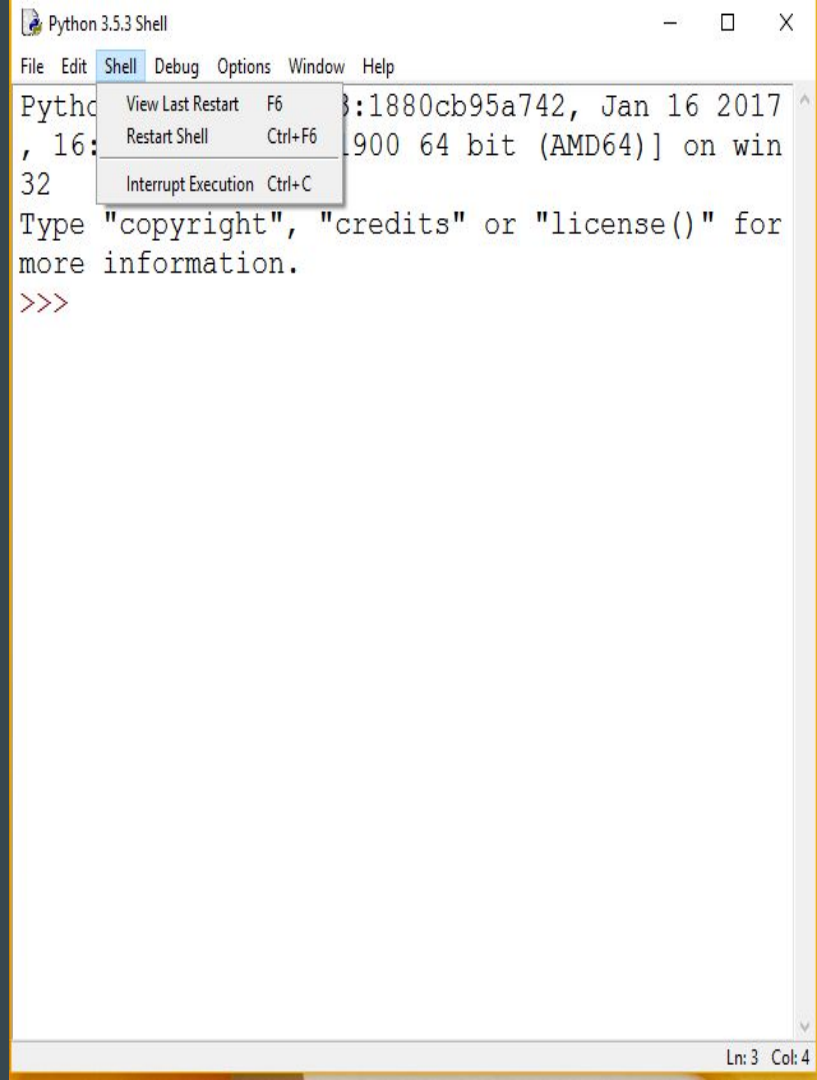
IDLE SHELL (Edit MENU)

Entity	Function
Show Completions	Open a scrollable list allowing selection of keyword and attribute
Expand word	Expand a prefix you have type to match a full word in the same window
Show call tip	After an unclosed parenthesis for a function , open a small window with function parameter hints
Show surrounding parens	Highlight the surrounding parenthesis



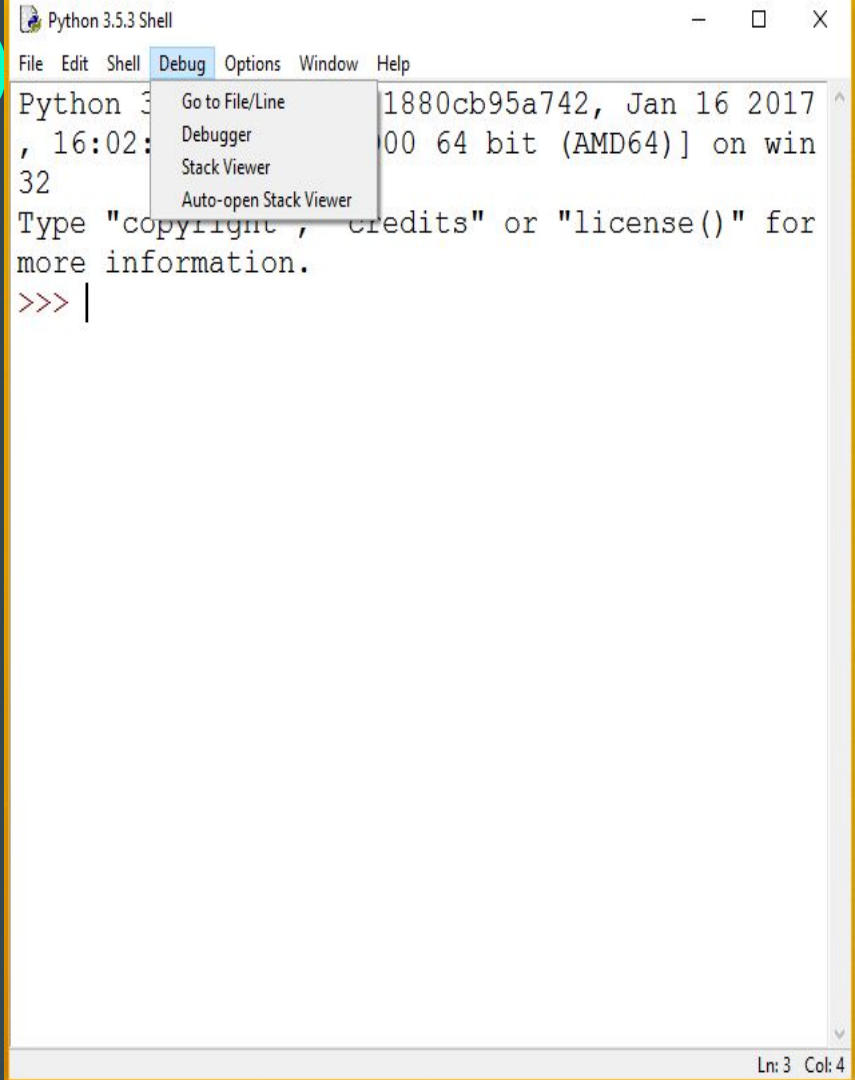
IDLE SHELL (Shell MENU)

Entity	Function
View Last Restart	Scroll the shell window to the last shell restart
Restart shell	Restart shell to clean the environment
Interrupt Execution	Stop a running program



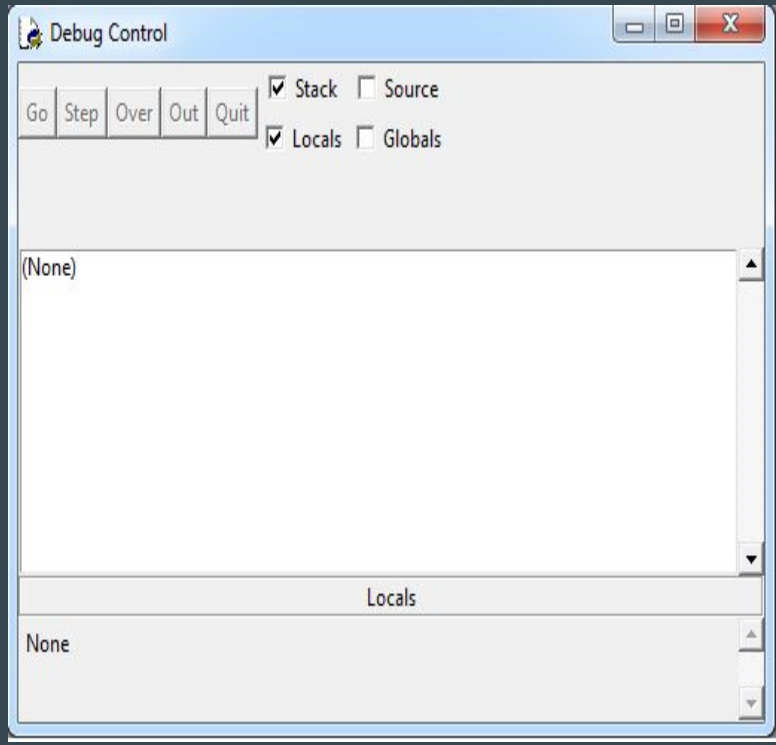
IDLE SHELL (Debug MENU)

Entity	Function
Go to file/line	Look on the current line, for a file name and line number
Debugger	Code entered in shell or run from an editor will run under the debugger
Stack viewer	Show the stack traceback of last exception with local and global
Auto open stack viewer	Toggle automatically opening the stack viewer on an unhandled exception



Debug MENU (Debugger)

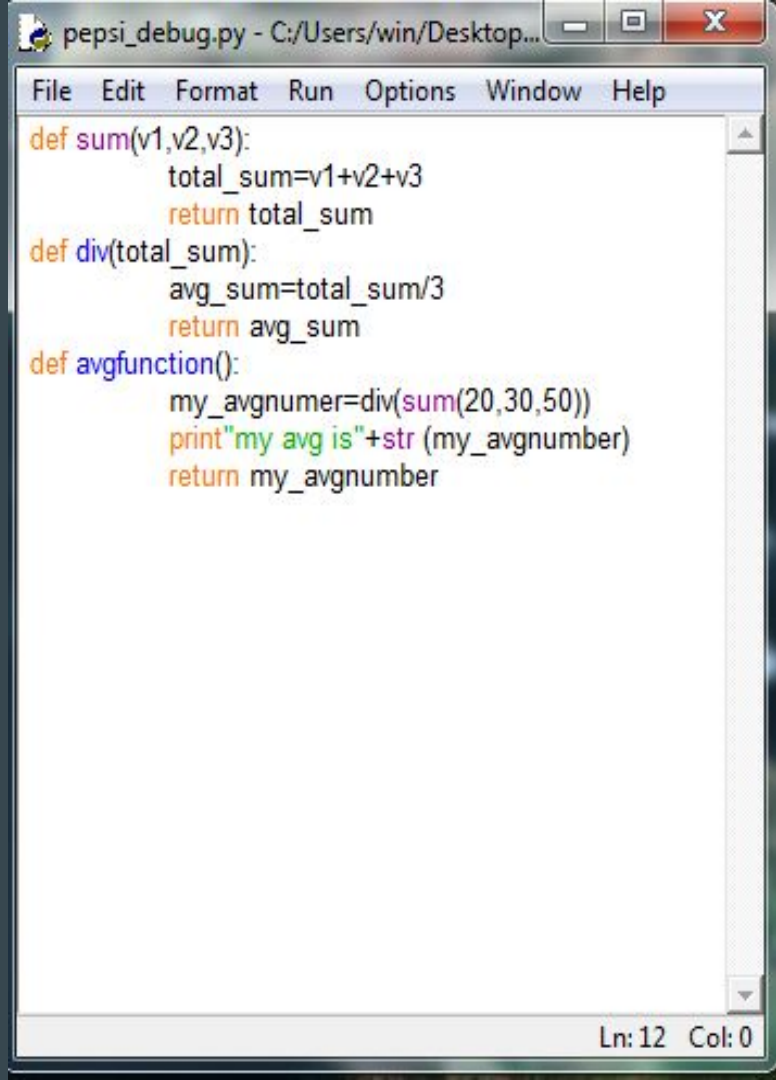
- Debug control and it's entities



Debug MENU (Debugger)

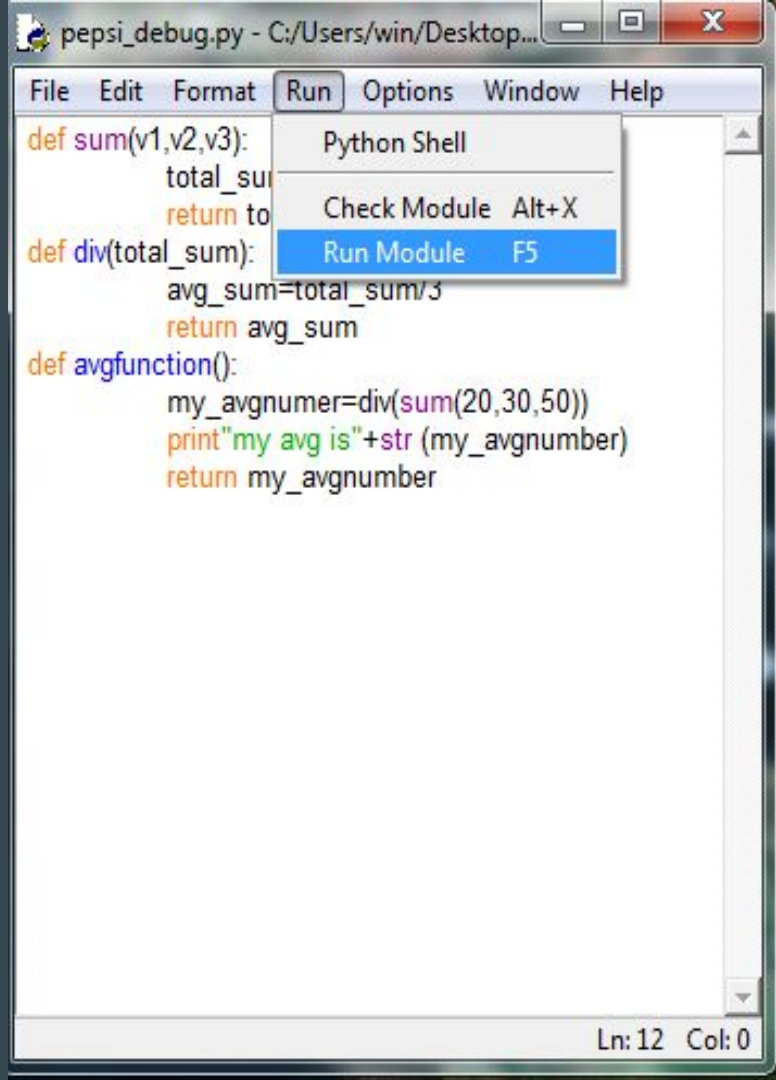
Problem statement : write a program an average of the three value[20,30,50].

- **STEP 1**: write a program in new file.



Debug MENU (Debugger)

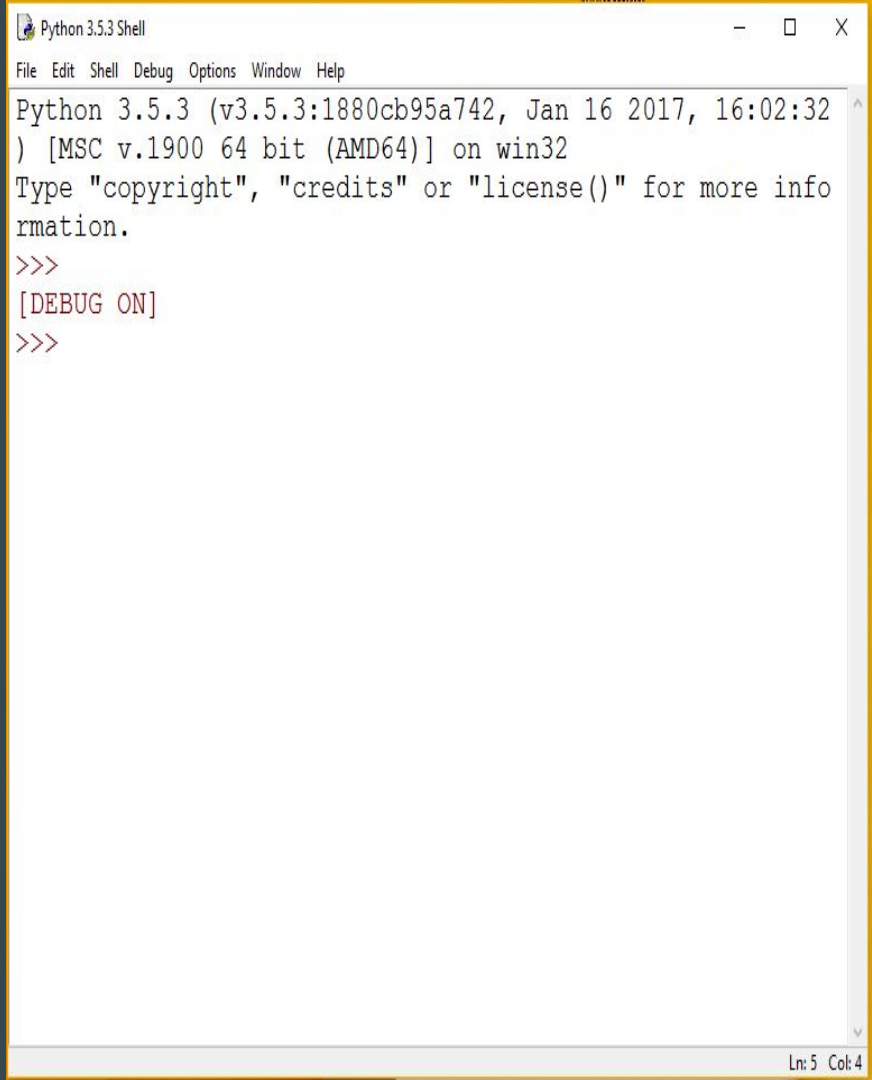
- **STEP 2**: Run the Module



Debug MENU (Debugger)

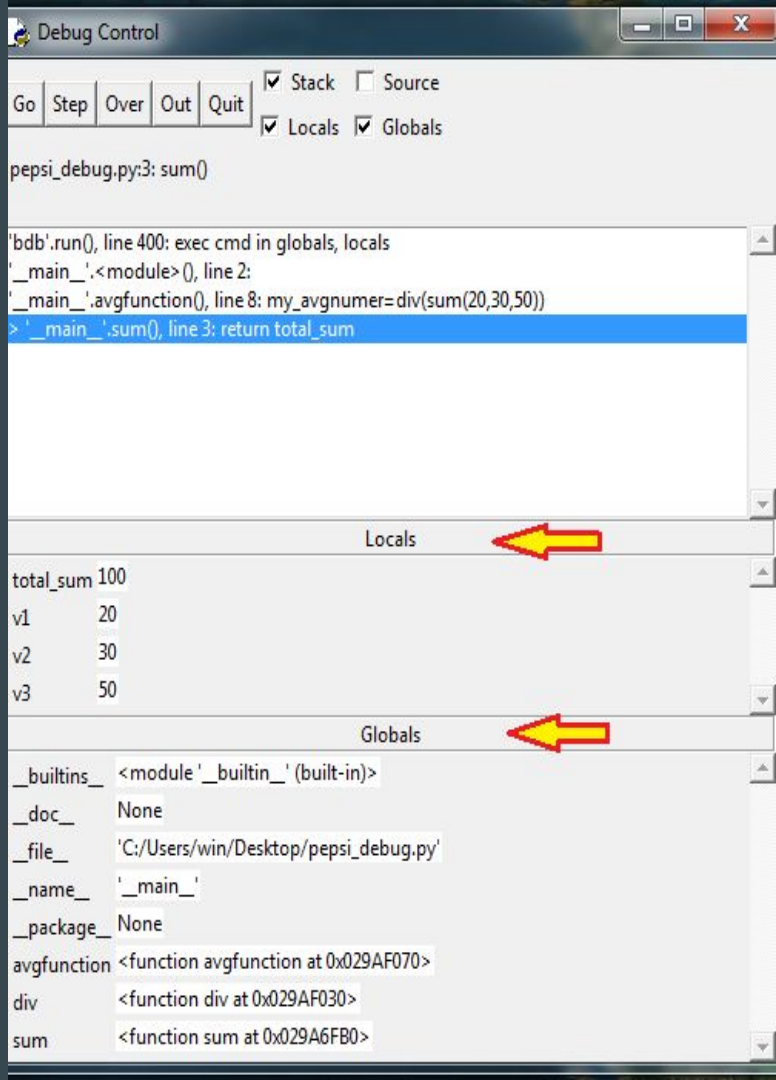
- **STEP 3** :On the Debug

- **STEP 4** :Call a Function



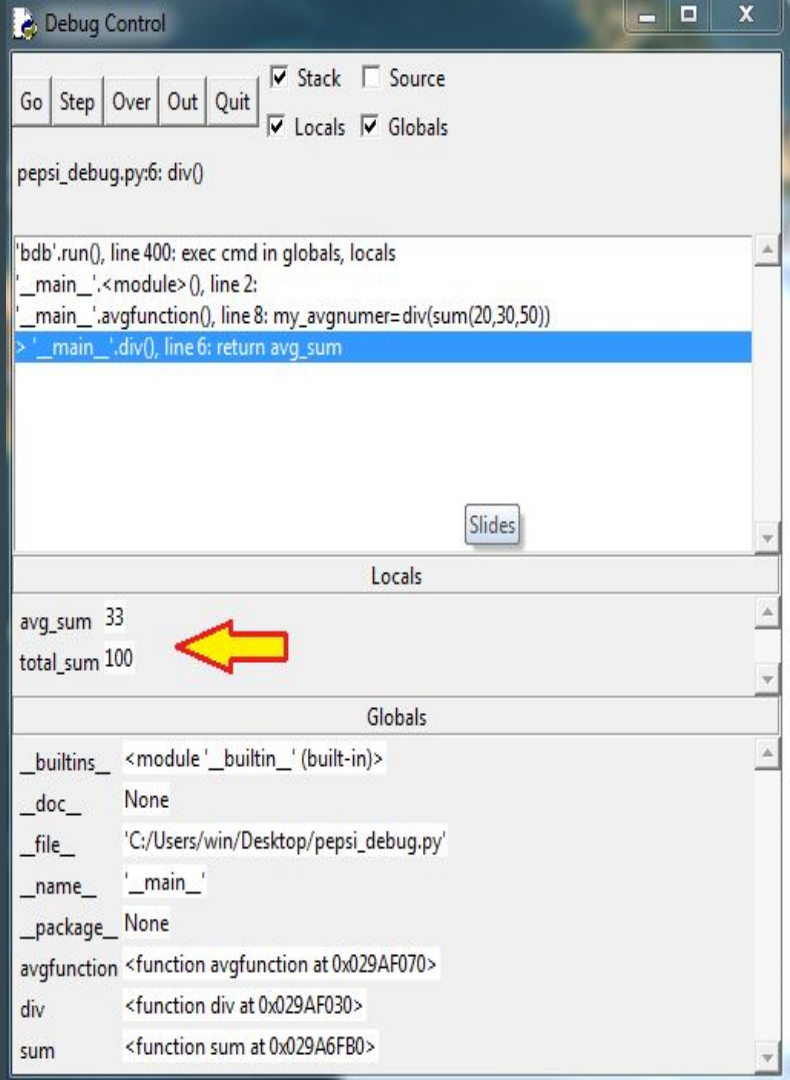
Debug MENU (Debugger)

Step 5 : Check all step one by one by clicking the step button



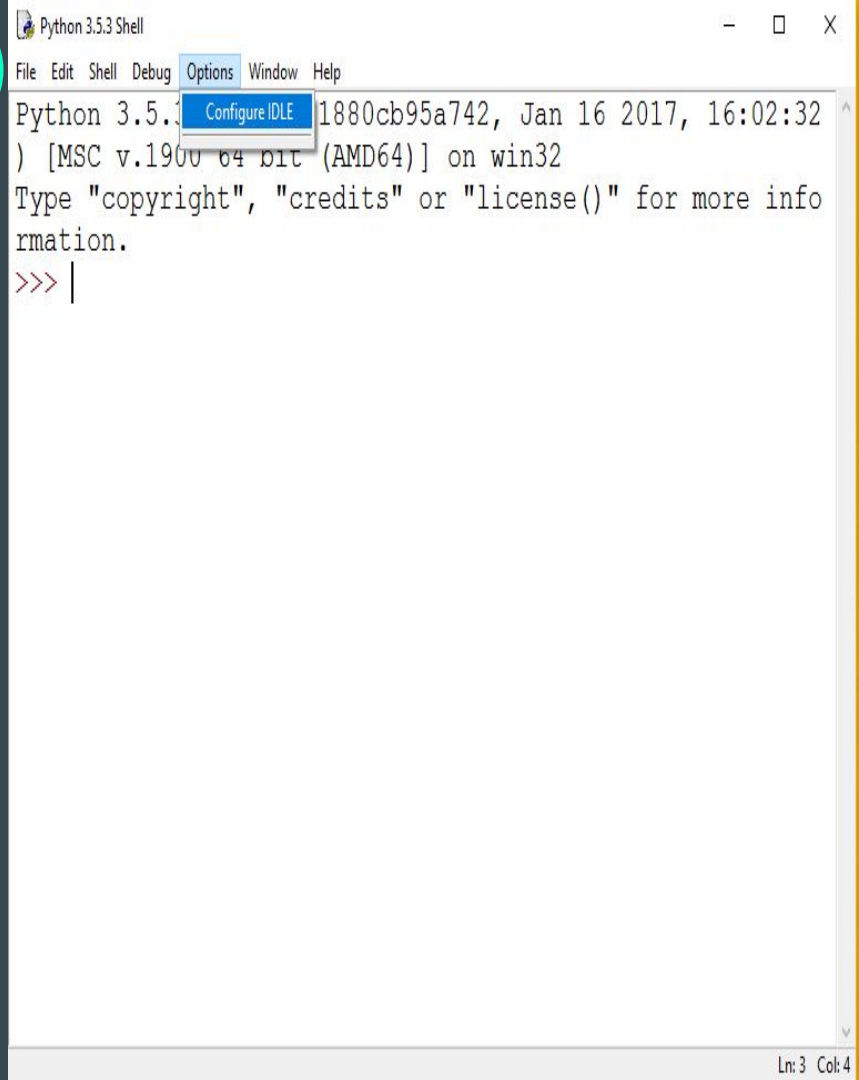
Debug MENU (Debugger)

- **STEP 6**: Getting an average of the three value[20,30,50].



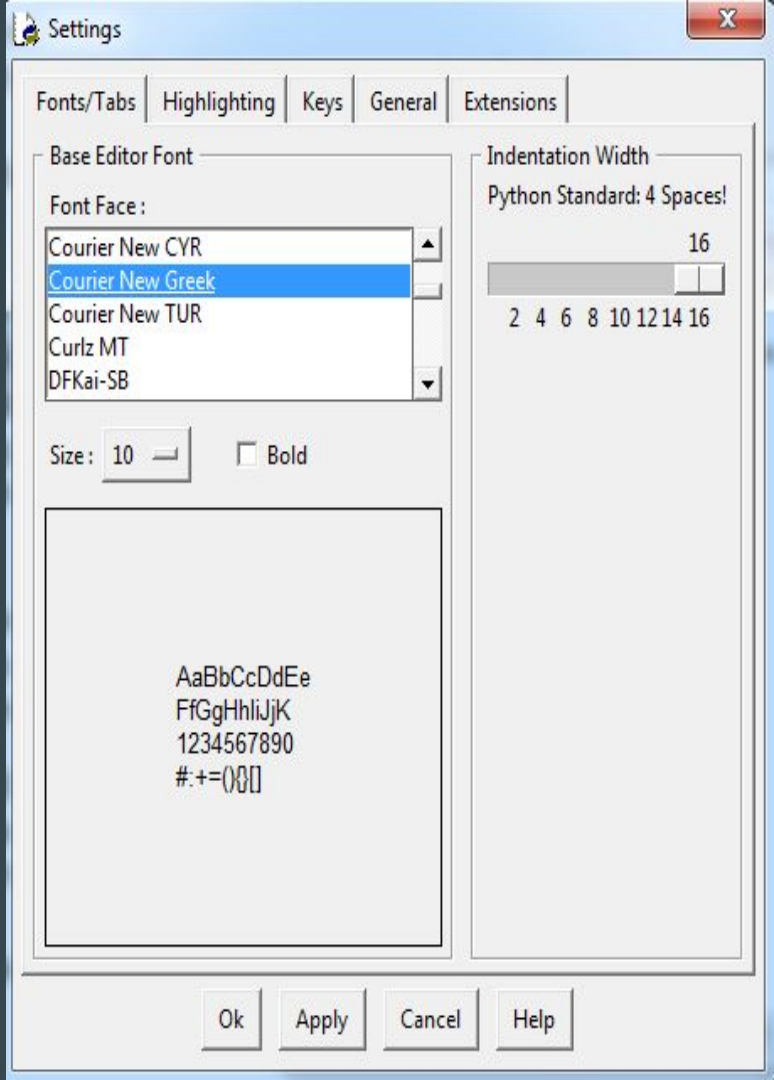
IDLE SHELL (option MENU)

Entity	Function
Configure Idle	change preferences for the following: fonts, indentation, key bindings, text color themes, startup windows and size, additional help sources, and extensions



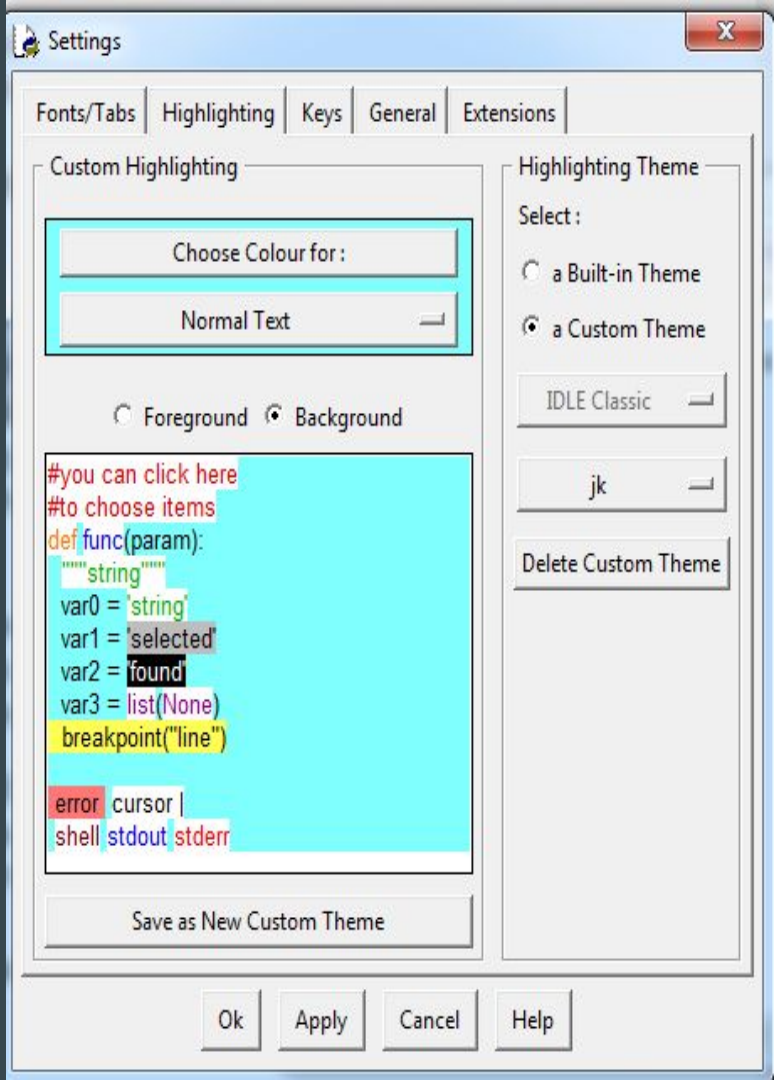
Option MENU (Font & Tabs)

- It has .....
  - Font Face
  - Indentation Width



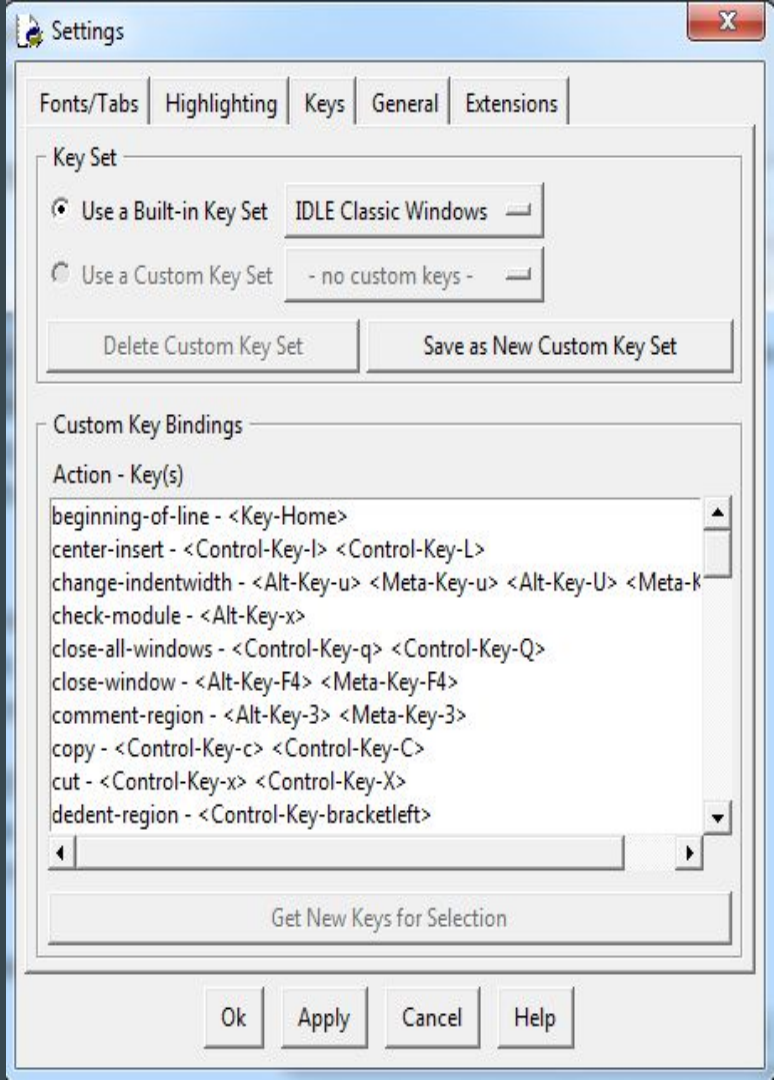
OptionMENU (Highlighting)

- It Has.....
  - Custom Highlighting
  - Highlighting Theme
  - Foreground and Background



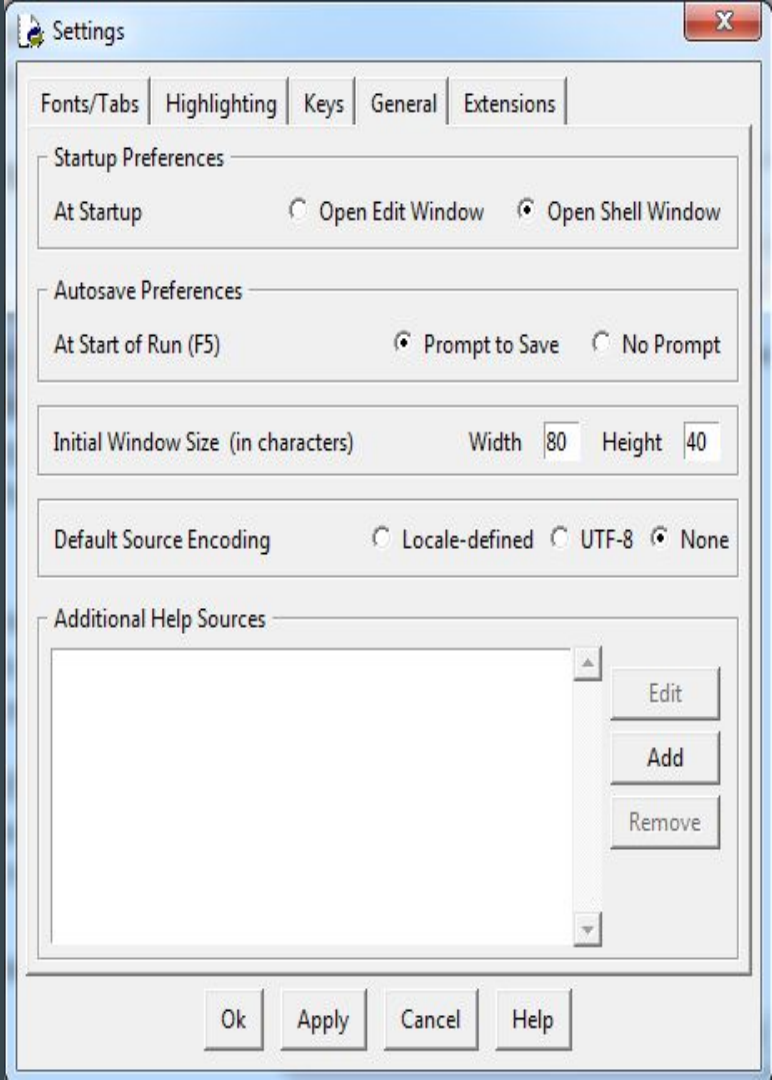
Option MENU (Keys)

- It Has...
  - Key set
  - Custom Key Building



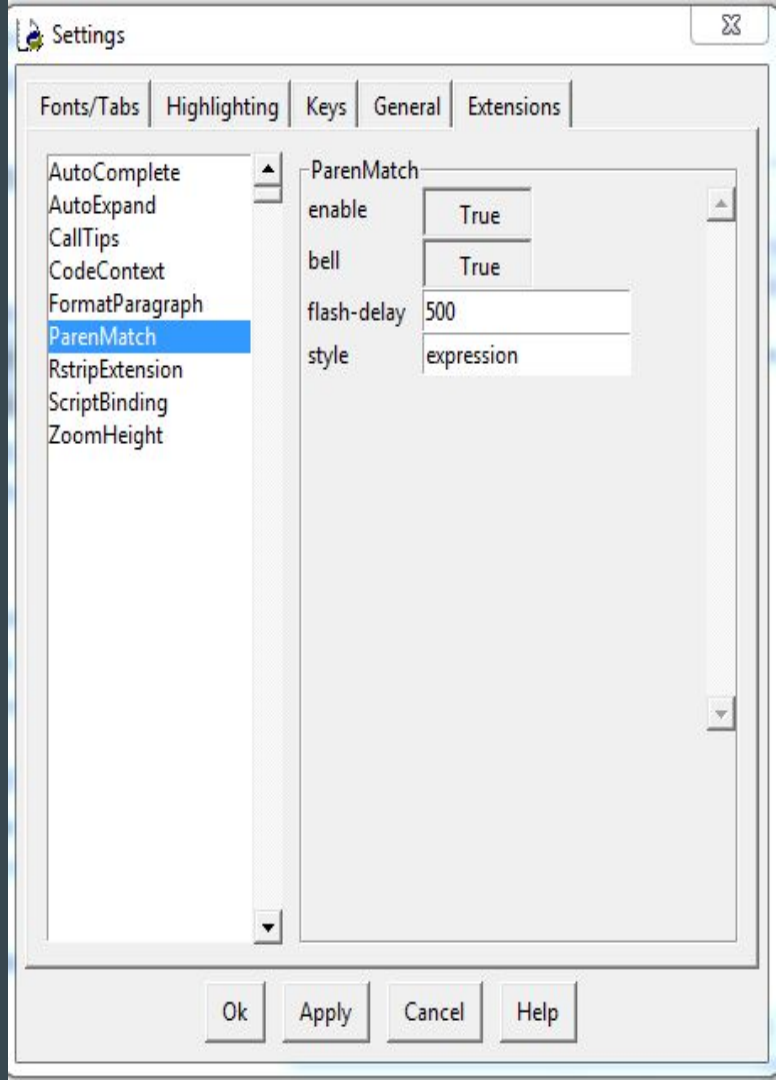
Option MENU (General)

- It Has.....
  - Start up preferences
  - Auto save Preferences
  - Initial Window Size
  - Default Source Coding



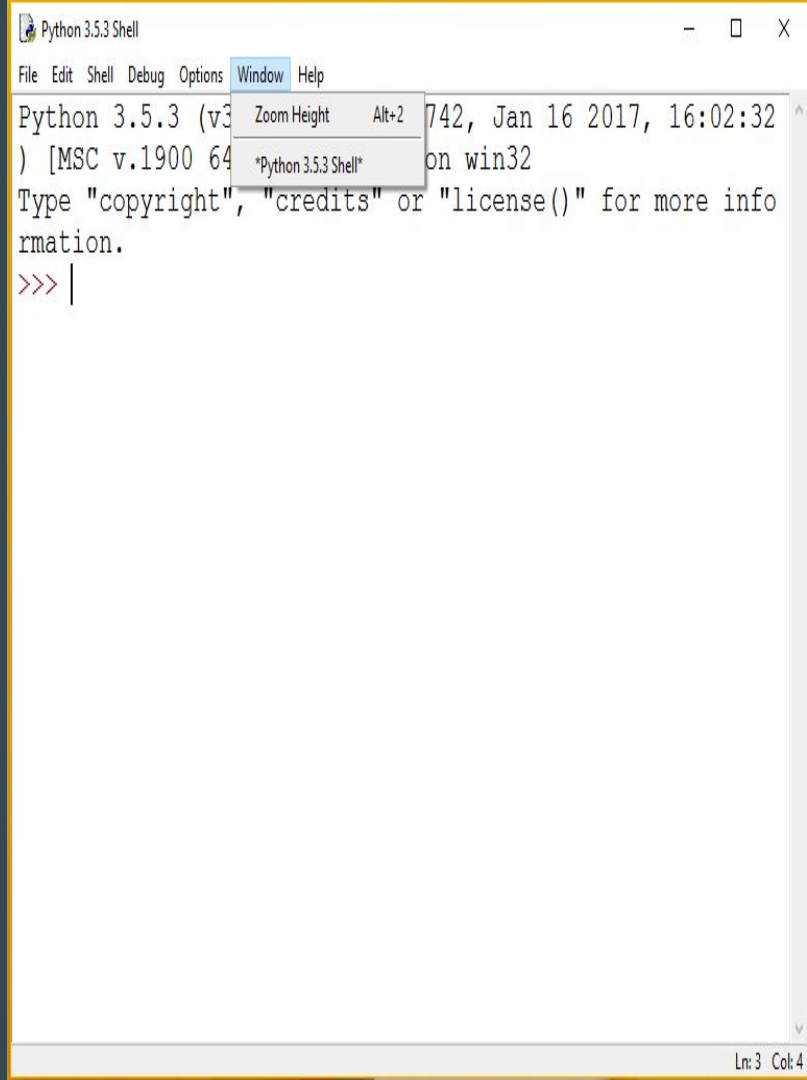
Option MENU (Extensions)

- It Has.....
  - Auto Complete
  - Auto Expand
  - Call tips
  - Code context
  - Format Paragraph
  - Parenmatch
  - Rstripextension
  - Script Binding
  - Zoom Height



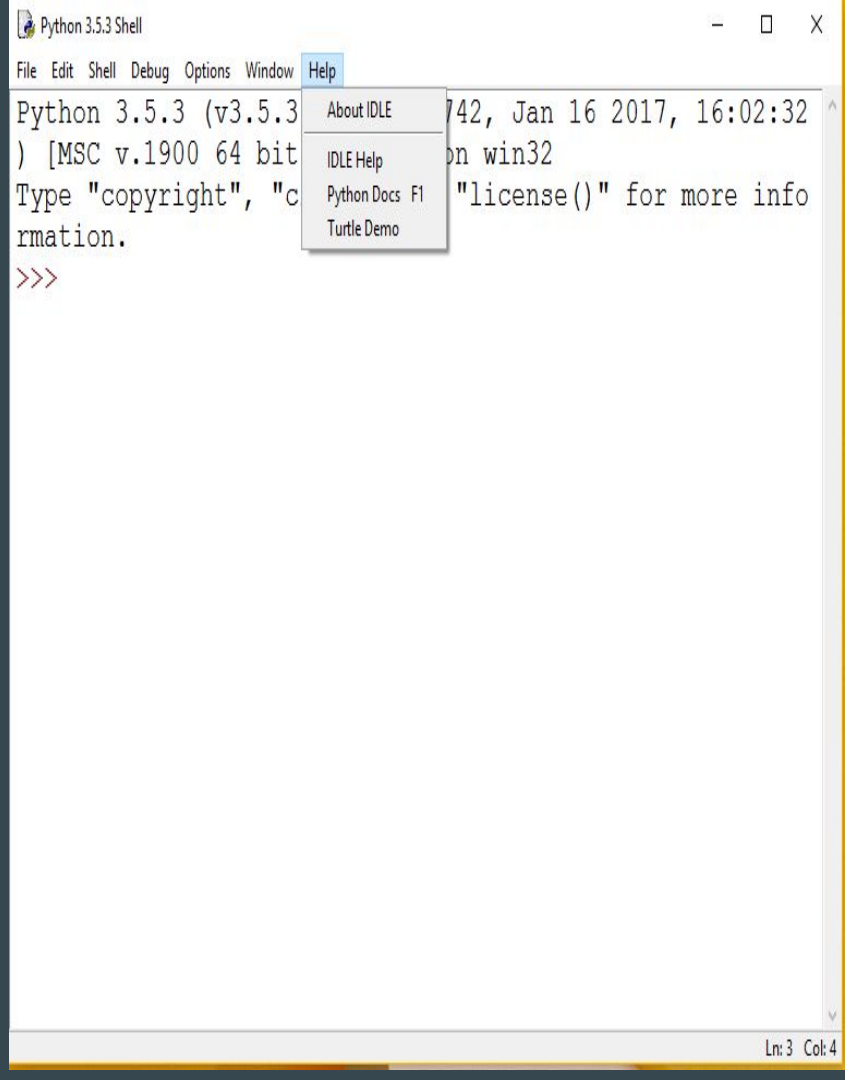
IDLE SHELL (Window Menu)

Entity	Function
Zoom Height	Toggles the window between normal size and maximum height. The initial size defaults to 40 lines by 80 char.



IDLE SHELL (Help MENU)

Entity	Function
About IDLE	Display version, copyright, license, credits, and more
IDLE Help	Display a help file for IDLE detailing the menu options, basic editing and navigation, and other tips.
Python Docs	Access local Python documentation



Variable Type

What is variable?

- Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.
- Based on the datatype of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory.
- Can store integers, decimals or characters in these variables.

Assign Value to Variable

- Python variables do not need explicit declaration to reserve memory space.
- The declaration happens automatically when you assign a value to a variable. The equal sign (=) is used to assign values to variables.

= Operator	It's Mean
Left Side	Name of the variable
Right Side	Value stored in the variable

Assign Value to Variable

- Write a program that assign 10,50.00 and jack are the values assigned to counter, kilometer and name variables.

```
>>> counter=10 # An integer assignment
>>> kilometer=50.00 # A floating point
>>> name="jack" # A string
>>>
>>> print counter
10
>>> print kilometer
50.0
>>> print name
jack
>>>
```

Multiple Assignment

- Python allows you to assign a single value to several variables simultaneously.

- Write a program ,assign the same value to the three different variable

```
>>> x=y=z=7
>>> print x
7
>>> print y
7
>>> print z
7
>>>
```

Multiple Assignment

- Assign multiple objects to multiple variables

- Write a program , assign two values to two variable and one variable with string Object.

```
>>> x,y,z=7,8,"jack" # Multiple  Assignment
>>> print x
7
>>> print y
8
>>> print z
jack
>>>
```

Basic Operator

What is Operator ?

- Operators are the constructs which can manipulate the value of operands

- Consider the expression 1 + 6 = 7. Here, 1 and 6 are called operands and + is called operator

Operator Types

1	Arithmetic Operators
2	Comparison (Relational) Operators
3	Assignment Operators
4	Logical Operators
5	Bitwise Operators
6	Membership Operators
7	Identity Operators

Arithmetic Operators

Operator	Description
+ Addition	Adds values on either side of the operator
- Subtraction	Subtracts right hand operand from left hand operand
* Multiplication	Multiplies values on either side of the operator
/ Division	Divides left hand operand by right hand operand
% Modulus	Divides left hand operand by right hand operand and returns remainder
** Exponent	Performs exponential (power)
// Floor Division	Division of operands where the result is the quotient

```
>>> # example of the different Arithmetic Operators
>>>
>>> a=10 # Assign value to the first variable
>>> b=5  # Assign value to the second variable
>>>
>>> a+b # Addition
15
>>> a-b # subtraction
5
>>> a*b # muliplication
50
>>> a/b #division
2
>>> a%b #modulus
0
>>> a**b #exponent
100000
>>> a//b #floor division
2
>>>
```

Arithmetic Operators

- Difference b/w Division and Floor Division

```
>>> # Differnce b/w division and floor division
>>>
>>> a=-4.3333 # assign value to the first varible
>>> b= 2   # assign value to the second variable
>>>
>>> a/b # division operation
-2.16665
>>>
>>> a//b # floor division
-3.0
>>> # digits after the decimal point are removed. But if one of the operan
ds is negative, the result is floored, i.e., rounded away from zero
>>>
```

Comparison Operators

Operator	Description
==	If the values of two operands are equal, then the condition becomes true
!=	If values of two operands are not equal, then condition becomes true.
>	left operand is greater than the value of right operand, then condition becomes true.
<	left operand is less than the value of right operand, then condition becomes true
>=	left operand is greater than or equal to the value of right operand, then condition becomes true
<=	left operand is less than or equal to the value of right operand, then condition becomes true

```
>>> #  example of comparison operators
>>>
>>> a=7 # assign value
>>> b=5 # assign value
>>>
>>> a==b # == operator
False
>>> a!=b # != operator
True
>>> a>b # > operator
True
>>> a<b # < operator
False
>>> a>=b # >= operator
True
>>> a<=b # <= operator
False
>>>
>>> a=10
>>> b=10
>>> a==b
True
>>> a>=b
True
>>> a<=b
True
```

Assignment Operators

Operator	Description
=	Assigns values from right side operands to left side operand
+=	It adds right operand to the left operand and assign the result to left operand
-=	It subtracts right operand from the left operand and assign the result to left operand
*=	It multiplies right operand with the left operand and assign the result to left operand
/=	It divides left operand with the right operand and assign the result to left operand

```
>>> # assignment operators
>>> a=10 # assign value
>>> b=7 # assign value
>>> c=a+b # assign value of a+b into c
>>> c
17
>>> c+=a # c=c+a
>>> c
27
>>> c-=a # c=c-a
>>> c
17
>>> c*=a # c=c*a
>>> c
170
>>> c/=a # c=c/a
>>> c
17
>>>
```

Assignment Operators

Operator	Description
%=	It takes modulus using two operands and assign the result to left operand
**=	Performs exponential (power) calculation on operators and assign value to the left operand
//=	It performs floor division on operators and assign value to the left operand

```
>>> a=10 # assign value
>>> b=5 # assign value
>>> c=a+b
>>> c
15
>>> c%=a  #  c=c%a
>>> c
5
>>> c**=a #  c=c**a
>>> c
9765625
>>>
>>> c=20.23
>>>
>>> c//=a # c=c//a
>>> c
2.0
>>>
```

Logical Operators

Operator	Description	Example
and	If both the operands are true then condition becomes true.	(a and b) is true
or	If any of the two operands are non-zero then condition becomes true.	(a 0r b) is true
not	Used to reverse the logical state of its operand.	Not (a and b) is false

Bitwise Operators

Operator	Description
& binary and	Operator copies a bit to the result if it exists in both operands
| binary or	It copies a bit if it exists in either operand
^ binary xor	It copies the bit if it is set in one operand but not both
~ binary ones complement	It is unary and has the effect of 'flipping' bits
<< binary Left Shift	left operands value is moved left by the number of bits specified by the right operand
>> binary Right Shift	left operands value is moved right by the number of bits specified by the right operand

```
>>> a=10 # 10 =1010
>>> b=12 #  12=1100
>>> c=(a&b)
>>> c
8
>>> bin(c)
'0b1000'
>>> d=(a|b)
>>> d
14
>>> bin(d)
'0b1110'
>>>
>>>
>>> ~a
-11
>>> bin(-11)
'-0b1011'
>>>
>>> # binary left shift
>>> a<<=2
>>> a
40
>>> bin(40)
'0b101000'
>>> a>>=2
>>> a
10
>>> a>>=2
>>> a
2
>>> bin(2)
'0b10'
```

Membership Operators

Operator	Description
in	True if it finds a variable in the specified sequence and false otherwise
not in	True if it does not finds a variable in the specified sequence and false otherwise

```
>>> # Membership operators
>>> x=1
>>> y=[1,2,3,4,5,6]
>>> x in y
True
>>> x=7
>>> x in y
False
>>> x not in y
True
>>>
```

Identity Operators

Operator	Description
is	True if it finds a variable in the specified sequence and false otherwise
is not	True if it does not finds a variable in the specified sequence and false otherwise

```
>>> # identity operator
>>> x=1
>>> y=[1,2,3,4]
>>> x is y
False
>>> x is not y
True
>>> # second example
>>> x=1
>>> y=1
>>> x is y
True
>>> x is not y
False
>>>
```

Number

Different Types

- Python supports four different numerical types

Type	Examples
int	10,100,-786
long	51924368L , -4721584512L
float	-21.9,-90,15.24
complex	3+4j,3e+26j

Number Type Conversion

Type	Function
int(x)	convert x to a plain integer
long(x)	convert x to a long integer
float(x)	convert x to a floating-point number
complex (x)	x to a complex number with real part x and imaginary part zero.
complex(x,y)	convert x and y to a complex number with real part x and imaginary part y

```
>>> # example of number type conversion
>>> x=10.0007
>>> int(x) # convert into int
10
>>> long(x) # convert into long
10L
>>> float(x) # convert to a floating point
10.0007
>>> x=10
>>> float(x) # convert to a floating point
10.0
>>> complex(x)
(10+0j)
>>> x=10
>>> y=4
>>> complex(x,y) # convert into x+yj form
(10+4j)
>>>
```

Mathematical Function

Function	Description
abs(x)	The absolute value of x
cmp(x,y)	-1 if x < y, 0 if x == y, or 1 if x > y
max(x1,x2,..)	The largest of its arguments
min(x1,x2,..)	The smallest of its arguments
pow(x,y)	The value of x**y
round(x)	**x** rounded
sqrt(x)	The square root of x for x > 0

```
>>> # mathematical function
>>> x=7.00
>>> abs(x)
7.0
>>> x=-7.40 # negative value
>>> abs(x)
7.4
>>> x,y=7,6
>>> cmp(x,y)
1
>>> x,y=6,7
>>> cmp(x,y)
-1
>>> x,y=7,7
>>> cmp(x,y)
0
>>> max(2,34,567,34,55,90)
567
>>> min(2,34,567,34,55,90)
2
>>> pow(x,y)
823543
>>> x=7.7
>>> round(x)
8.0
>>> x=7.2
>>> round(x)
7.0
>>>
```

Trigonometric Function

Function	Description
Sin(x)	Return the sine of x, in radians
Cos(x)	Return the cosine of x, in radians
Tan(x)	Return the tangent of x, in radians

```
>>> # trigonometric function
>>> # want to find sin 30?
>>> sin(30)

Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    sin(30)
NameError: name 'sin' is not defined
>>> import math
>>> x=math.sin(math.radians(30))
>>> x
0.49999999999999994
>>>
>>> # same for cos and tan
>>> x=math.cos(math.radians(30))
>>> x
0.8660254037844387
>>> x=math.tan(math.radians(30))
>>> x
0.5773502691896257
>>>
```

Mathematical Constant

Constants	Description
pi	The mathematical constant pi.
e	The mathematical constant e

```
>>> # constatnt
>>> # how to use pi in the python
>>> pi

Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    pi
NameError: name 'pi' is not defined
>>> math.pi

Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    math.pi
NameError: name 'math' is not defined
>>> import math
>>> math.pi
3.141592653589793
>>> math.e
2.718281828459045
>>>
```

String

String

- Strings are amongst the most popular types in Python.
- We can create them simply by enclosing characters in quotes.
- Python treats single quotes the same as double quotes.

```
>>> # string example
>>>
>>> print('i am jack') # string in single quotes
i am jack
>>>
>>> print("i am jack sparrow")  #string in double quotes
i am jack sparrow
>>>
```

Values in String

- Python does not support a character type; these are treated as strings of length one, also considered a substring.
- To access substrings, use the square brackets

```
>>> # assign value in string
>>>
>>> a="i am jack"
>>>
>>> # access substring use square brackets
>>>
>>> print "a[0] is = ",a[0]
a[0] is = i
>>> print "a[1] is = ",a[1]
a[1] is =
>>> print "a[2] is = ",a[2]
a[2] is = a
>>> print "a[3] is = ",a[3]
a[3] is = m
>>>
>>> # in range
>>>
>>> print "a[4:8] is  : ",a[4:8]
a[4:8] is  :  jac
>>>
```

Updating String

- You can "update" an existing string by (re)assigning a variable to another string.

```
>>> # updating string
>>>
>>> a="i am jack"
>>> print "old string is : ",a
old string is :  i am jack
>>>
>>> #want to add sparrow in the string
>>>
>>> print"update string is : ",a[:9]+"sparrow"
update string is :  i am jacksparrow
>>>
```

Escape Characters

Backslash notation	Description
\n	For a new line
\t	For a tab (space)

```
>>> # backslash example
>>>
>>> print "i am captain jack sparrow"
i am captain jack sparrow
>>>
>>> # use \n for a new line
>>>
>>> print "i am captain \n jack sparrow"
i am captain
 jack sparrow
>>>
>>>
>>> # use twice \n than....
>>> print "i am captain \n\n jack sparrow"
i am captain


 jack sparrow
>>>
>>>
>>> # use a \t for a space b/w
>>> print "i am captain \t jack sparrow"
i am captain          jack sparrow
>>>
```

String Special Operator

Operator	Description
+	Adds values on either side of the operator
*	Multiple copies of the same string
[]	Gives the character from the given index
[:]	Gives the characters from the given range
In	Returns true if a character exists in the given string
not in	Returns true if a character does not exist in the given string

```
>>> # string special operator
>>> a="hello" # assign first string
>>> b="black pearl" # assign second string
>>>
>>> a+b # add string
'helloblack pearl'
>>> a*2 # make a copies
'hellohello'
>>> a*7 # make a copies
'hellohellohellohellohellohellohello'
>>> print"a[0] is  =  ",a[0]
a[0] is  =  h
>>> print"a[0:4] is  =  ",a[0:4]
a[0:4] is  =  hell
>>>
>>> # how to use in in the string
>>> h in a   # is it working?????

Traceback (most recent call last):
  File "<pyshell#11>", line 1, in <module>
    h in a   # is it working?????
NameError: name 'h' is not defined
>>> a=['h','l','l','o']
>>> h in a # its works

Traceback (most recent call last):
  File "<pyshell#13>", line 1, in <module>
    h in a # its works
NameError: name 'h' is not defined
>>> 'h' in a # its works
True
>>> 'h' not in a # its works
False
```

String Formatting Operator

Format symbol	conversion
%c	character
%i	signed decimal integer
%d	signed decimal integer
%u	unsigned decimal integer
%o	octal integer
%x	hexadecimal integer
%e	exponential notation
%f	floating point real number

```
>>> # string formating operator
>>> print ("this is a character %c")%('a')
this is a character a
>>> print ("this is number %i")%(7)
this is number 7
>>> print ("this is number %d")%(7)
this is number 7
>>> print ("this is number %o")%(7) # for an octal integer
this is number 7
>>> print ("this is number %o")%(24) # for an octal integer
this is number 30
>>> print ("this is number %o")%(9) # for an octal integer
this is number 11
>>> print ("this is number %x")%(9) # for an hexadecimal
this is number 9
>>> print ("this is number %x")%(11) # for an hexadecimal
this is number b
>>> print ("this is number %d")%(7.70008) # for an decimal
this is number 7
>>> print ("this is number %f")%(7.70008) # for a floating
this is number 7.700080
>>>
```

Triple Quotes

- Python's triple quotes comes to the rescue by allowing strings to span multiple lines
- No need of the \backslash n, for the new line

```
>>> # use triple quotes
>>>
>>> print(""" this is  long  sentence
                        use new line without using backslash
n""")
 this is    long  sentence
                        use new line without using backslash
n
>>>
```

Built In String Method

1. **capitalize ()**

•It returns a copy of the string with only its first character capitalized.

```
>>> # capitalize example
>>>
>>> # step 1: enter a sting
>>>
>>> str="i am looking for a black pearl"
>>>
>>> # step 2 : use inbuilt function of capitalize
>>>
>>> print"capitalize is =  ",str.capitalize()
capitalize is =   I am looking for a black pearl
>>>
>>> # i becomes I "first character is capitalize"
```

Built In String Method

2. **center(width, fillchar)**

width -- This is the total width of the string.

fillchar -- This is the filler character

- This method returns centered in a string of length width

```
>>> # center(width,fillchar) example
>>>
>>> #step 1: enter a string
>>> str="i am looking for a black pearl"
>>>
>>> # step 2: use sting function
>>>
>>> print"center (80,'z') is = ",str.center(80,'z')
center (80,'z') is =  zzzzzzzzzzzzzzzzzzzzzzzzzi am looking for a black
 pearlzzzzzzzzzzzzzzzzzzzzzzzzz
>>>
```

Built In String Method

3. **count(sub, start= 0,end=len(string))**

sub -- This is the substring to be searched.

start -- Search starts from this index. First character starts from 0 index. By default search starts from 0 index.

end -- Search ends from this index. First character starts from 0 index. By default search ends at the last index.

```
>>> # string count example
>>>
>>> # step 1: enter a string
>>> str="i am looking for a my ship "
>>>
>>> #step 2 : use string operation
>>> sub="o"
>>> print"str.count is =  ",str.count(sub[,0[,len(str)]])
SyntaxError: invalid syntax
>>> print"str.count is =  ",str.count(sub,0,50)
str.count is =  3
>>>
>>> # tab method not working use direct(sub,start,end)
>>>
```

Built In String Method

4. **String encode and decode**

encode(encoding='UTF-8',errors='strict')

decode(encoding='UTF-8',errors='strict')

Ref:

https://docs.python.org/3/library/codecs.html#standard-encodings

```
>>> # encode and decode use...for string
>>> str="i am captain jack sparrow"
>>> str=str.encode('base64','strict')
>>>
>>> # print encode
>>> print"encode string is = " +str
encode string is = aSBhbSBjYXB0YWluIGphY2sgc3BhcnJvdw==

>>> # print decode
>>> print"decode string is = " +str.decode('base64','strict')
decode string is = i am captain jack sparrow
>>>
>>> #use another encoding scheme
>>> str="i am captain jack sparrow"
>>> str=str.encode('hex','strict')
>>> # print encode
>>> print"encode string is = " +str
encode string is = 6920616d2063617074616e206a61636b2073706172
726f77
>>>
>>> # print decode
>>> print"decode string is = " +str.decode('hex','strict')
decode string is = i am captain jack sparrow
>>>
```

Built In String Method

5. **String endswith ()**

•It returns True if the string ends with the specified *suffix*, otherwise return False

suffix -- This could be a string or could also a tuple of suffixes to look for.

start -- The slice begins from here.

end -- The slice ends here.

```
>>> # string endwith example
>>> str="pirate looking for treasure"
>>> suffix="treasure"
>>> print str.endswith(suffix)
True
>>> # lets change suffix
>>> suffix="pirate"
>>> print str.endswith(suffix)
False
>>>
>>> # add start and end
>>> str="pirate looking for treasure"
>>> suffix="treasure"
>>> print str.endswith(suffix,19)
True
>>> print str.endswith(suffix,15)
True
>>> print str.endswith(suffix,5)
True
>>> print str.endswith(suffix,5,9)
False
>>> print str.endswith(suffix,0)  #end not given
True
>>>
```

Built In String Method

6.**String find**

•Determine if str occurs in string or in a substring of string if starting index beg and ending index end are given returns index if found and -1 otherwise.

str.find(str, beg=0, end=len(string))

```
>>> # string find example
>>>
>>> # step1 : enter string
>>> str="i am jack"
>>> #step 2: enter what you want to find in string
>>> str2="jack"
>>>
>>> # use string find function
>>> print str.find(str2)
5
>>> #its gives its postion
>>> #take other example
>>> str2="am"
>>> print str.find(str2)
2
>>> str2="pirate"
>>> print str.find(str2)
-1
>>> # -1 indicate ....not in first string
>>>
```

Built In String Method

7. **isalnum()**

Returns true if string has at least 1 character and all characters are alphanumeric and false otherwise.

8. **isalpha()**

Returns true if string has at least 1 character and all characters are alphabetic and false otherwise

9. **isdigit()**

Returns true if string contains only digits and false otherwise

```
>>> # isalnum example
>>> str="this is 2009"
>>> print str.isalnum()
False
>>> str="thisis2009" # with no space
>>> print str.isalnum()
True
>>> str="2009" # only numbers
>>> print str.isalnum()
True
>>> # take a example of isalpha
>>> str="hi jack"
>>> print str.isalpha()
False
>>> str="hijack" #no space
>>> print str.isalpha()
True
>>> #take a example of isdigit
>>> str="124546"
>>> print str.isdigit()
True
>>> str="hijack"
>>> print str.isdigit()
False
>>>
```

Built In String Method

10. **islower()**

Returns true if string has at least 1 cased character and all cased characters are in lowercase and false otherwise.

11. **isspace()**

Returns true if string contains only whitespace characters and false otherwise

12.i**stitle()**

Returns true if string is properly "titlecased" and false otherwise

```
>>> # islower example
>>> str="Hl,Jack"
>>> print str.islower()
False
>>> str="hi,jack"
>>> print str.islower()
True
>>> str="        "
>>> print str.isspace() # chk for space in string
True
>>> str="hi,jack"
>>> print str.isspace() # chk for space in string
False
>>>
>>> #istitle example
>>> str="hi,jack"
>>> print str.istitle()
False
>>> str="Hl JACK"
>>> print str.istitle()
False
>>> str="Hi Jack"
>>> print str.istitle()
True
>>> # only first character is capital.....
```

Built In String Method

13. **join(seq).**

Merges (concatenates) the string representations of elements in sequence seq into a string, with separator string

14. **len(string)**

Returns the length of the string

15.**lstrip()**

Removes all leading whitespace in string

```
>>> # join sequence
>>> s=" " # by space add.....
>>> seq=("hi","i am","jack")
>>> print s.join(seq)
hi i am jack
>>>
>>> # find length of the string
>>> str="i am jack"
>>> print "length of the string is = ",len(str)
length of the string is =  9
>>>
>>> #lstrip operation
>>> str="      hi  iam jack"
>>> print str.lstrip()
hi  iam jack
>>>
>>> str="!!!!!!!!!!!!!!!hi  iam jack"
>>> print str.lstrip('!')
hi  iam jack
>>>
```

Built In String Method

16. **maketrans()**

Returns a translation table to be used in translate function.

17. **max(str)**

Returns the max alphabetical character from the string str

18. **min(str)**

Returns the min alphabetical character from the string str

```
>>> from string import maketrans # to call maketrans function
>>> exist="aeiou"
>>> replace="12345"
>>> trantab=maketrans(exist,replace)
>>>
>>>
>>> str="this is a simple string "
>>> print str.translate(trantab)
th3s 3s 1 s3mpl2 str3ng
>>>
>>>
>>> # max()
>>> str="this is a simple string "
>>> print "max character is  = " +max(str)
max character is  = t
>>>
>>> #min
>>> str="this is a simple string "
>>> print "min character is  = " +min(str)
min character is  =
>>>
```

Built In String Method

19. **replace(old, new [, max])**

Replaces all occurrences of old in string with new or at most max occurrences if max given.

20.**rindex(str, beg=0, end=len(string))**

rindex() returns the last index where the substring **str** is found

21. **startswith(str, beg=0,end=len(string))**

Determines if string or a substring of string (if starting index beg and ending index end are given) starts with substring str; returns true if so and false otherwise

```
>>> # replace example
>>> str="this is a simple example"
>>> print str.replace("is","was")
thwas was a simple example
>>>
>>> #rindex example
>>> str="this is a simple example"
>>> str1="is"
>>> print str.rindex(str1)
5
>>>
>>> #start with example
>>> str="this is a simple example"
>>> print str.startswith('this')
True
>>> print str.startswith('is')
False
>>> print str.startswith('is',2,4)
True
>>>
```

Built In String Method

22. **swapcase()**

Inverts case for all letters in string

23. **zfill()**

Returns original string leftpadded with zeros to a total of width characters

```
>>> #swapcase example
>>> str="this is a simple string"
>>> print str.swapcase()
THIS IS A SIMPLE STRING
>>>
>>> str="THIS IS A SIMPLE STRING"
>>> print str.swapcase()
this is a simple string
>>>
>>> #zfill example
>>> str="this is a simple string"
>>> print str.zfill(70)
0000000000000000000000000000000000000000000000000this is a simple
string
>>> print str.zfill(7)
this is a simple string
>>> print str.zfill(27)
0000this is a simple string
>>>
```

List

List in python

- The list is a most versatile data type available in Python which can be written as a list of comma-separated values (items) between square brackets.
- Important thing about a list is that items in a list need not be of the same type.

```
>>> # pyhton list example
>>> list=['hellooo','kung-fu']
>>> print 'list is =',list[]
SyntaxError: invalid syntax
>>> print 'list is =',list()
list is =

Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    print 'list is =',list()
TypeError: 'list' object is not callable
>>> print 'list is =',list[:]
list is = ['hellooo', 'kung-fu']
>>> list=['hellooo','kung-fu',2017]
>>> print 'list is =',list[:]
list is = ['hellooo', 'kung-fu', 2017]
>>>
>>> # integer + character....
>>>
```

Assign Value in List

- To access values in lists, use the square brackets for slicing along with the index

```
>>> # values in lists.....
>>>
>>> #make a list using list function
>>> list=['i','love','kung-fu',2017]
>>>
>>> # want to print first in list.....
>>> print"list [0] is a = ",list[0]
list [0] is a =  i
>>> # want to print second in list.....
>>> print"list [1] is a = ",list[1]
list [1] is a =  love
>>>
>>> #want to print in a range.....love to 2017
>>> print"list [1:3] is a = ",list[1:3]
list [1:3] is a =  ['love', 'kung-fu']
>>> # start with 1 and end with before 3....
>>>
>>> print"list [1:4] is a = ",list[1:4]
list [1:4] is a =  ['love', 'kung-fu', 2017]
>>>
>>> #done....
```

Updating List

- You can update single or multiple elements of lists …

```
>>> # update list in python
>>> # step 1: assign a list using list function.....
>>> list=['i','love'.'kung-fu'.2017,007]
SyntaxError: invalid syntax
>>> list=['i','love','kung-fu',2017,007]   # seprate by commaa
>>> print list[2]
kung-fu
>>>
>>> #want to update...yoga not kungfu
>>>
>>> #assign or update list 2
>>> list[2]='yoga'
>>> print"new updated list is =",list[:]
new updated list is = ['i', 'love', 'yoga', 2017, 7]
>>>
```

Delete List

- To remove a list element, you can use either the del statement if you know exactly which element(s) you want to delete.

```
>>> # delete list element...example
>>> # step-1 assign list
>>> list =['i','love','kung-fu',2017]
>>> print list[:]
['i', 'love', 'kung-fu', 2017]
>>> # want to delete  kung-fu
>>> del list[2]
>>> print "after delete the list is = ",list[:]
after delete the list is =  ['i', 'love', 2017]
>>>
```

Basic List Operation

List
Length
Concatenation
Repetition
Membership
Iteration

```
>>> #basic list operation
>>> #find length of list
>>> len([7,8,9,152,41])
5
>>> #concatenation
>>> [1,2,3]+[4,6,7]
[1, 2, 3, 4, 6, 7]
>>> ['hi']+['jack']
['hi', 'jack']
>>>
>>> #repetition
>>> ['hi wtsaop']*4
['hi wtsaop', 'hi wtsaop', 'hi wtsaop', 'hi wtsaop']
>>>
>>> #membership
>>> 3 in [1,2,3,879]
True
>>> 3 in[2,58,78,33]
False
>>>
>>> #iteration
>>> for x in[1,2,3]: print x

1
2
3
>>>
```

Indexing ,slice & Matrixes

- Because lists are sequences, indexing and slicing work the same way for lists as they do for strings

```
>>> # list indexing,slicing and matrixes example
>>>
>>> # step-1 enter a list
>>> l=['i','am','fan of','Marvel & DC']
>>> l[0]
'i'
>>> l[3]
'Marvel & DC'
>>> l[-1]
'Marvel & DC'
>>> l[-4]
'i'
>>> l[1:]
['am', 'fan of', 'Marvel & DC']
>>> l[-1:]
['Marvel & DC']
>>> l[:-1]
['i', 'am', 'fan of']
>>>
```

Built in Function & Methods

1 **.Cmp(list1,list2)**

The method **cmp()** compares elements of two lists.

```
>>> # compares example
>>> # step-1 enter the two lists
>>> l1=['i','am','jack']
>>> l2=['i','am','jack'] # exact same
>>> cmp(l1,l2)
0
>>> #for the exact same ...return 0
>>> l2=['i','am','jack','sparrow'] # not same but same type
>>> cmp(l1,l2)
-1
>>> # length is different ...return -1
>>>
>>> l2=['i','am','jack','7'] # not same & diff type
>>> cmp(l1,l2)
-1
>>> l2=['i','am','7'] # same length & diff type
>>> cmp(l1,l2)
1
>>> # length same but type different ....return 1
>>>
>>> l2=['i','am','sparrow'] # same length but not exact same
>>> cmp(l1,l2)
-1
>>>
```

Built in Function & Methods

2. **len(list)**

•Gives the total length of the list

3.**max(list)**

•Returns item from the list with max value

4. **min(list)**

•Returns item from the list with min value

```
>>> # built in function & methods for the list....
>>> # length operation
>>> # step-1 enter a string
>>> l1=[123,'i','love','kung-fu']
>>>
>>> print'length of the first string is = ',len(l1)
length of the first string is =  4
>>>
>>>
>>> #find maximum...
>>> l2=['appy','sam','zara']
>>>
>>> print"max value is = ",max(l2)
max value is =  zara
>>> l3=[232,598,99,41015]
>>> print"max value is = ",max(l3)
max value is =  41015
>>>
>>> #find minimum
>>> l2=['appy','sam','zara']
>>> print"min value is = ",min(l2)
min value is =  appy
>>> l3=[232,598,99,41015]
>>> print"min value is = ",min(l3)
min value is =  99
>>>
```

Built in Function & Methods

5. **list(seq)**

•Converts a tuple into list

```
>>> #list sequence example
>>> atuple= (786,'zara','veer')
>>> alist=list(atuple)  # to convert
>>>
>>> print 'list element are = ' ,alist
list element are =  [786, 'zara', 'veer']
>>>
>>> # convert tuples into list
```

Python list methods

1. **list.append(obj)**

- The method **append()** appends a passed *obj* into the existing list

2. **list.count(obj)**

- The method **count()** returns count of how many times *obj* occurs in list

3. **list.extend(seq)**

- The method **extend()** appends the contents of *seq* to list

```
>>> # pyhton list method example
>>> list1=[786,'veer','zara']
>>>
>>> # want to add 2009 in it using function
>>>
>>> list1.append(2009)
>>> print' update list is  = ' ,list1
 update list is  =  [786, 'veer', 'zara', 2009]
>>> list1.append(2009,2017)

Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    list1.append(2009,2017)
TypeError: append() takes exactly one argument (2 given)
>>> # u can't add two obj...
>>>
>>> # count list
>>> list1=[786,'veer','zara',2017,786,2012,'jack']
>>> print 'for a count 786 = ',list1.count(786)
for a count 786 =  2
>>> print 'for a count zara = ',list1.count('zara')
for a count zara =  1
>>>
>>> # extend
>>> l2=[2017,'SRK']  # want to add in the list1
>>> list1.extend(l2)
>>> print ' extend list is = ',list1
 extend list is =  [786, 'veer', 'zara', 2017, 786, 2012, 'jack', 2017, 'SRK']
>>>
>>> # add more than one obj use extend not a append
```

Python list methods

4. **list.index(obj)**

•The method **index()** returns the lowest index in list that *obj* appears

5. **list.insert(index, obj)**

•The method **insert()** inserts object *obj* into list at offset *index*

6. **list.pop(obj=list[-1])**

•The method **pop()** removes and returns last object or *obj* from the list

```
>>> # pyhton list methods example
>>> list1=[786,'veer','zara']
>>> print "index for the veer is = ",list1.index('veer')
index for the veer is =  1
>>> list1=[786,'veer','zara',2017,'veer']
>>> print "index for the veer is = ",list1.index('veer')
index for the veer is =  1
>>>
>>> # its shows lowest index....
>>>
>>> #insert
>>> list1=[786,'veer','zara',2017,'veer']
>>> list1.insert(2012,'jack')
>>>
>>> print "final list is = ",list1
final list is =  [786, 'veer', 'zara', 2017, 'veer', 'jack']
>>> # 2012..mistake.....want to add jack in position 2
>>> list1.insert(2,'jack')
>>> print "final list is = ",list1
final list is =  [786, 'veer', 'jack', 'zara', 2017, 'veer', 'jack']
>>>
>>> #pop example
>>> list1=[786,'veer','zara']
>>> print "list 1  is =  ",list1.pop()
list 1  is =  zara
>>> print "list 1  is =  ",list1
list 1  is =  [786, 'veer']
>>> # remove particular....
>>> list1=[786,'veer','zara']
>>> print "list 1  is =  ",list1.pop(1)
list 1  is =  veer
>>> print "list 1  is =  ",list1
list 1  is =  [786, 'zara']
>>>
```

Python list methods

7. **list.remove(obj)**

•Removes object obj from list

8. **list.reverse()**

•Reverses objects of list in place

9. **list.sort([func])**

•Sorts objects of list, use compare function if given

```
>>> # python list method
>>> # step-1 enter a string
>>> list1=[786,'veer','zara','pirate']
>>>
>>> # want to remove pirate
>>> list1.remove('pirate')
>>> print ' now list is = ',list1
 now list is =  [786, 'veer', 'zara']
>>>
>>> #reverse list
>>> list1=[786,'veer','zara','pirate']
>>> list1.reverse()
>>> print ' now list is = ',list1
 now list is =  ['pirate', 'zara', 'veer', 786]
>>>
>>> #sort
>>> list1=[786,'veer','zara','pirate']
>>> list1.sort()
>>> print ' now list is = ',list1
 now list is =  [786, 'pirate', 'veer', 'zara']
>>>
>>> list2=['a','e','o','u']
>>> list2.sort()
>>> print ' now list is = ',list2
 now list is =  ['a', 'e', 'o', 'u']
>>> list2=['z','a','e','o','u']
>>> list2.sort()
>>> print ' now list is = ',list2
 now list is =  ['a', 'e', 'o', 'u', 'z']
```

Tuples

Tuples

- Tuples are sequences, just like lists.
- The differences between tuples and lists are, the tuples cannot be changed unlike lists
- The tuples use parentheses, whereas lists use square brackets

```
>>> # simple tuples example
>>> tuple1=('dead',' man' ,'tell no tales')
>>> print 'tuple is = ',tuple1
tuple is =  ('dead', ' man', 'tell no tales')
>>>
>>> tuple2=(786,007,586,999)
>>> print 'tuple is = ',tuple2
tuple is =  (786, 7, 586, 999)
>>>
>>> tuple3=(786,'captain','jack',999)
>>> print 'tuple is = ',tuple3
tuple is =  (786, 'captain', 'jack', 999)
>>>
```

Assign Values in Tuples

- To access values in tuple, use the square brackets for slicing along with the index

```
>>> # assign values to tuples
>>> tuple1=('dead man ','tell',' no tales')
>>> print 'tuple[0] is = ',tuple1[0]
tuple[0] is =  dead man
>>> print 'tuple[1] is = ',tuple1[1]
tuple[1] is =  tell
>>> print 'tuple[2] is = ',tuple1[2]
tuple[2] is =   no tales
>>> print 'tuple[:2] is = ',tuple1[:2]
tuple[:2] is =  ('dead man ', 'tell')
>>> # from left(start 0) to ...0 1
>>> print 'tuple[0:] is = ',tuple1[0:]
tuple[0:] is =  ('dead man ', 'tell', ' no tales')
>>> # start from 0 to upto end..
```

Updating Tuples

- Tuples are immutable which means you cannot update or change the values of tuple elements.

```
>>> # update tuples.....
>>> tuple1=(786,999,100,18)
>>> list1=[786,999,100,18]
>>>
>>> #update....
>>> tuple1[0]=777

Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    tuple1[0]=777
TypeError: 'tuple' object does not support item assignment
>>> list1[0]=777
>>> print list1
[777, 999, 100, 18]
>>> tuple1[0]='zara'

Traceback (most recent call last):
  File "<pyshell#8>", line 1, in <module>
    tuple1[0]='zara'
TypeError: 'tuple' object does not support item assignment
>>> list1[0]='zara'
>>> print list1
['zara', 999, 100, 18]
>>>
```

Delete Tuples

- Removing individual tuple elements is n
possible.

```
>>> # want to delete tuple
>>> # taking example with tuple and list
>>> # step 1 : enter tuple and list
>>> tuple1=('dead man tell no tales',' 786' ,'jack sparrow')
>>> list1=['dead man tell no tales',' 786' ,'jack sparrow']
>>>
>>> print tuple1
('dead man tell no tales', ' 786', 'jack sparrow')
>>> del(786)
SyntaxError: can't delete literal
>>> list1.remove(786)

Traceback (most recent call last):
  File "<pyshell#8>", line 1, in <module>
    list1.remove(786)
ValueError: list.remove(x): x not in list
>>> list1=['dead man tell no tales',786 ,'jack sparrow'] # number no need
quotes
>>> list1.remove(786)
>>>
>>> print list1
['dead man tell no tales', 'jack sparrow']
>>>
```

Basic Tuples Operation

Function
Length
Concatenation
Repetition
Membership
iteration

```
>>> # basic tuples operation
>>> # enter tuple
>>> tuple1=(786,999,18,100)
>>> # find length
>>> len(tuple1)
4
>>> #concatenation
>>> tuple1=(786,999,18,100)
>>> tuple2=(986,959,7,100)
>>> tuple1+tuple2
(786, 999, 18, 100, 986, 959, 7, 100)
>>>
>>> #Repetition
>>> tuple1=(786,999,18,100)
>>> tuple1*4
(786, 999, 18, 100, 786, 999, 18, 100, 786, 999, 18, 100, 786, 999, 18, 1
00)
>>>
>>> #Membership
>>> tuple1=(786,999,18,100)
>>> 786 in tuple1
True
>>>
>>> #Iteration
>>> for x in(7,18,100):
        print x


7
18
100
>>>
```

Indexing, Slicing and Matrixes

- Because tuples are sequences, indexing and slicing work the same way for tuples as they do for strings

```
>>> #Indexing, Slicing, and Matrixes
>>>
>>> # enter tuples
>>> tuple1=(786,18,7,100)
>>>
>>> # want to find at zero position
>>> tuple1[0]
786
>>> # in negative index
>>> tuple1=(786,18,7,100)
>>> tuple1[-1]
100
>>>
>>> # in range
>>> tuple1[1:]
(18, 7, 100)
>>> tuple1[:2]
(786, 18)
>>> tuple1[:-1]
(786, 18, 7)
>>> tuple1[:-2]
(786, 18)
>>>
```

Built-in Tuple Function

1. **cmp(tuple1, tuple2)**

Compares elements of both tuples.

2. **len(tuple)**

Gives the total length of the tuple.

```
>>> #built in tuple function
>>> # compares function
>>> tuple1=(786,18,7,100)
>>> tuple2=(786,18,7,100)
>>> print cmp(tuple1,tuple2) # exact same value.....
0
>>> # change tuple2.....
>>> tuple2=(786,7,18,100)
>>> print cmp(tuple1,tuple2) # not exact same value.....
1
>>> # change tuple2.....
>>> tuple2=('veer','zara','jack')
>>> print cmp(tuple1,tuple2) # not exact same types change.....
-1
>>>
>>> # chk for the length of tuple....
>>> tuple1=(786,18,7,100)
>>> tuple2=('veer','zara','jack')
>>> len(tuple1)
4
>>> len(tuple2)
3
>>>
```

Built-in Tuple Function

3. max(tuple)

Returns item from the tuple with max value

4. min(tuple)

Returns item from the tuple with min value

5.tuple(seq)

Converts a list into tuple

```
>>> #built in tuple function
>>> # step1 : enter a tuple
>>> tuple1=(786,18,7,100)
>>> tuple2=('veer','zara','jack')
>>>
>>> # find max in tuple
>>> max(tuple1)
786
>>> max(tuple2)
'zara'
>>> # numeric & alphabatic differnce
>>> tuple2=('veer','zara','jack','zen') # two start with z example
>>> max(tuple2)
'zen'
>>> # e grater than a....
>>>
>>> #find min in tuple
>>> tuple1=(786,18,7,100)
>>> tuple2=('veer','zara','jack')
>>> min(tuple1)
7
>>> min(tuple2)
'jack'
>>> list1=['jack',786]
>>> tuple1=tuple(list1)
>>> print "list converted in tuple = ",tuple1
list converted in tuple =  ('jack', 786)
>>>
```

Dictionary

Dictionary in Python

- Each key is separated from its value by a colon (:)
- The items are separated by commas, and the whole thing is enclosed in curly braces
- Keys are unique within a dictionary while values may not be.
- The values of a dictionary can be of any type, but the keys must be of an immutable data type such as strings, numbers

```
>>> # python dictionary example.....
>>> # step 1 :
>>> dict={'name':'veer', 'age' : 7}
>>>
>>> # keys: name, age
>>> #values : veer, 7
>>>
```

Assign Value

- To access dictionary elements, you can use the familiar square brackets along with the key to obtain its value

```
>>> # accessing values in dictionary
>>> # enter a dictionary
>>> dict={'name':'jack','age': 7}
>>>
>>> print"dict ['name'] is = ",dict['name']
dict ['name'] is =  jack
>>> print"dict ['age'] is = ",dict['age']
dict ['age'] is =  7
>>>
>>> # call by its value,not by key
>>> print"dict ['jack'] is = ",dict['jack']
dict ['jack'] is =

Traceback (most recent call last):
  File "<pyshell#8>", line 1, in <module>
    print"dict ['jack'] is = ",dict['jack']
KeyError: 'jack'
>>>
```

Updating Dictionary

- You can update a dictionary by adding a new entry or a key-value pair, modifying an existing entry, or deleting an existing entry

```
>>> #updating the dictionary
>>> #step-1 enter a dictionary
>>> dict={'name':'veer','age':18,'university':'IIT'}
>>>
>>> # what you want to update
>>> dict['age']=21
>>> dict['university']='GTU'
>>>
>>> print dict
{'university': 'GTU', 'age': 21, 'name': 'veer'}
>>>
>>> dict={'name':'veer','age':18,'university':'IIT'}
>>> # what you want to update
>>> dict['age']=21
>>> dict['university']='GTU'
>>>
>>> # at same?e time both updat
>>> # at same time both update?
>>>
>>> dict['age':22,'university':'NIIT']

Traceback (most recent call last):
  File "<pyshell#18>", line 1, in <module>
    dict['age':22,'university':'NIIT']
TypeError: unhashable type
>>>
>>> # you have to do update one by one.....
```

Deleting Dictionary

- You can either remove individual dictionary elements or clear the entire contents of a dictionary.

- You can also delete entire dictionary in a single operation

```
>>> # delete dictionary
>>> # step-1  enter a dict
>>> dict={'name':'jack','age': 21}
>>>
>>> # want to delete name
>>> del dict['name']
>>> print dict
{'age': 21}
>>>
>>> # to remove all dict
>>> dict.clear()
>>> print dict
{}
>>>
>>> # use delete ,to delete entire dict
>>> dict={'name':'jack','age': 21}
>>> del dect

Traceback (most recent call last):
  File "<pyshell#14>", line 1, in <module>
    del dect
NameError: name 'dect' is not defined
>>> del dict
>>> print dict
<type 'dict'>
>>>
```

Properties of Dictionary keys

(a) More than one entry per key not allowed. Which means no duplicate key is allowed.

When duplicate keys encountered during assignment, the last assignment wins.

```
>>> #properties of dictionary keys
>>> # step 1: enter a dict
>>> dict={'name':'jack','pos.':'Captain','name':'Barbossa'}
>>> print dict
{'pos.': 'Captain', 'name': 'Barbossa'}
>>>
>>> # reason 'no duplicate keys'
>>> dict={'name':'jack','pos.':'Captain','name2':'Barbossa'}
>>> print dict
{'pos.': 'Captain', 'name2': 'Barbossa', 'name': 'jack'}
>>>
>>> # use different key than it print..
>>>
```

Properties of Dictionary keys

(b) Keys must be immutable. Which means you can use strings, numbers or tuples as dictionary keys but something like ['key'] is not allowed.

```
>>> # properties of dictionary keys
>>> # step-1 enter a dict
>>>
>>>  dict={['name']:'jack','age': 21}

  File "<pyshell#3>", line 2
    dict={['name']:'jack','age': 21}
    ^
IndentationError: unexpected indent
>>>
>>> dict={'name':'jack','age': 21}
>>> print dict
{'age': 21, 'name': 'jack'}
>>>
```

Built in Dictionary

1. cmp(dict1,dict2)

   Compares elements of both dict

2. len(dict)

   Gives the total length of the dictionary. This would be equal to the number of items in the dictionary

```
>>> # built in dictionary
>>> # step-1 enter a dict
>>> dict1={'name':'jack','age':21}
>>> dict2={'name':'veer','age':27}
>>> dict3={'name':'zara','age':27 }
>>> dict4={'name':'jack','age':21}
>>>
>>> # start comp
>>> cmp(dict1,dict2)
-1
>>> # dict1 and dict2 totally diff....
>>>
>>> cmp(dict2,dict3)
-1
>>> cmp(dict1,dict4)
0
>>> #dict1 and dict4 is matching....
>>>
>>>
>>> #find length ....
>>> len(dect1)

Traceback (most recent call last):
  File "<pyshell#17>", line 1, in <module>
    len(dect1)
NameError: name 'dect1' is not defined
>>> len(dict1)
2
>>>
```

Built in Dictionary

3. str(dict)

Produces a printable string representation of a dictionary

4. type(variable)

Returns the type of the passed variable. If passed variable is dictionary, then it would return a dictionary type

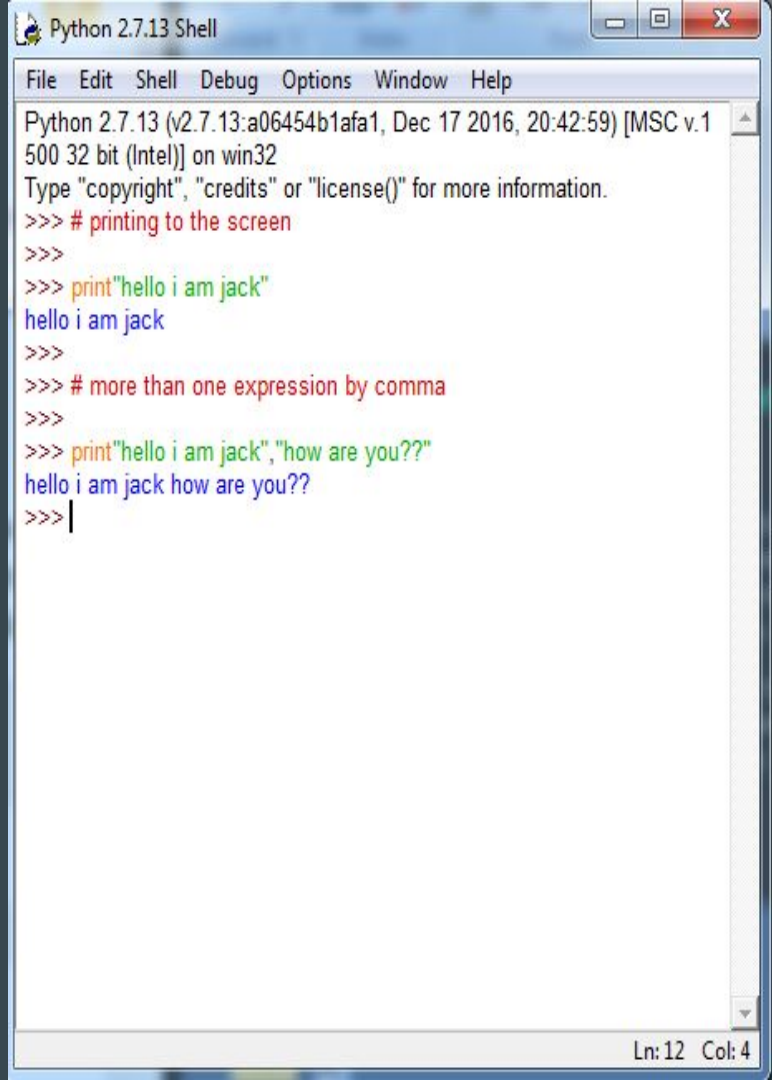
```
>>> # built in dictionary
>>> #step-1 enter a dict
>>> dict={'name':'jack','age':21}
>>>
>>> print'the string id  : %s'%str(dict)
the string id  : {'age': 21, 'name': 'jack'}
>>>
>>> #find a type
>>>
>>> type(dict)
<type 'dict'>
>>>
>>> list=[786,452,12,'jack']
>>> type(list)
<type 'list'>
>>>
>>> # store by other name
>>>
>>> hr=[78,18,7,100]
>>> srk=('hi','wtsap')
>>>
>>> type(hr)
<type 'list'>
>>> type(srk)
<type 'tuple'>
>>>
>>> # thats it.....
```

I/O Files

Printing to the screen

- The simplest way to produce output is using the *print* statement where you can pass zero or more expressions separated by commas.

- This function converts the expressions you pass into a string and writes the result to standard output



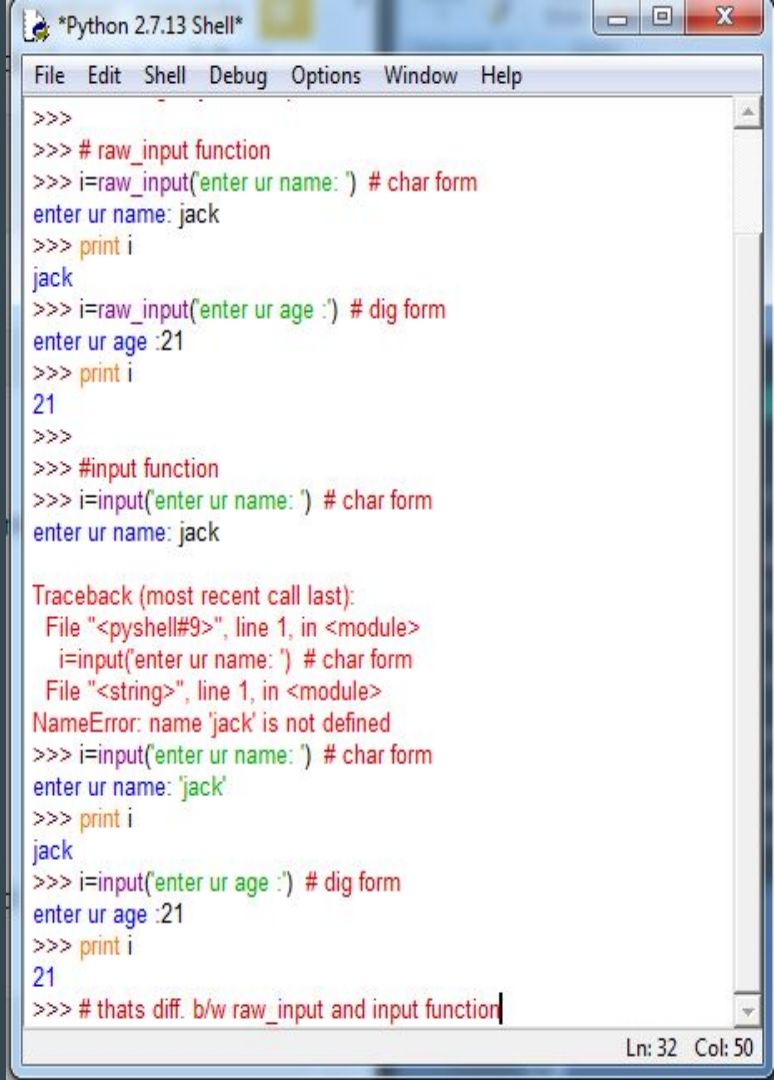
Reading Keyboard Input

a) **raw_____ input Function**

The raw_input([prompt]) function reads one line from standard input and returns it as a string.

b) **input Function**

The input([prompt]) function is equivalent to raw_input, except that it assumes the input is a valid Python expression and returns the evaluated result to you.

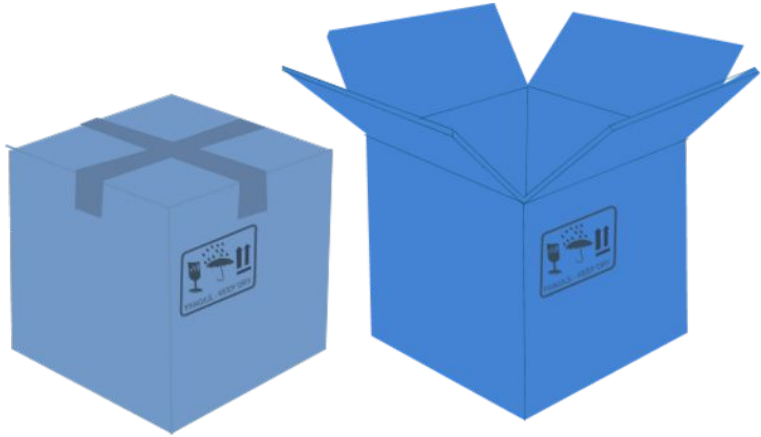


Opening and closing Files

•Until now, you have been reading and writing to the standard input and output. Now, we will see how to use actual data files.

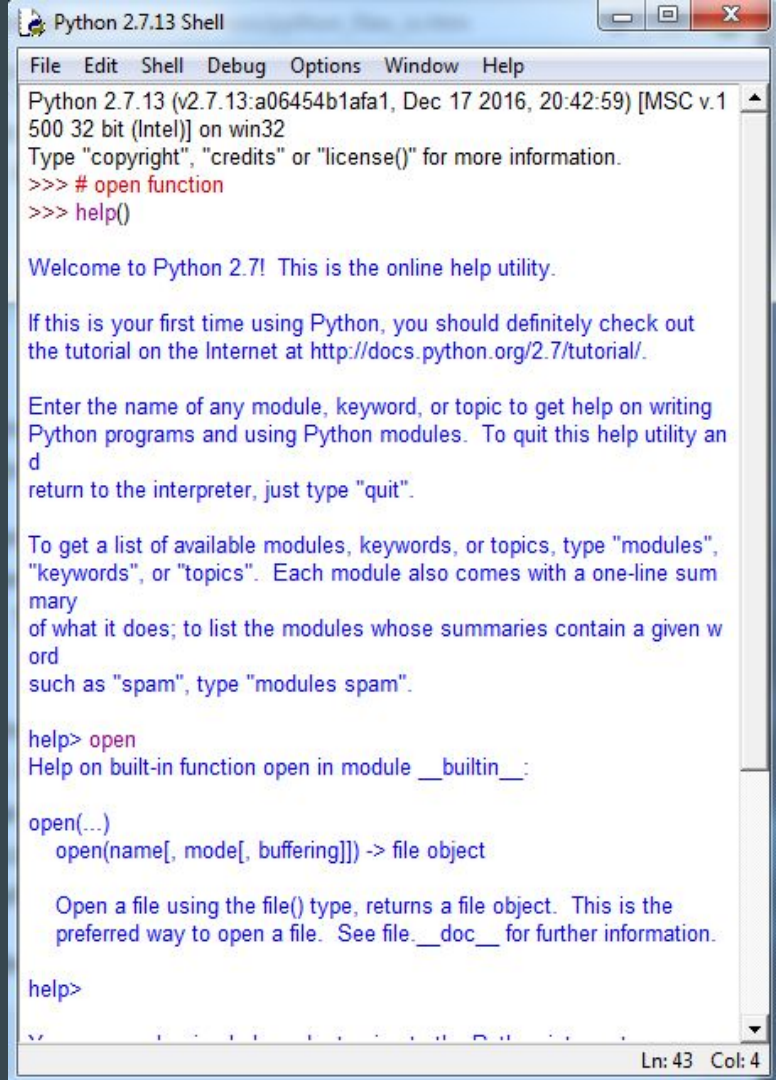
•Python provides basic functions and methods necessary to manipulate files by default.

•You can do most of the file manipulation using a **file** object.



Open Function

- Before you can read or write a file, you have to open it using Python's built-in *open()* function.
- This function creates a **file** object, which would be utilized to call other support methods associated with it.



Open Function

Syntax: file object = open(file_name [, access_mode][, buffering])

Name	Parameter
file_name	Argument is a string value that contains the name of the file that you want to access
access_mode	The access_mode determines the mode in which the file has to be opened, i.e., read, write, append, etc
buffering	If the buffering value is set to 0, no buffering takes place. If the buffering value is 1, line buffering is performed while accessing a file

Open Function

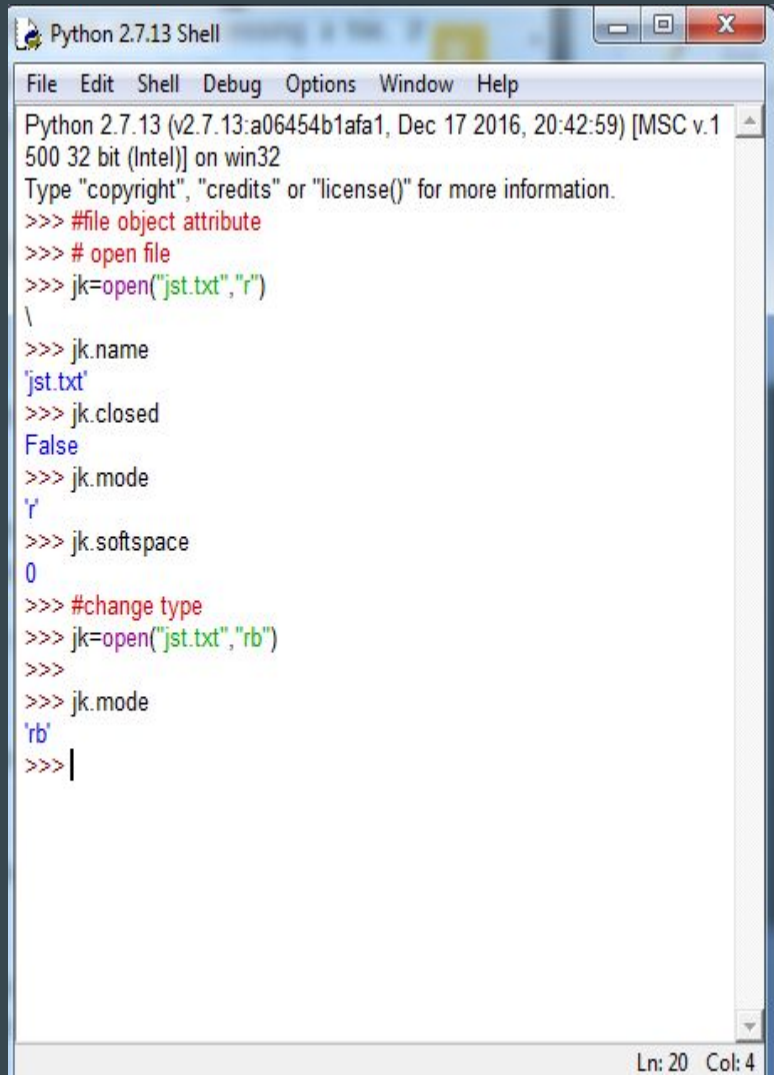
- Here is a list of the different modes of opening a file

Modes	Description	Modes	Description
r	Opens a file for reading only	wb	Opens a file for writing only in binary format
rb	Opens a file for reading only in binary format	w+	Opens a file for both writing and reading
r+	Opens a file for both reading and writing	wb+	Opens a file for both writing and reading in binary format
rb+	Opens a file for both reading and writing in binary format	a	Opens a file for appending.
w	Opens a file for writing only	ab	Opens a file for appending in binary format
ab+	Opens a file for both appending and reading in binary format		

File Object Attributes

- Once a file is opened and you have one *file* object, you can get various information related to that file.

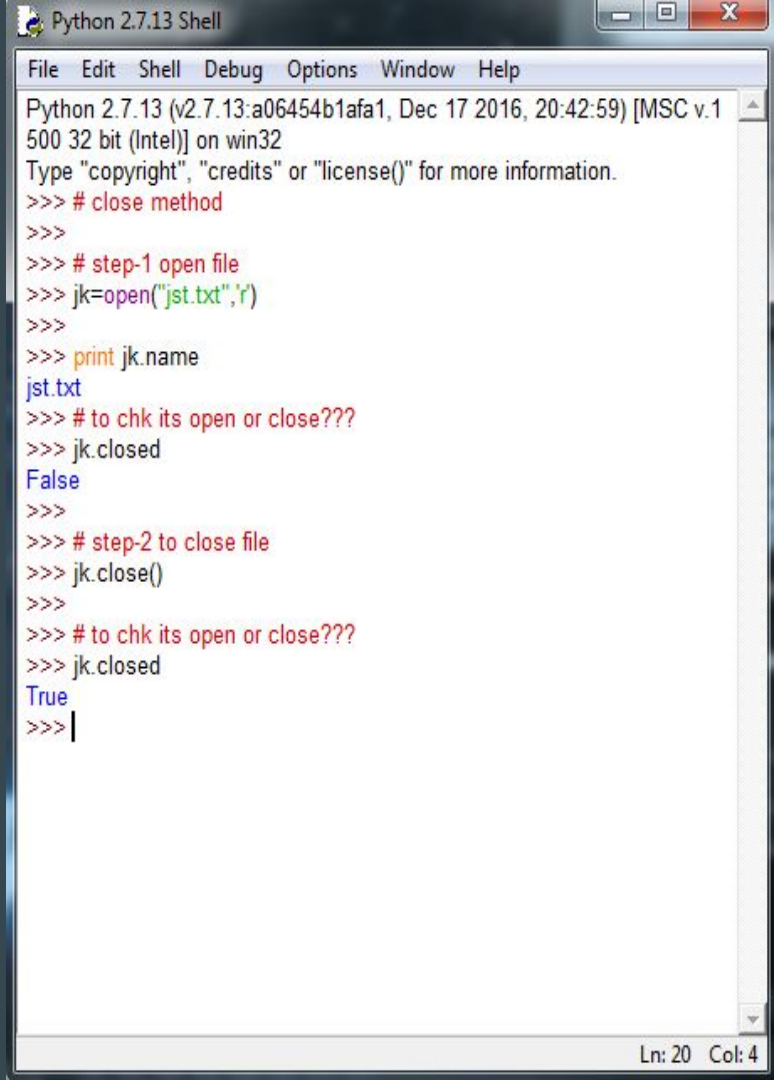
Attribute	Description
file.closed	Returns true if file is closed, false otherwise
file.mode	Returns access mode with which file was opened
file.name	Returns name of the file
file.softspace	Returns false if space explicitly required with print, true otherwise



Close Function

- The close() method of a *file* object flushes any unwritten information and closes the file object, after which no more writing can be done.

•**Syntax**: fileObject.close()



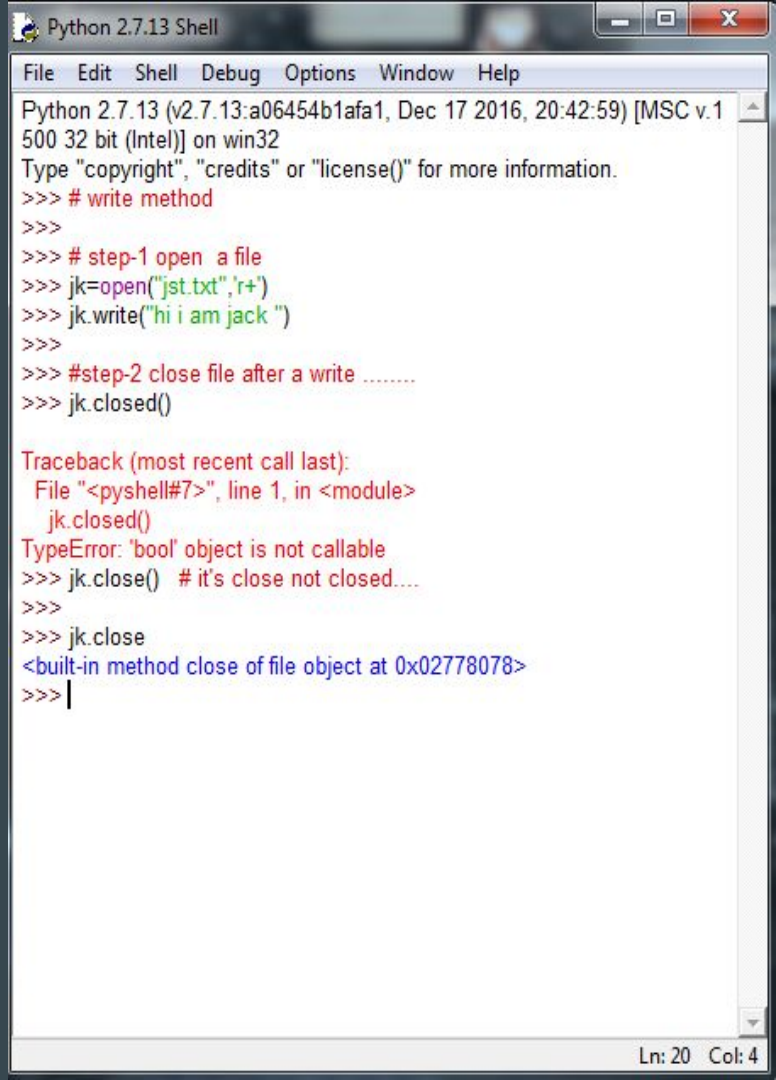
Reading and Writing Files

a) **Write() Method**

The write() method writes any string to an open file.

It is important to note that Python strings can have binary data and not just text.

Syntax: fileObject.write(string)



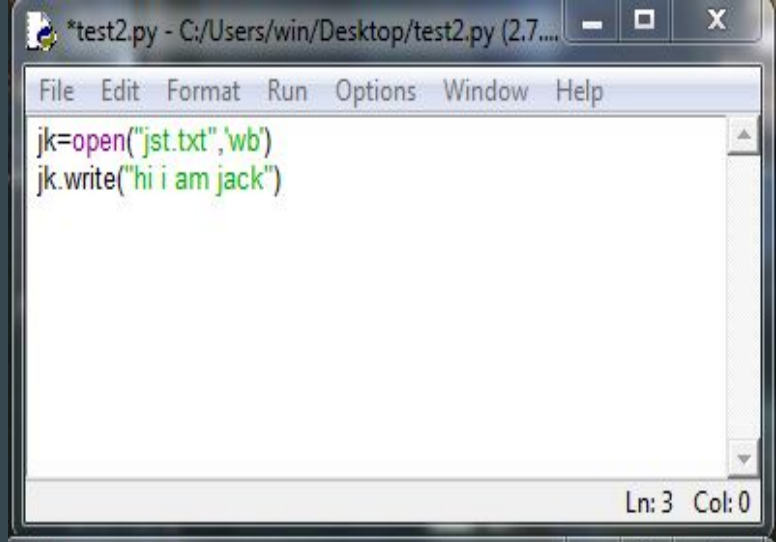
Writing Files

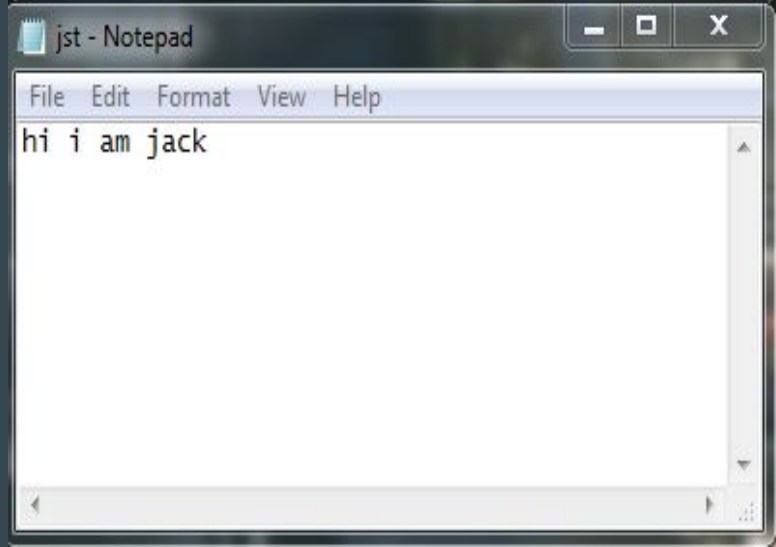
For writing a file,

>>In Python Shell ,you have to open a new file and write a code to open a file and write something in it.

>>After that you have to run a module

>>That will create your jst.txt file on your desktop and you can see a written string which you wrote in python file.



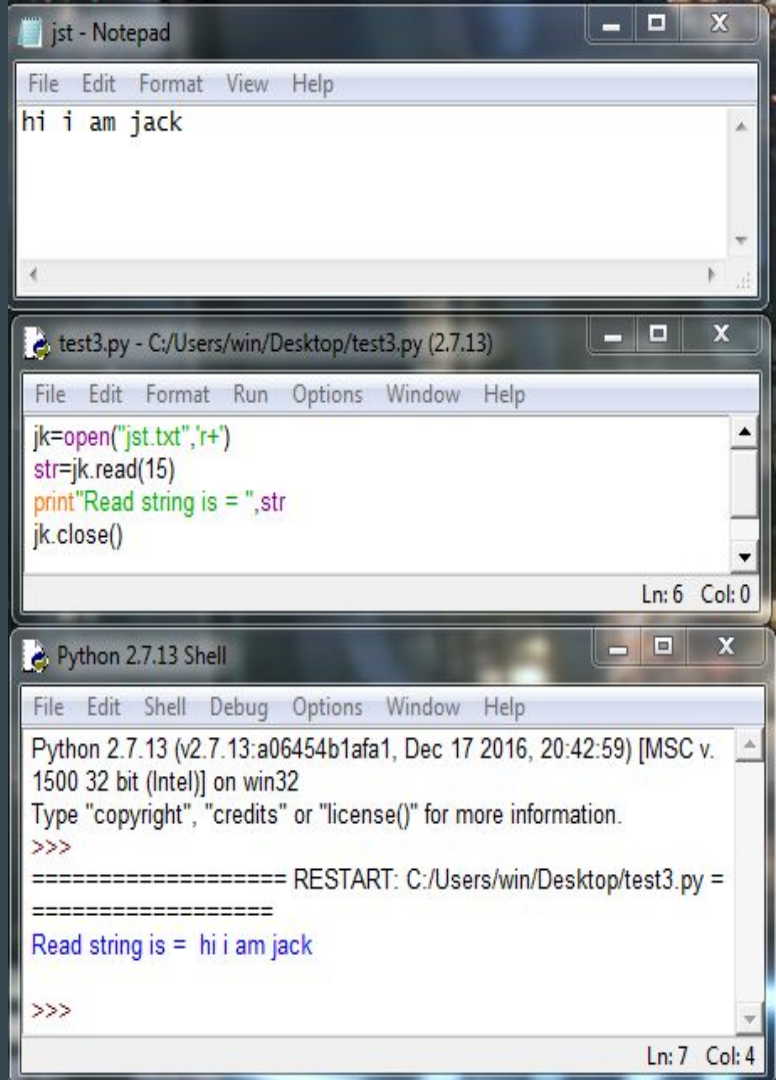


Reading and Writing Files

b) read() Method

The *read()* method reads a string from an open file. It is important to note that Python strings can have binary data. apart from text data.

Syntax:fileObject.read([count])

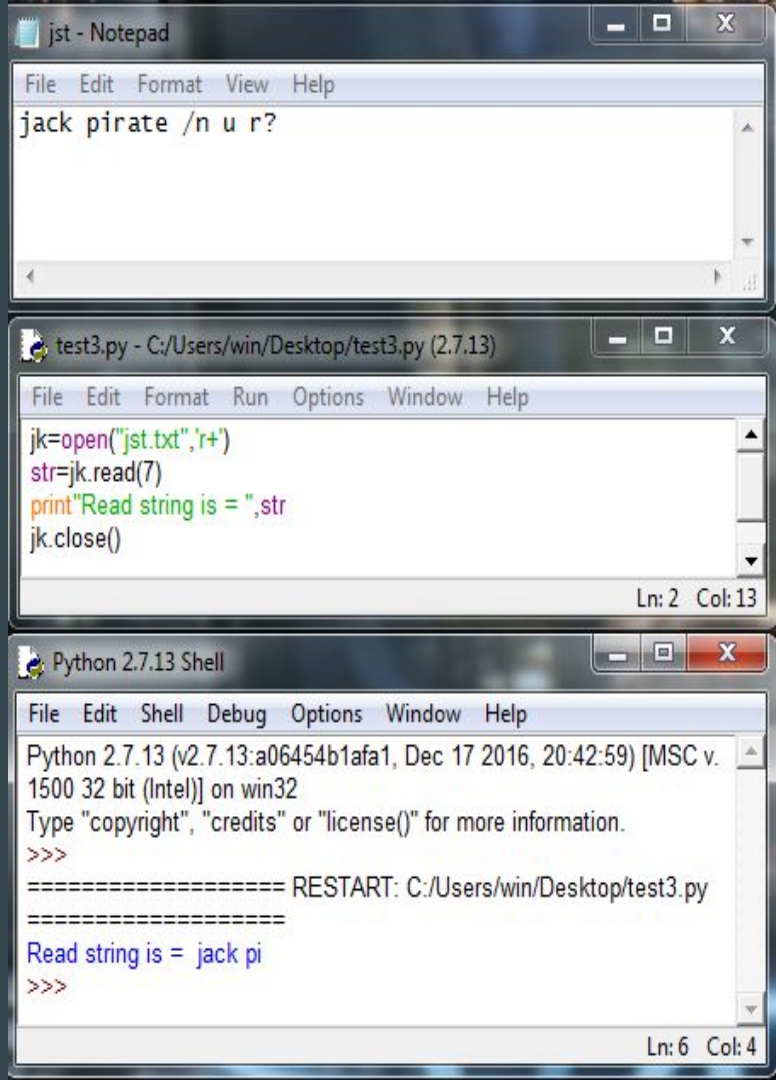


Reading and Writing Files

b) read() Method

Syntax:fileObject.read([count])

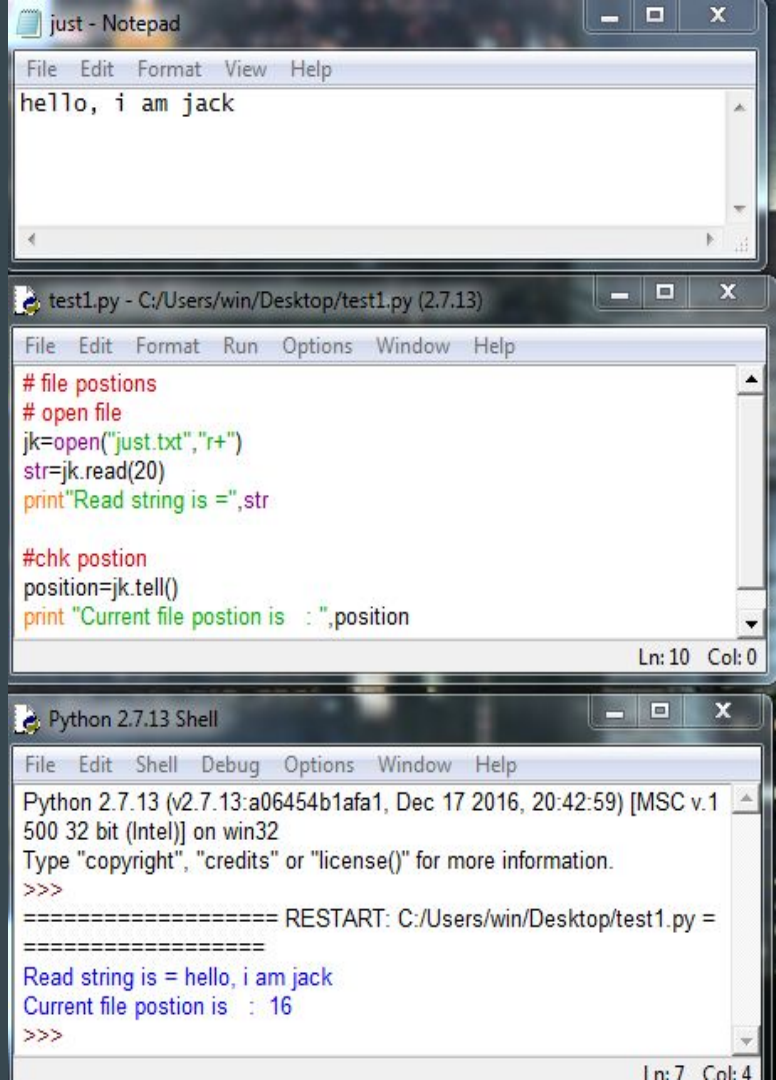
>> with different Count



File Positions

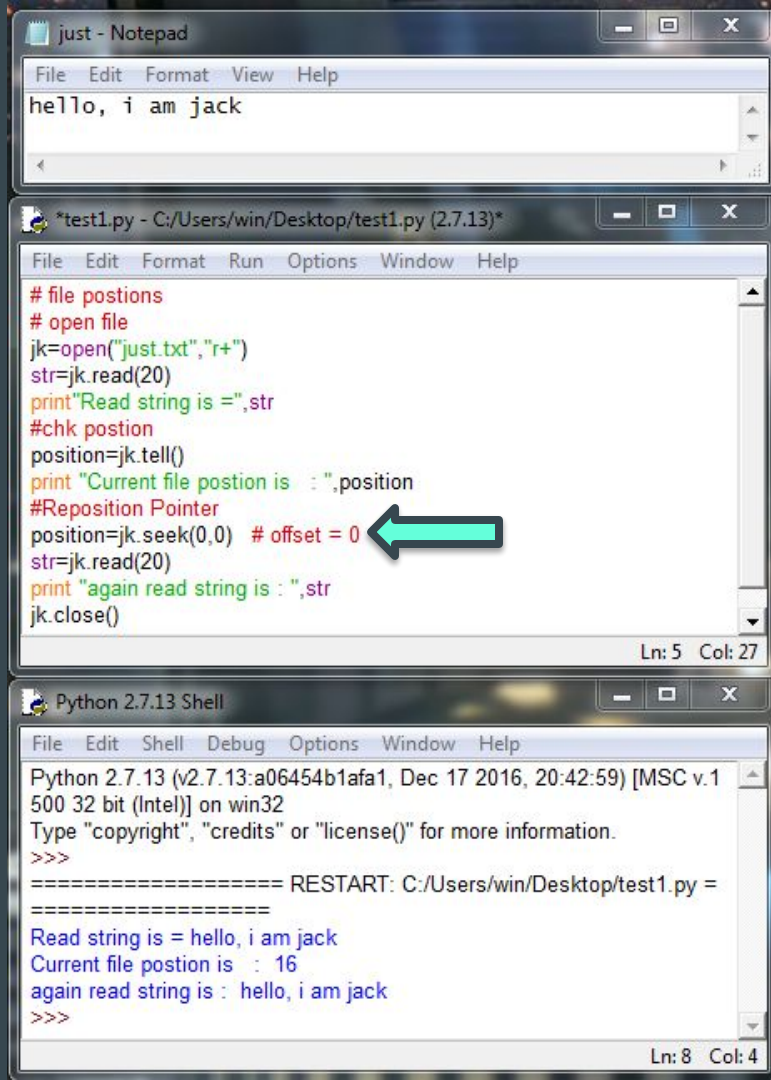
- The tell() method tells you the current position within the file;

- In other words, the next read or write will occur at that many bytes from the beginning of the file.



File Positions

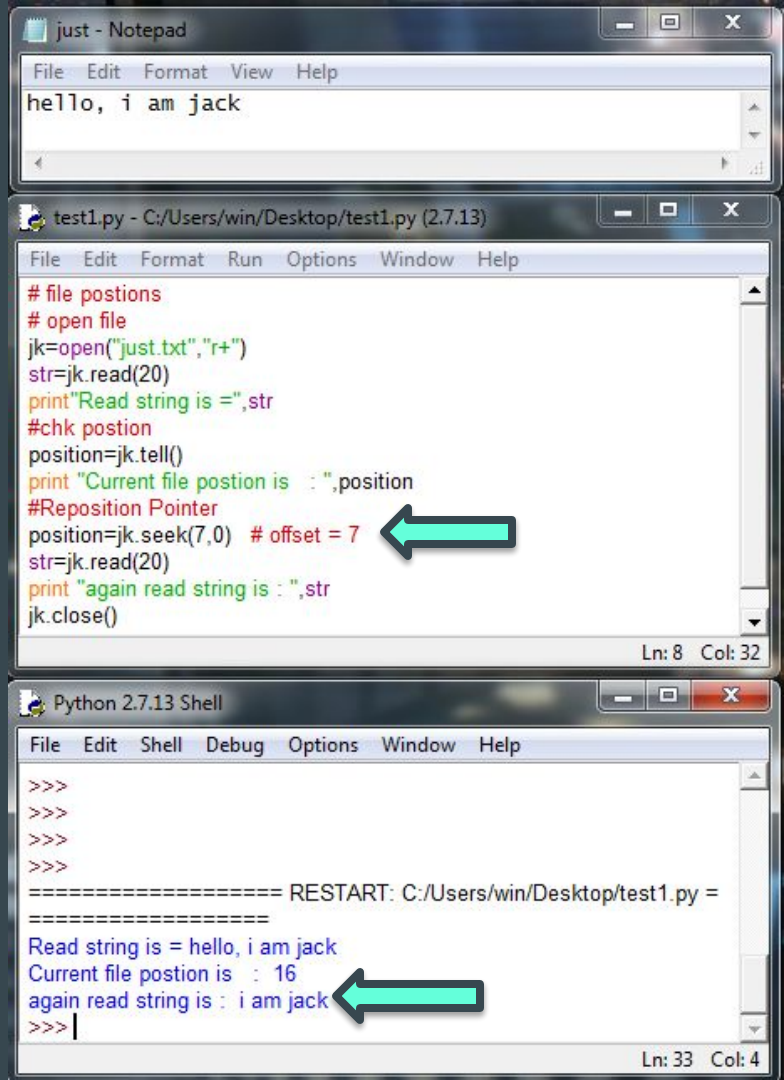
- The *seek(offset[,from])* method changes the current file position.
- The *offset* argument indicates the number of bytes to be moved.
- The *from* argument specifies the reference position from where the bytes are to be moved



File Positions

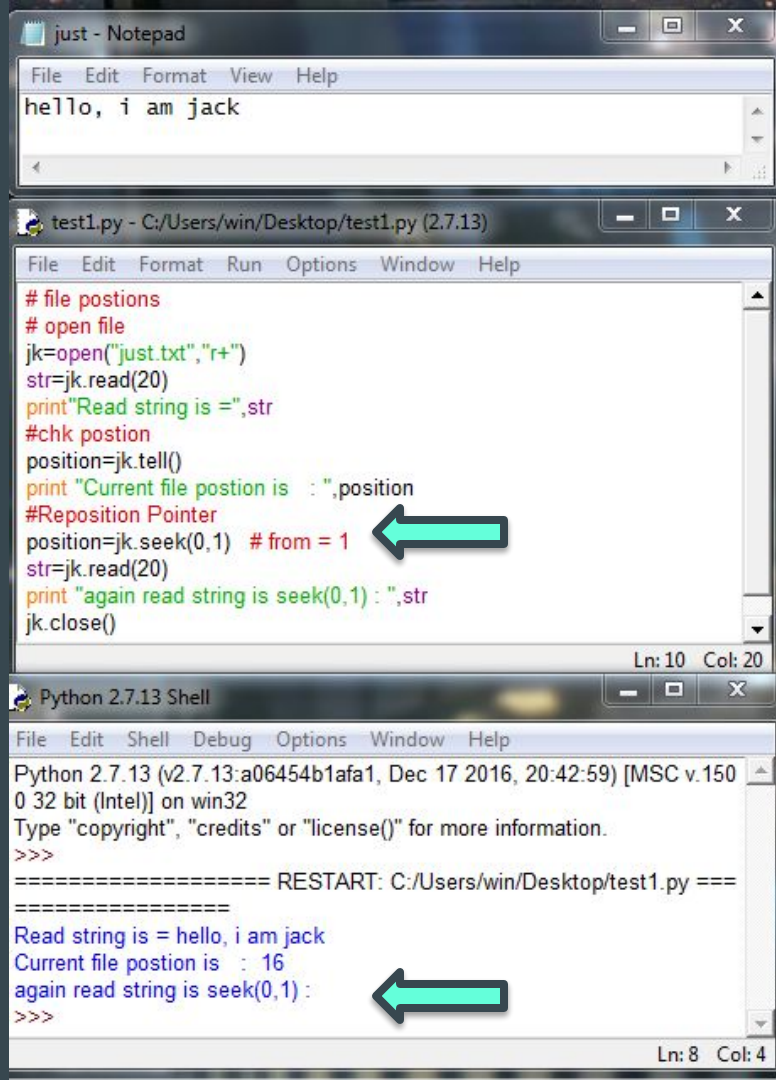
- The *seek(offset[, from])* method changes the current file position

- Offset position Change >>>>



File Positions

- If *from* is set to 0, it means use the beginning of the file as the reference position
- Set to 1 means use the current position as the reference position
- set to 2 then the end of the file would be taken as the reference position.

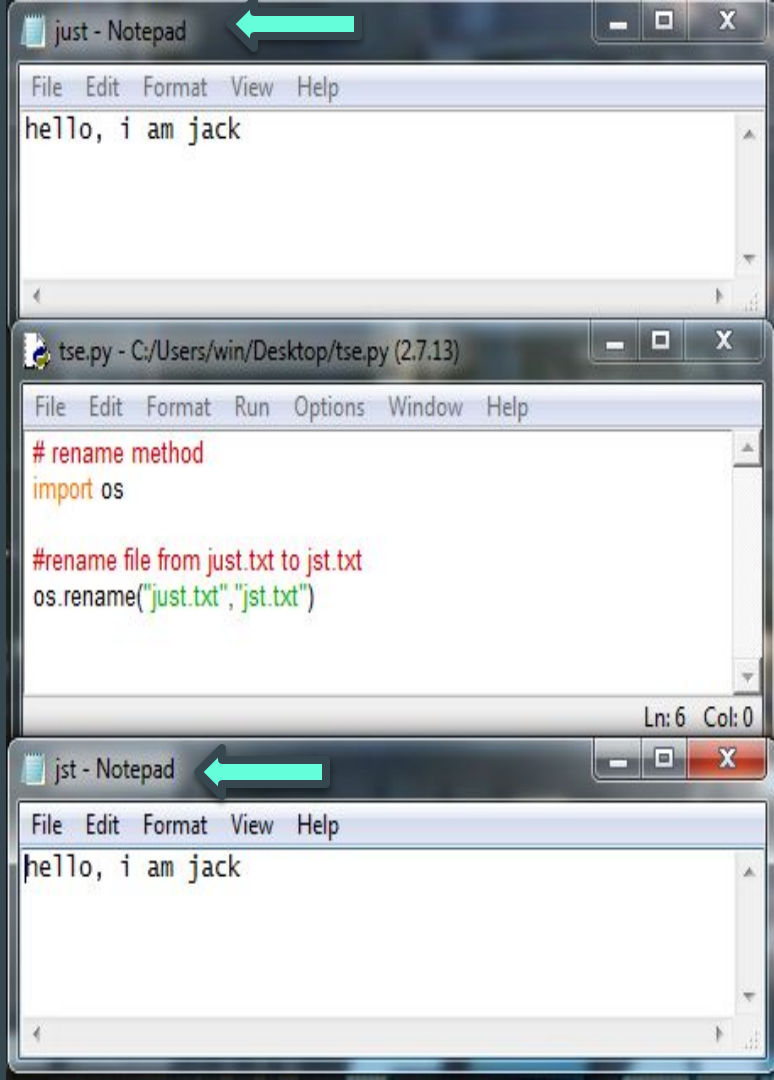


Renaming Files

- The *rename()* method takes two arguments, the current filename and the new filename

- **Syntax**:

os.rename(current_file_name, new_file_name)

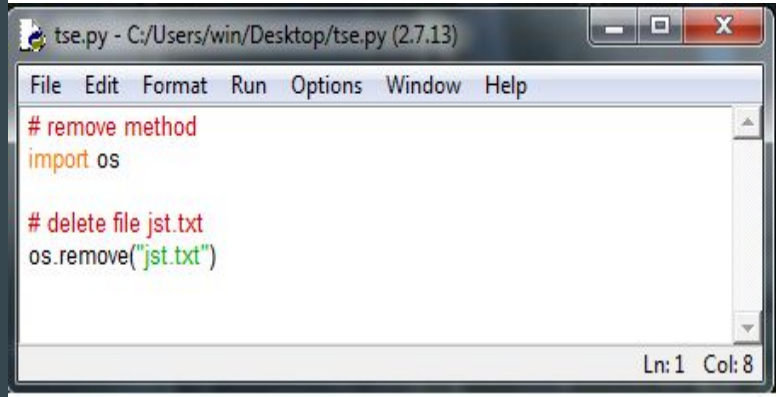


Deleting Files

- You can use the *remove()* method to delete files by supplying the name of the file to be deleted as the argument.

- **Syntax**:

os.remove(file_name)



Directories in Python

- All files are contained within various directories. & Python has no problem handling these directories .

- The **os** module has several methods that help you create, remove, and change directories.

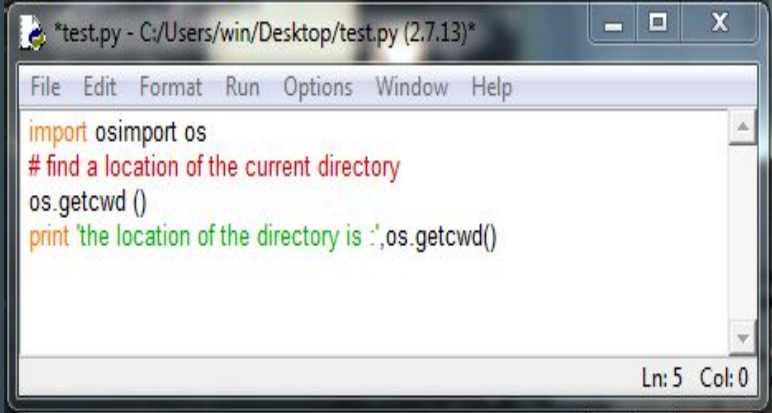
Get current working dict.

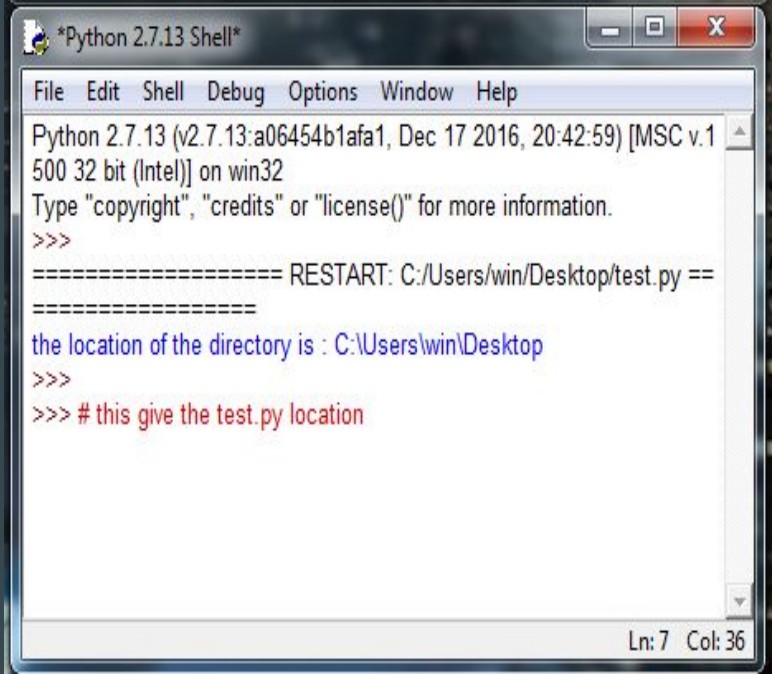
getcwd() :

•The getcwd() method displays the current working directory.

Syntax :

•os.getcwd()





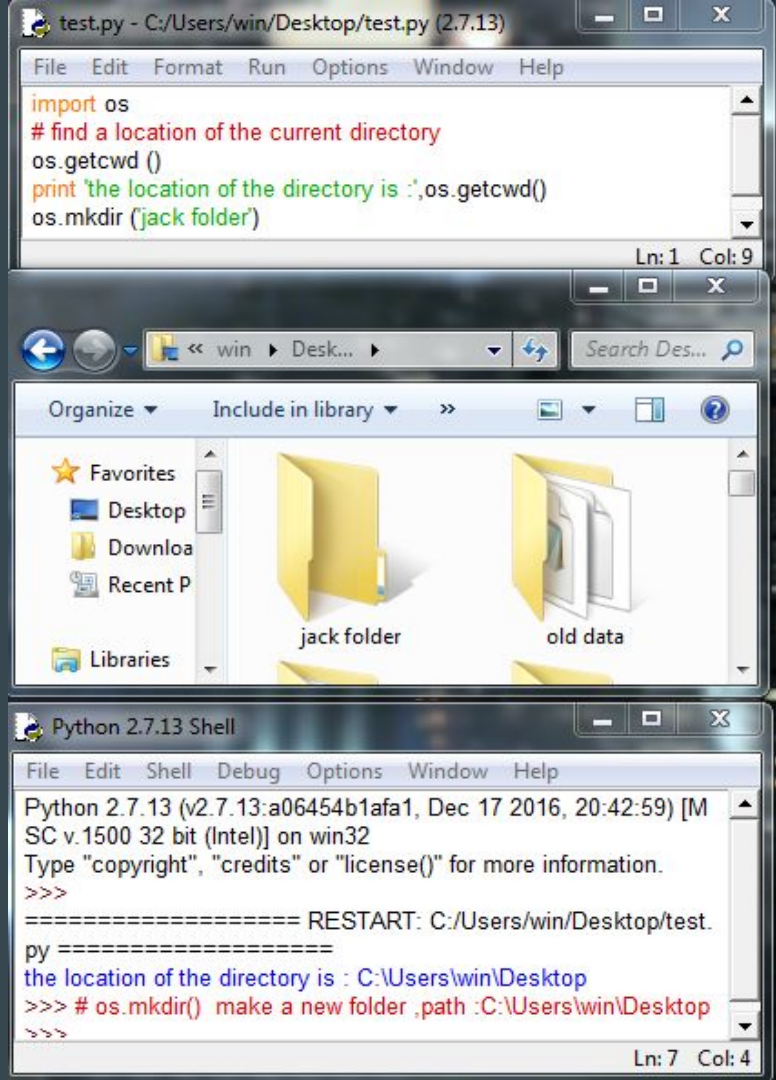
Create a dict.

mkdir() :

•The *mkdir()* method of the **os** module to create directories in the current directory.

Syntax :

•os.mkdir("newdir")



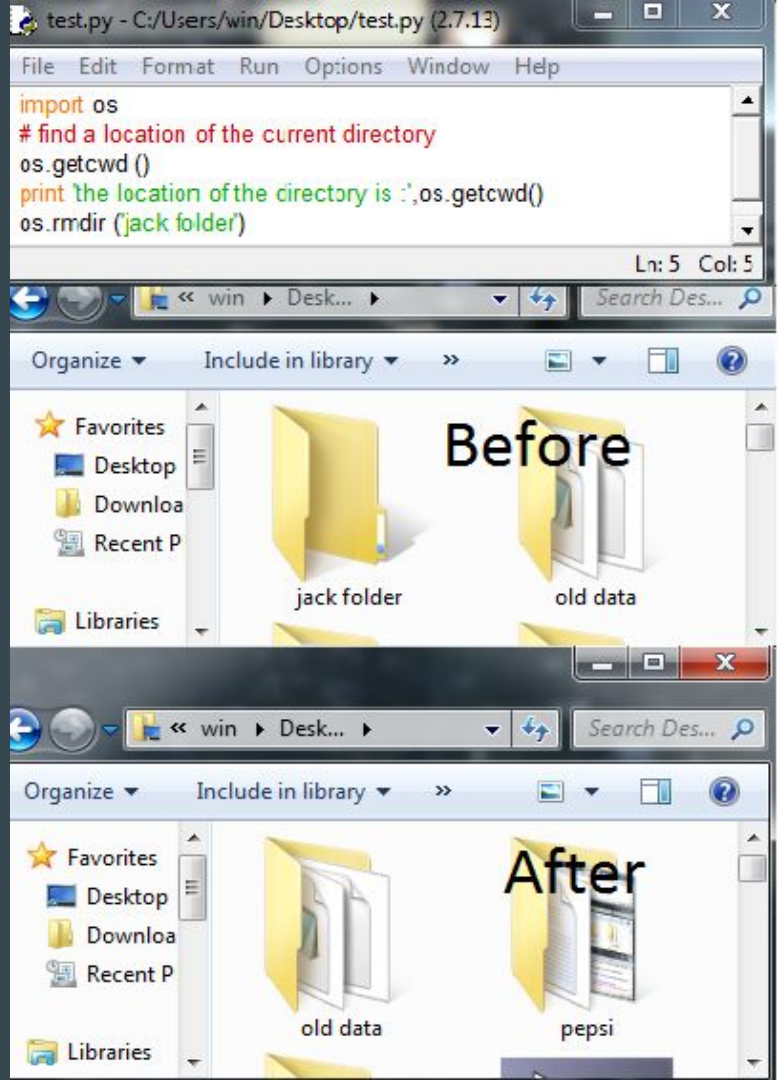
Remove a dict.

rmdir() :

•The rmdir() method deletes the directory, which is passed as an argument in the method.

Syntax :

•os.rmdir('dirname')



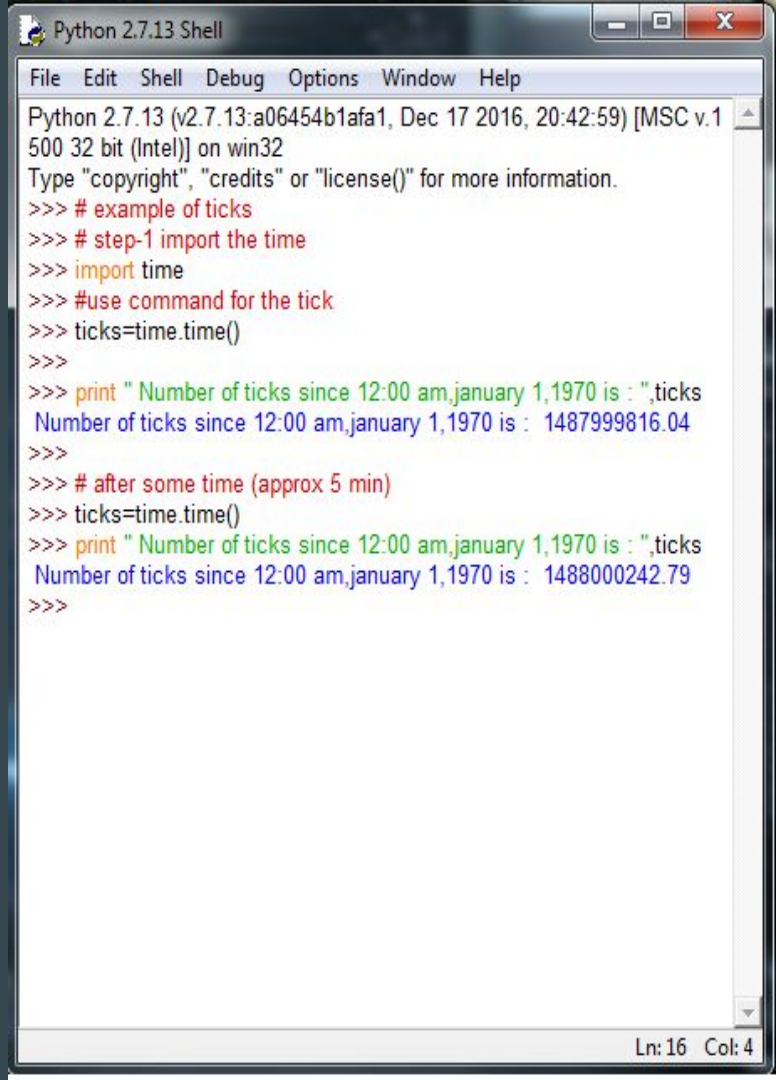
Date & Time

Date & Time

- A Python program can handle date and time in several ways.
- Python's time and calendar modules help track dates and times.

Tick

- Time intervals are floating-point numbers in units of seconds.
- Particular instants in time are expressed in seconds since 12:00am, January 1, 1970(epoch).



Time Tuple

- Many of Python's time functions handle time as a tuple of numbers.....

Index	Field	Values
0	4-digit year	2008
1	Month	1 to 12
2	Day	1 to 31
3	Hour	0 to 23
4	Minute	0 to 59
5	Second	0 to 61 (60 or 61 are leap-seconds)
6	Day of Week	0 to 6 (0 is Monday)
7	Day of year	1 to 366 (Julian day)
8	Daylight savings	-1, 0, 1, -1 means library determines DST

Time Attribute

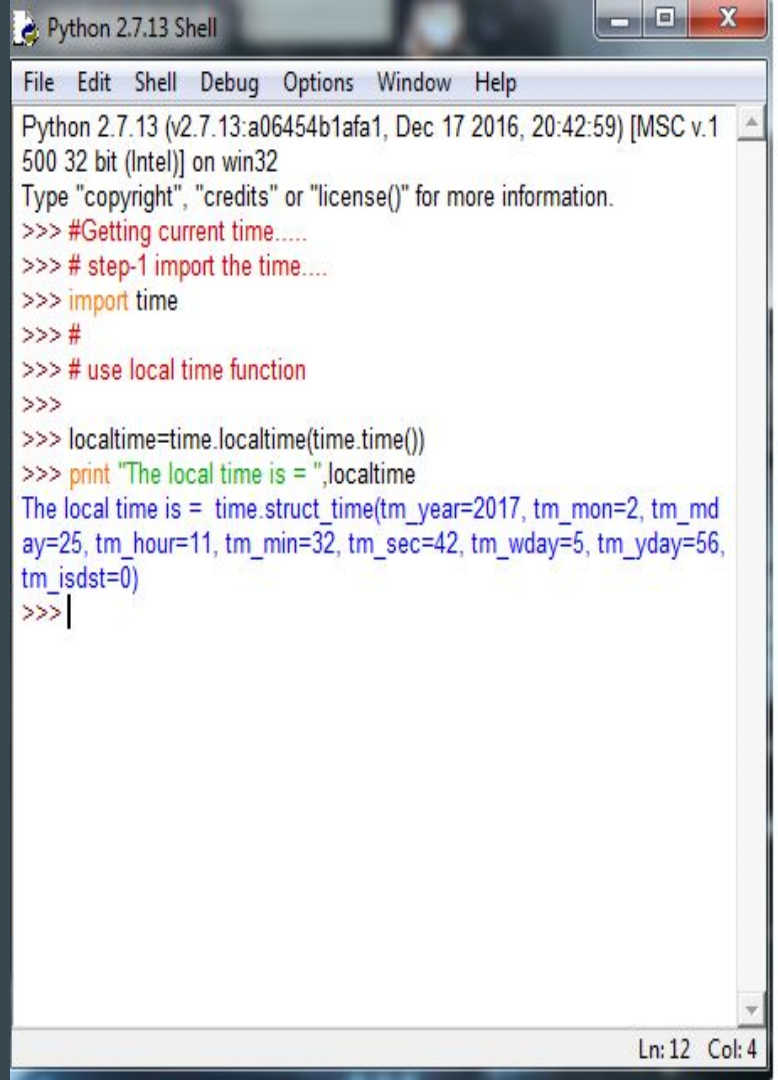
- Tuple is equivalent to **struct_time** structure. This structure has following attributes .....

Index	Attributes	Values
0	tm_year	2008
1	tm_mon	1 to 12
2	tm_mday	1 to 31
3	tm_hour	0 to 23
4	tm_min	0 to 59
5	tm_sec	0 to 61 (60 or 61 are leap-seconds)
6	tm_wday	0 to 6 (0 is Monday)
7	tm_yday	1 to 366 (Julian day)
8	tm_isdst	-1, 0, 1, -1 means library determines DST

Getting Current Time

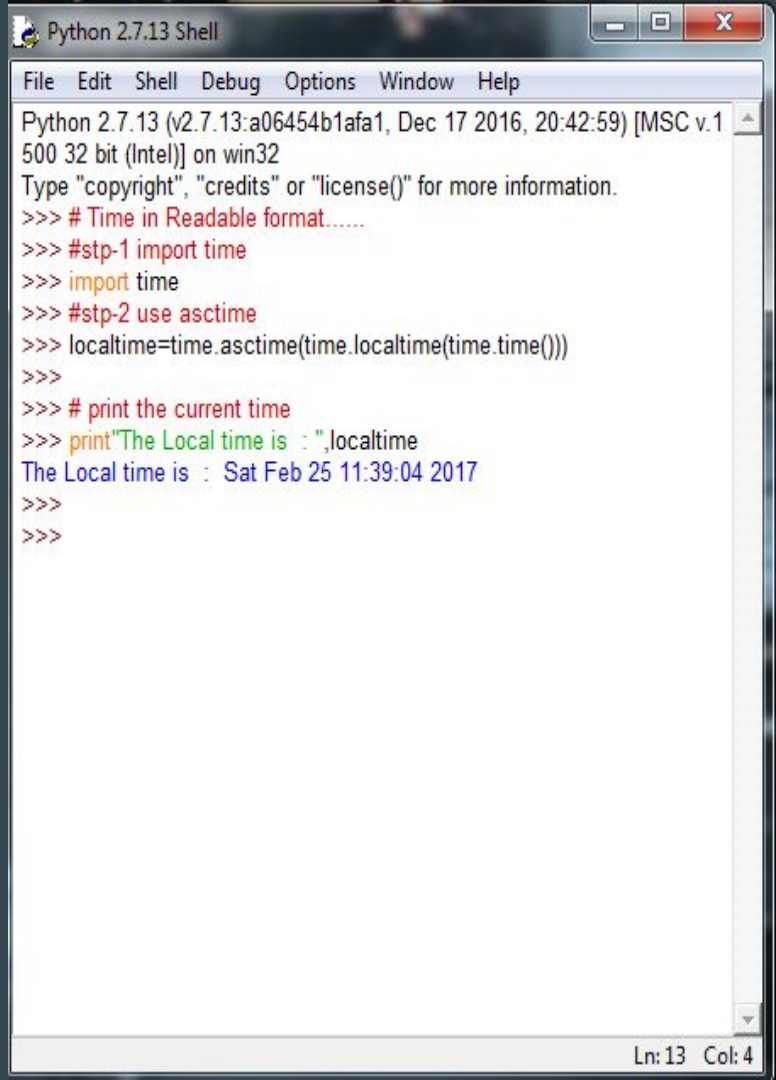
- To translate a time instant from a *seconds since the epoch* floating-point value into a time-tuple

- pass the floating-point value to a function (e.g., localtime) that returns a time-tuple with all nine items valid.



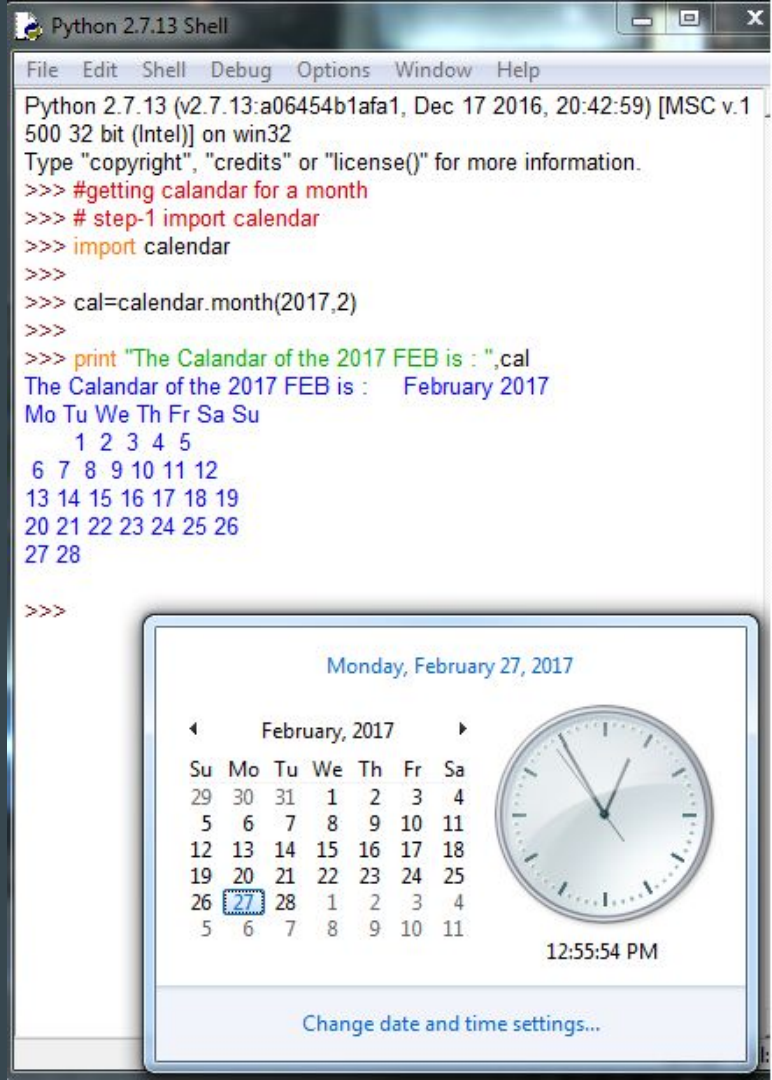
Getting Formatted Time

- Format any time as per your requirement, but simple method to get time in readable format is asctime()



Getting calendar

- The calendar module gives a wide range of methods to play with yearly and monthly calendars. Here, we print a calendar for a given month



calendar Module

1. **calendar.isleap(year)**

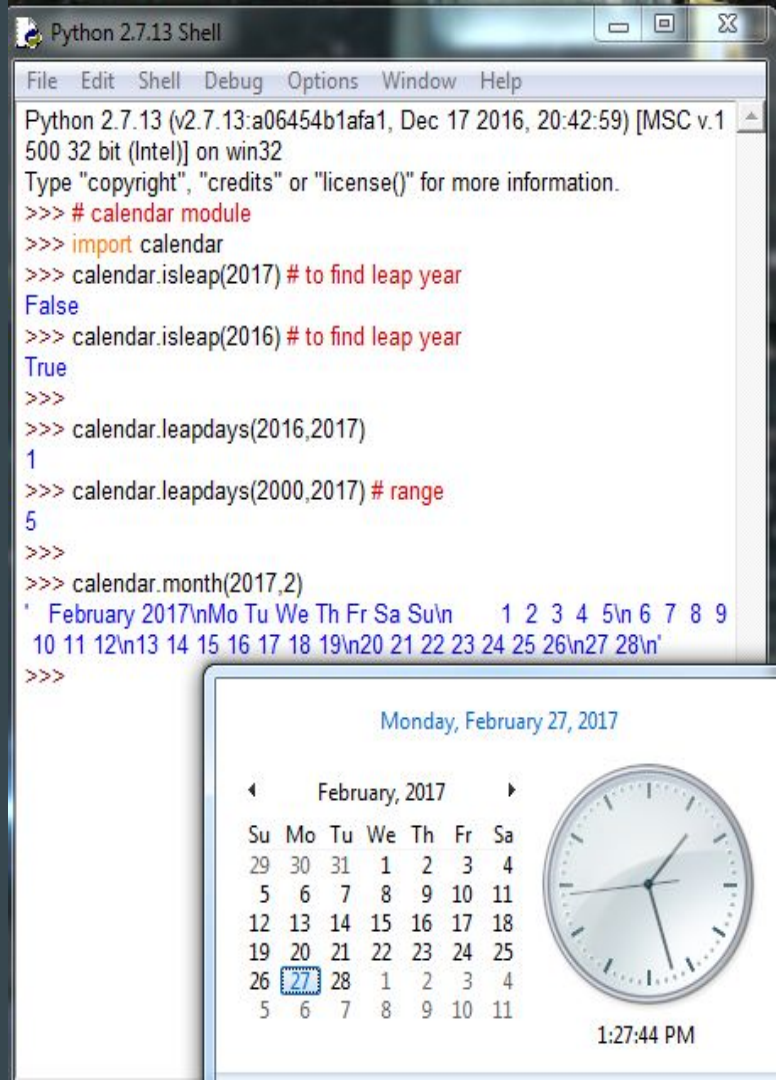
   Returns True if year is a leap year; otherwise, False.

2. **calendar.leapdays(y1,y2)**

   Returns the total number of leap days in the years within range(y1,y2)

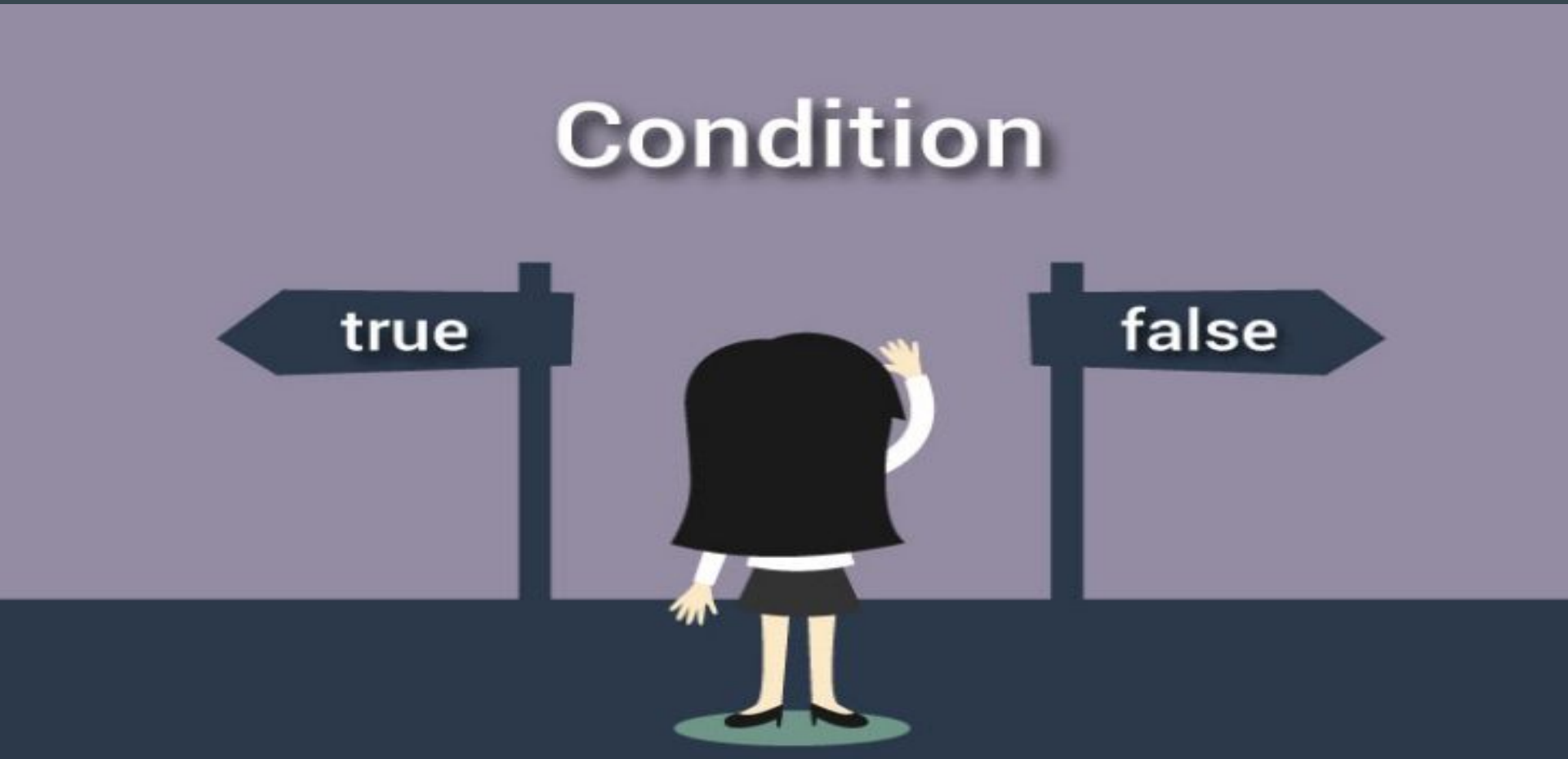
3. **calendar.month(year,month)**

   Returns a multiline string with a calendar for month month of year.



If....else Statement

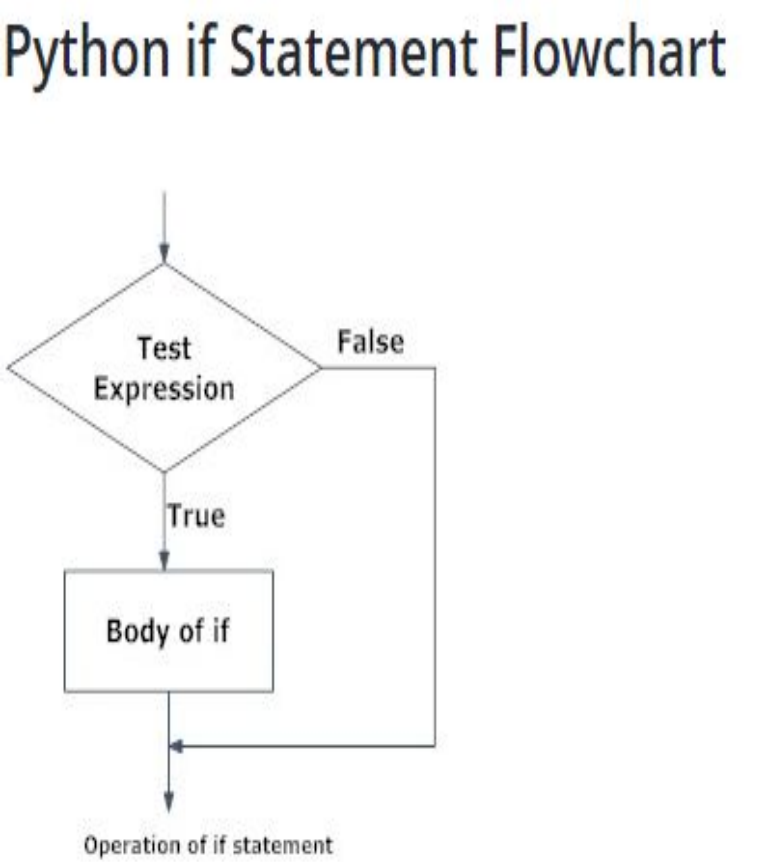
If....else Statement



If Statement Syntax

If test expression :
    statement(s)

- Here, the program evaluates the test expression and will execute statement(s) only if the text expression is True.
- If the text expression is False, the statement(s) is not executed.
- In Python, the body of the if statement is indicated by the indentation. Body starts with an indentation.
- Python interprets non-zero values as True. None and 0 are interpreted as False.



If Statement Example

num > 0 is the test expression.

The body of if is executed only if this evaluates to True.

When variable num is equal to 3, test expression is true and body inside body of if is executed.

If variable num is equal to -1, test expression is false and body inside body of if is skipped.

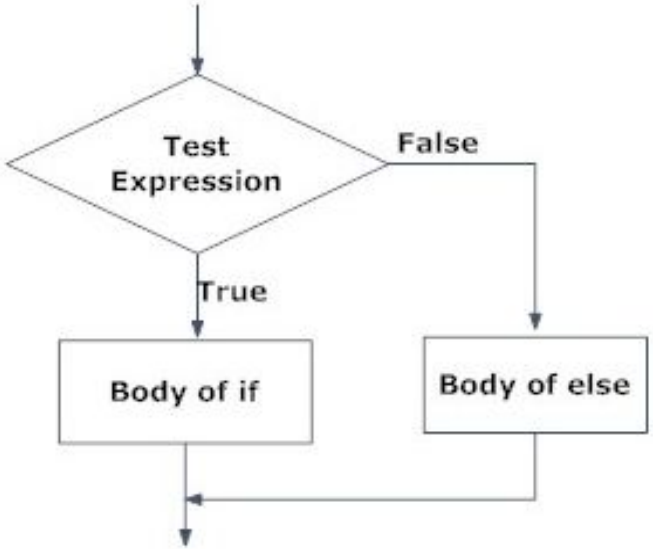
```
# If the number is positive, we print an appropriate message

num = 3

if num > 0:

print(num, "is a positive number")

print("This is always printed")

num = -1

if num > 0:

print(num, "is a positive number")

print("This is also always printed")
```

If..else Statement syntax

- The if..else statement evaluates test expression and will execute body of if only when test condition is True.
- If the condition is False, body of else is executed. Indentation is used to separate the blocks.

```
If test expression :
        Body of if
else:
        Body of else
```

Python if..else Flowchart



Operation of if...else statement

If..else Statement Example

when num is equal to 3, the test expression is true and body of if is executed and body of else is skipped.

If num is equal to -5, the test expression is false and body of else is executed and body of if is skipped.

If num is equal to 0, the test expression is true and body of if is executed and body of else is skipped.

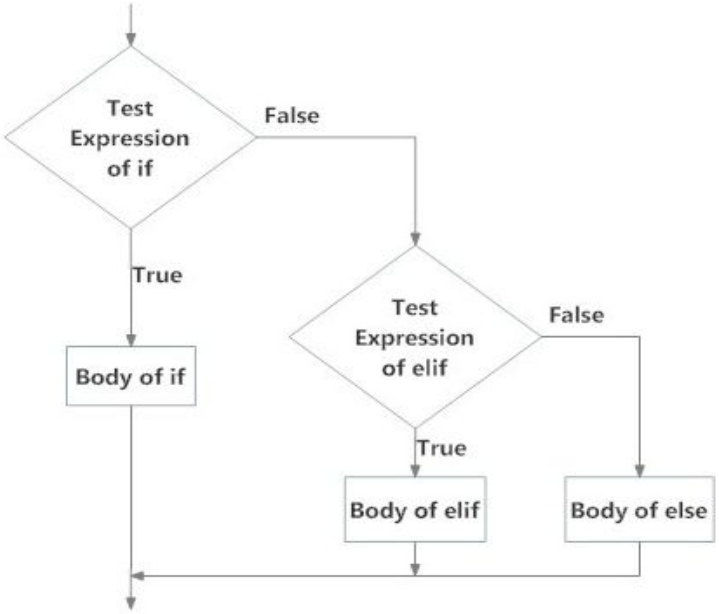
```
# Program checks if the number is positive or negative

# And displays an appropriate message

num = 3

# Try these two variations as well.

 # num = -5
# num = 0

if num >= 0:
 print("Positive or Zero")

else:
   print("Negative number")
```

If..elif..else Syntax

- The elif is short for else if. It allows us to check for multiple expressions.
- If all the conditions are False, body of else is executed.
- Only one block among the several if...elif...else blocks is executed according to the condition.
- The if block can have only one else block. But it can have multiple elif blocks

```
If test expression:
        Body of if
Elif test expression:
        Body of elif
else :
        Body of else
```

Flowchart of if...elif...else



Operation of if...elif...else statement

If..elif..else example

When variable num is positive, Positive number is printed.

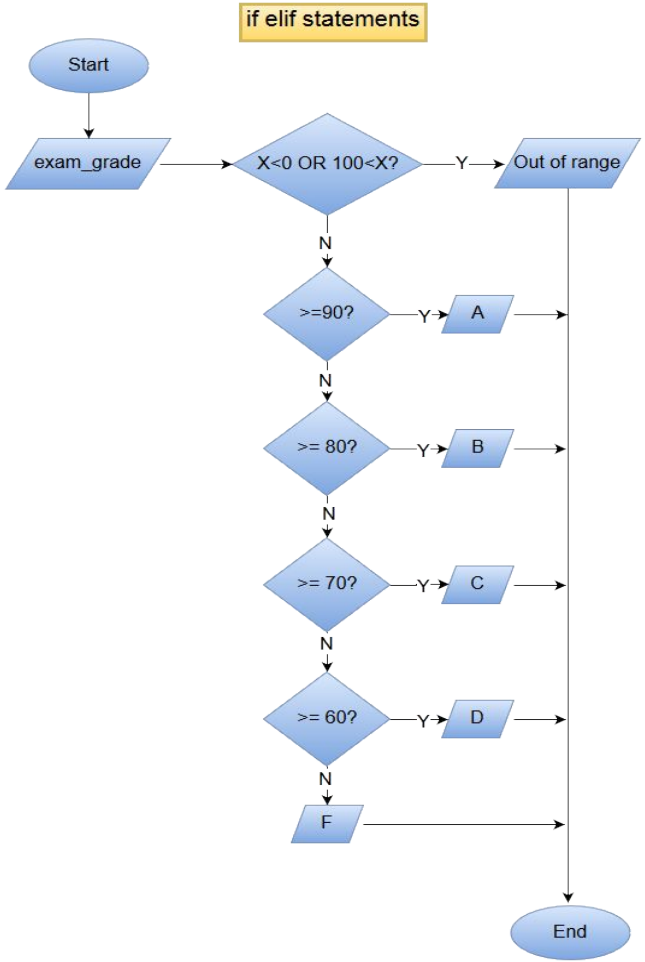
If num is equal to 0, Zero is printed.

If num is negative, Negative number is printed

```
# In this program
# we check if the number is
#positive or
# negative or zero and
# display an appropriate message
num = 3.4
# Try these two variations as well
:# num = 0
# num = -4.5

if num > 0:
  print("Positive number")

elif num == 0:
   print("Zero")

else:
  print("Negative number")
```

Nested if statements

- We can have a if...elif...else statement inside another if...elif...elsestatement. This is called nesting in computer programming.



Nested if example

Output 1

Enter a number: 7

Positive number

Output 2

Enter a number: -10

Negative number

Output 3

Enter a number: 0

Zero

```
# In this program, we input a
#number  check if the number is
#positive or  negative or zero and
#display an appropriate message
# This time we use nested if

num = float(input("Enter a
number: "))

 if num >= 0:
       if num == 0:
       print("Zero")

 else:
       print("Positive number")

 else:
             print("Negative number")
```

For ...Loop

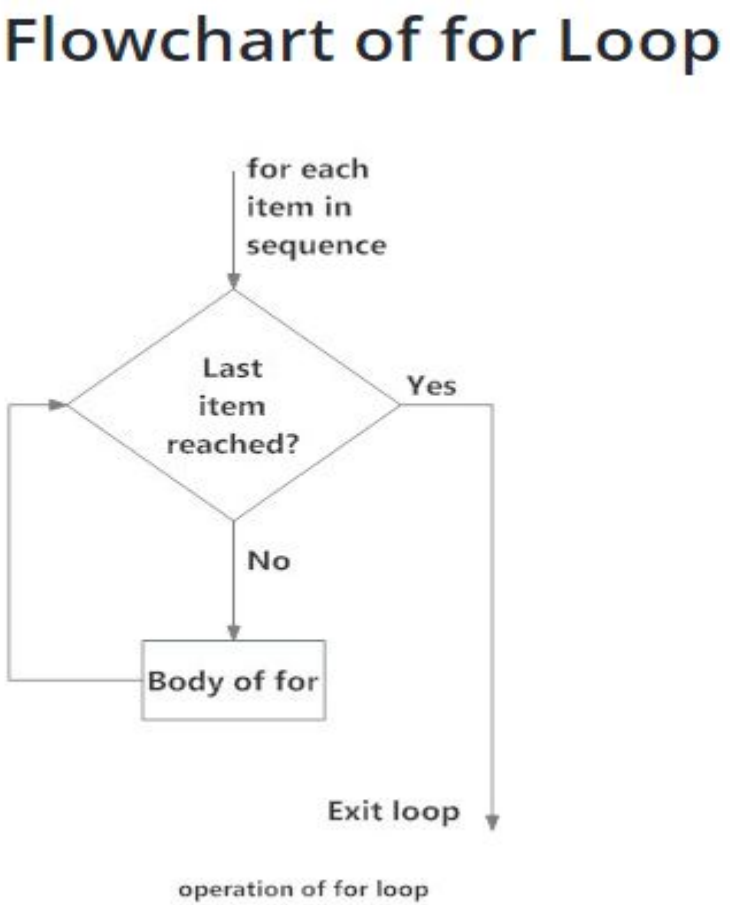
Syntax of for Loop

- Here, Val is the variable that takes the value of the item inside the sequence on each iteration.

- Loop continues until we reach the last item in the sequence. The body of for loop is separated from the rest of the code using indentation.

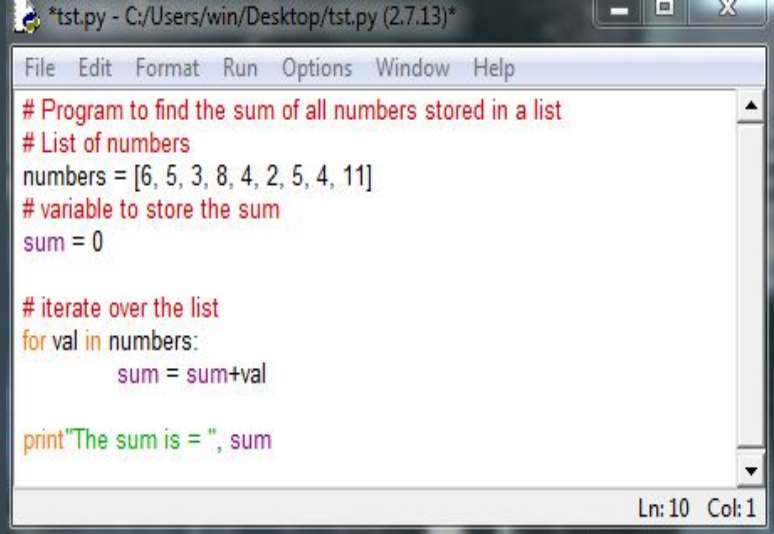
For Val in sequence:

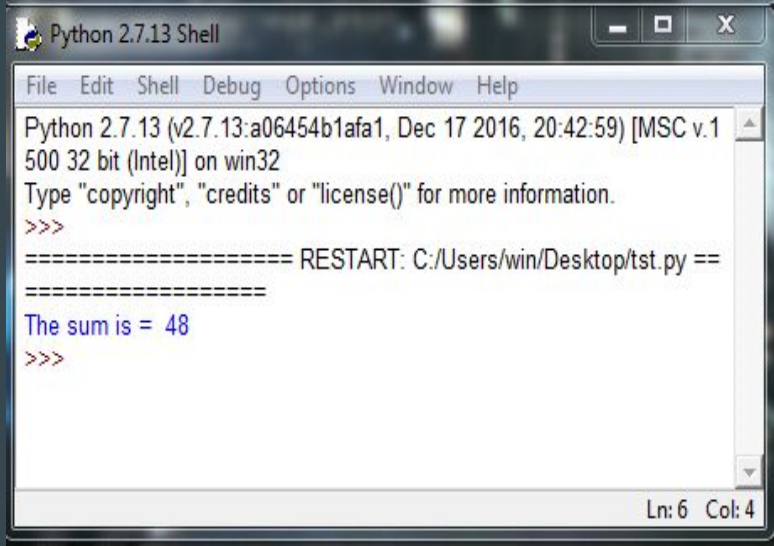
Body of for



for Loop example

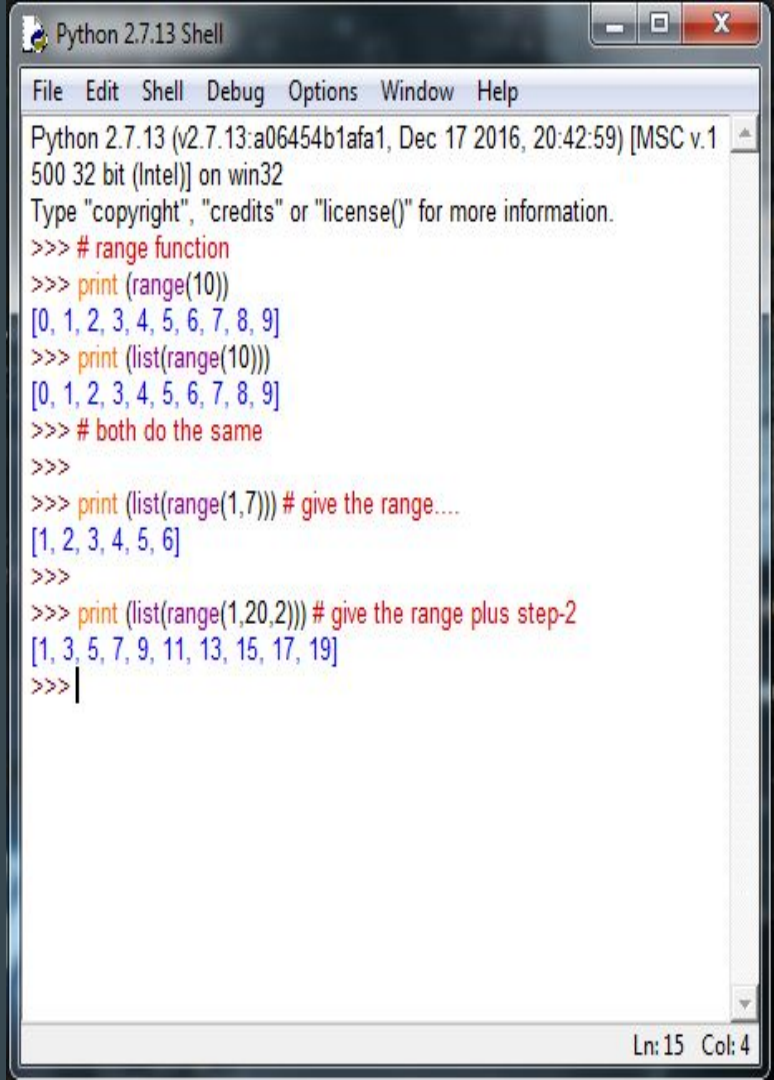
- In this program, we take a list as number.
- Sum of this list will be store in the SUM
- After that we use a for loop to sum all the numbers in the List.
- Sum of all the List number will be store in the Sum variable.





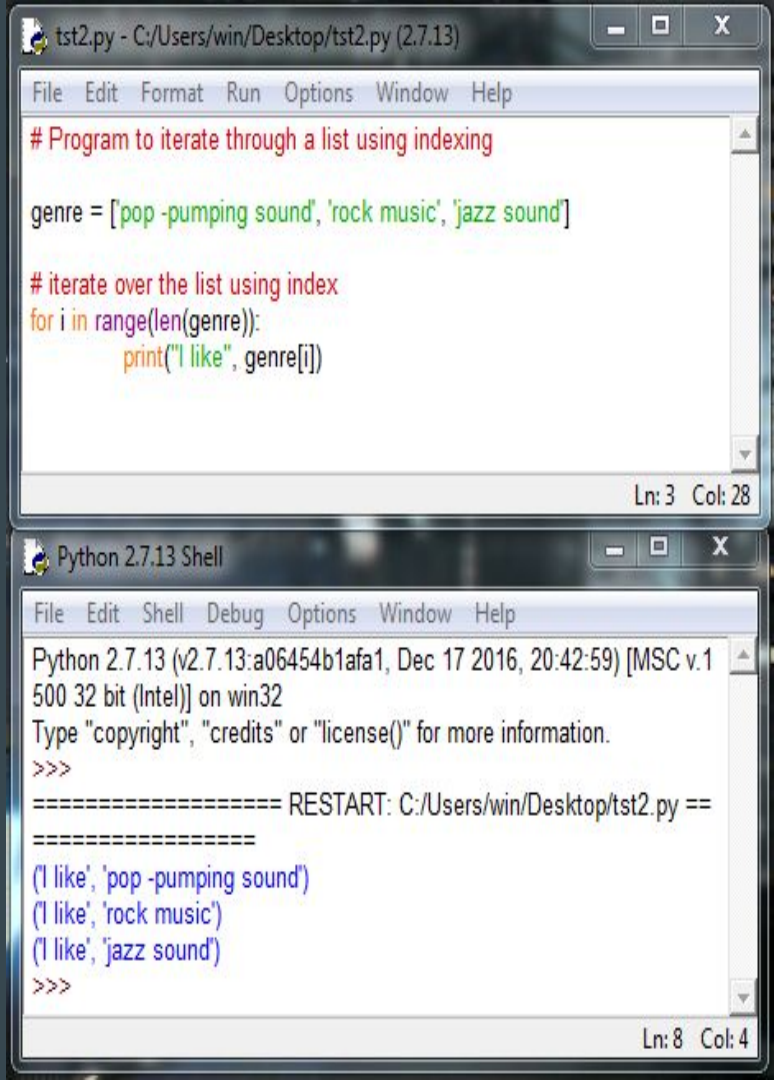
The range () function

- We can generate a sequence of numbers using range() function.
- We can also define the start, stop and step size as range(start,stop,step size). step size defaults to 1 if not provided.
- To force this function to output all the items, we can use the function list().



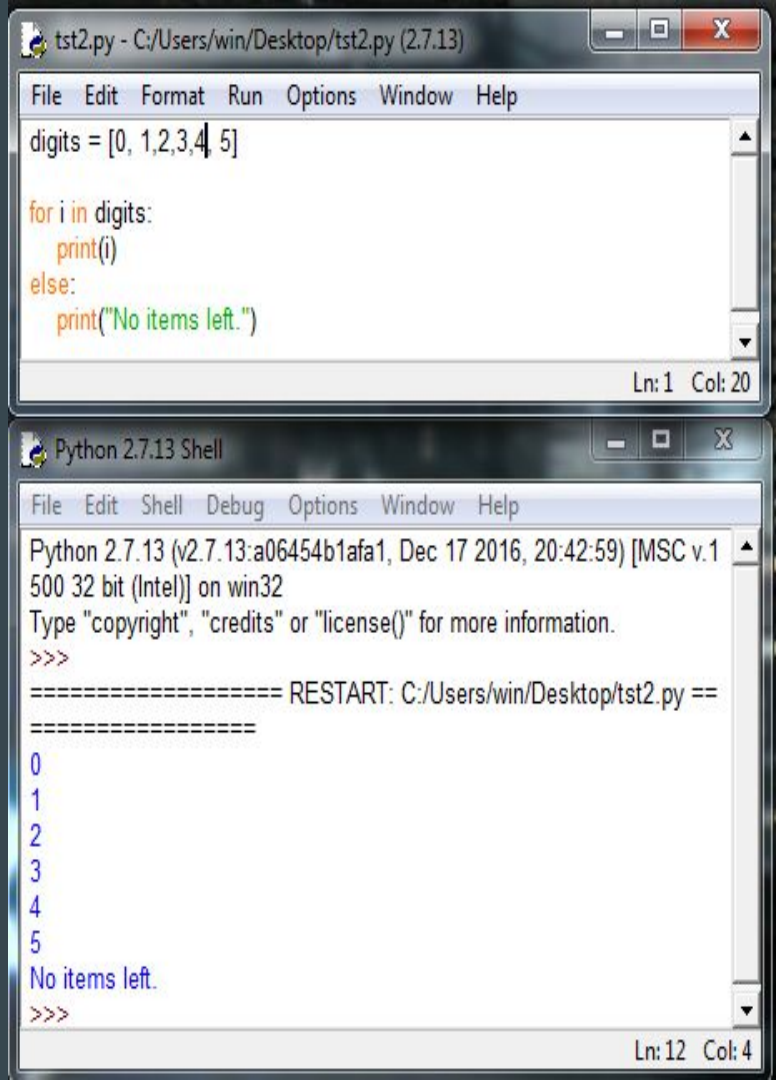
The len () Function

- We can use the range() function in for loops to iterate through a sequence of numbers.
- It can be combined with the len() function to iterate through a sequence using indexing.



For loop with else

- A for loop can have an optional else block as well.
- The else part is executed if the items in the sequence used in for loop exhausts.



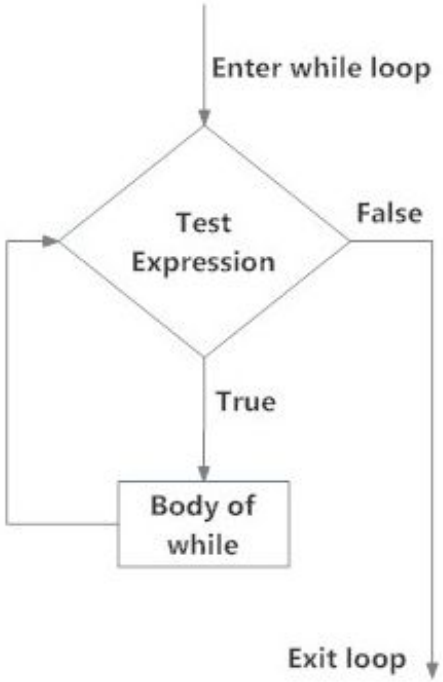
While... Loop

Syntax of While loop

- In while loop, test expression is checked first. The body of the loop is entered only if the test expression evaluates to True.
- After one iteration, the test expression is checked again.
- This process continues until the test expression evaluates to False.

While test expression :
Body Of While

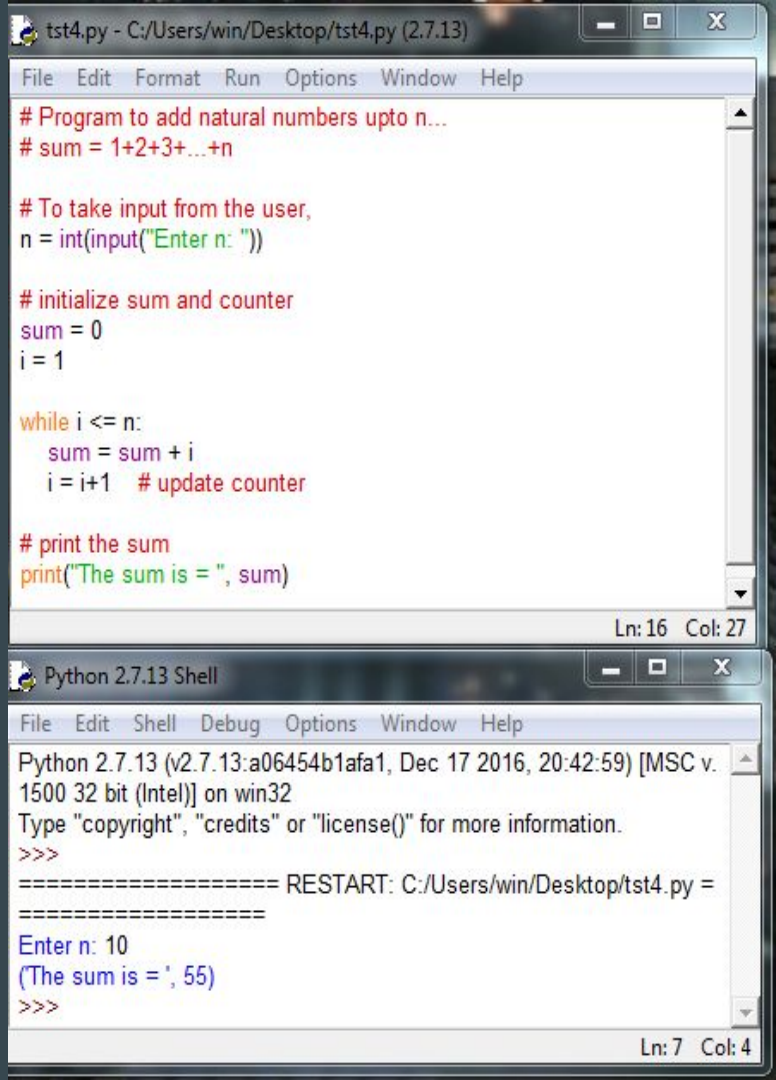
Flowchart of while Loop



operation of while loop

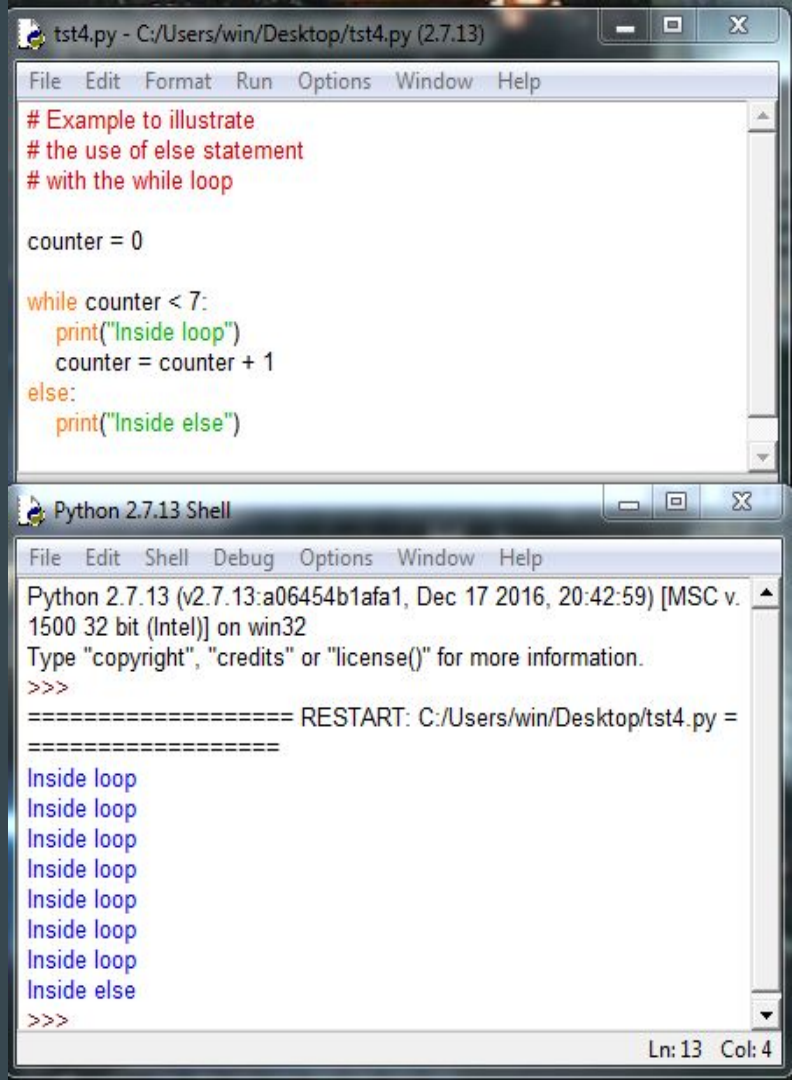
While loop example

- The test expression will be True as long as our counter variable i is less than or equal to n (10 in our program).



While loop with else

- Same as that of for loop, we can have an optional else block with while loop as well.
- The else part is executed if the condition in the while loop evaluates to False. The while loop can be terminated with a break statement.
- In such case, the else part is ignored. Hence, a while loop's else part runs if no break occurs and the condition is false.

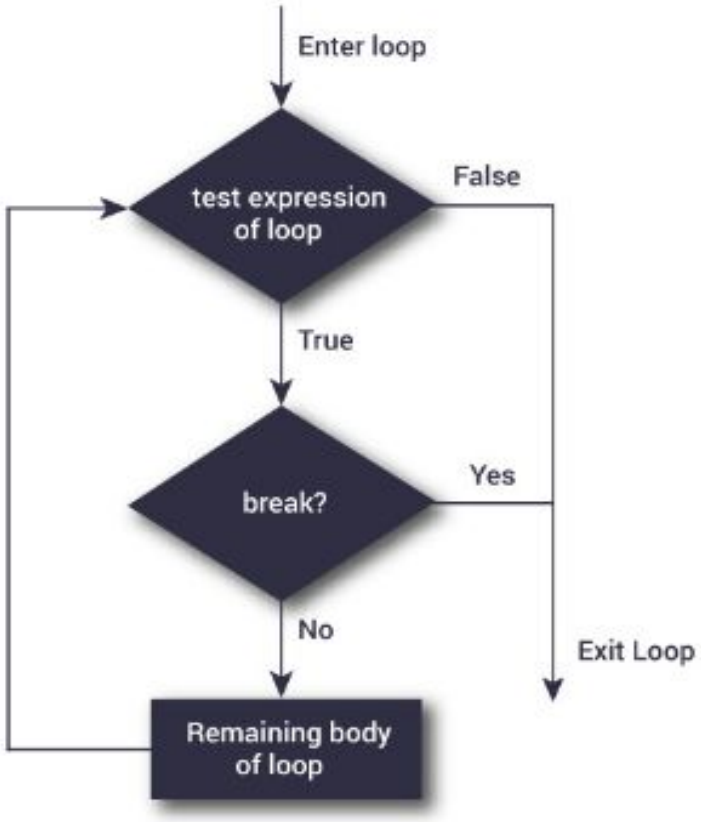


Break .....Continue

Python break statement

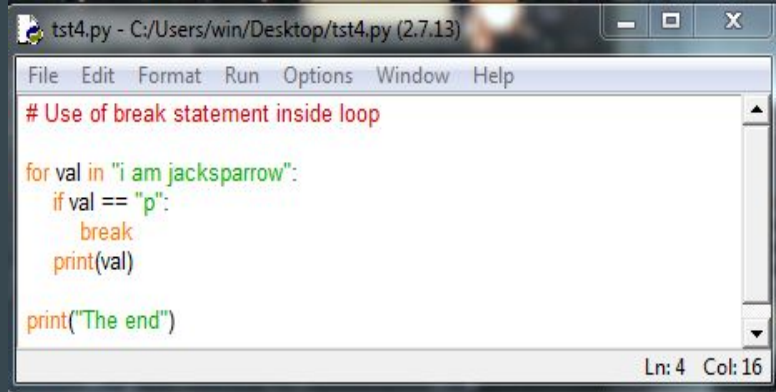
- The break statement terminates the loop containing it. Control of the program flows to the statement immediately after the body of the loop.
- If break statement is inside a nested loop (loop inside another loop), break will terminate the innermost loop.

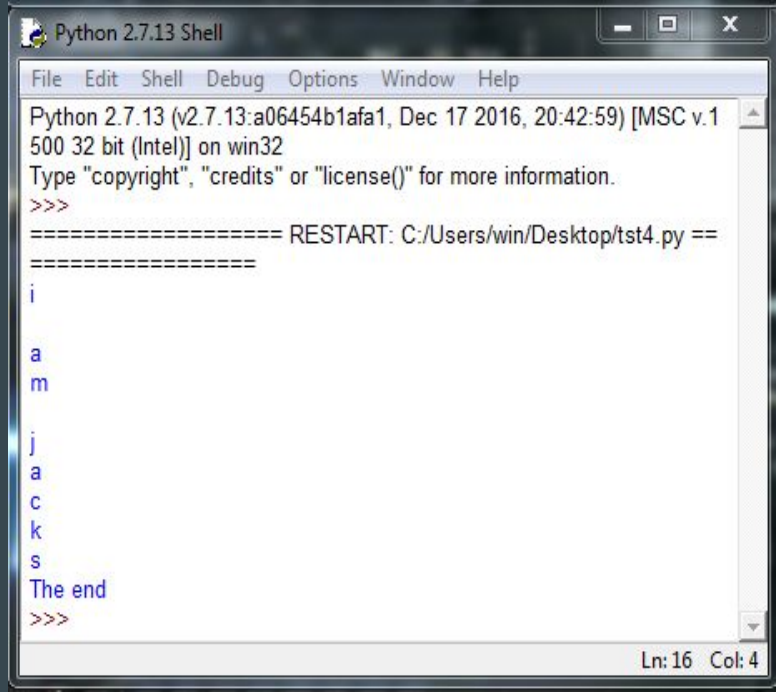
Flowchart of break



break statement example

- In this program, we iterate through the “I am jacksparrow " sequence.
- We check if the letter is “p", upon which we break from the loop.
- Hence, we see in our output that all the letters up till “p" gets printed. After that, the loop terminates.

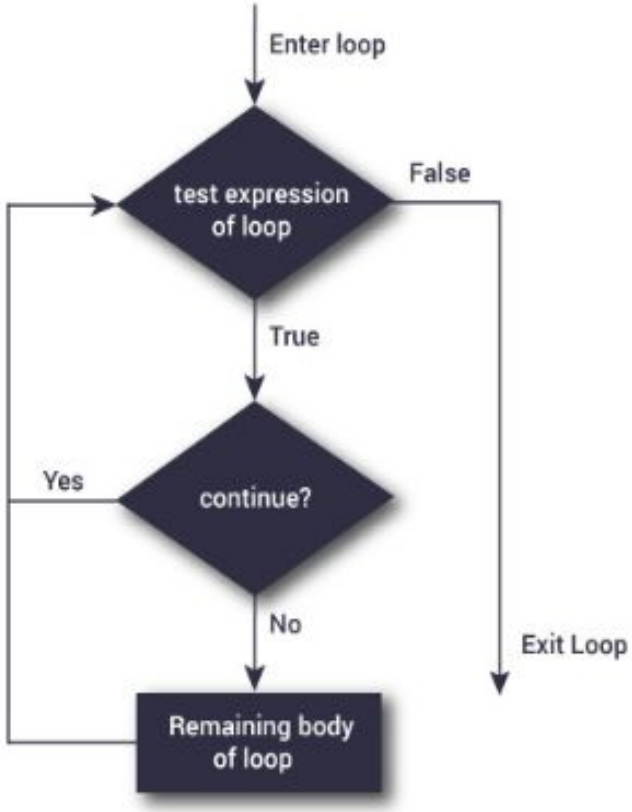




Python continue statement

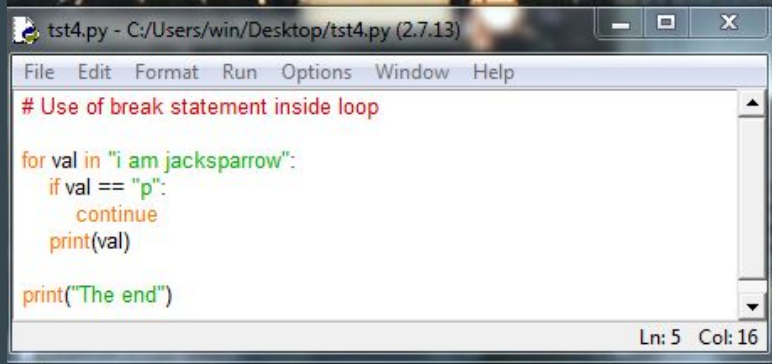
- The continue statement is used to skip the rest of the code inside a loop for the current iteration only.
- Loop does not terminate but continues on with the next iteration.

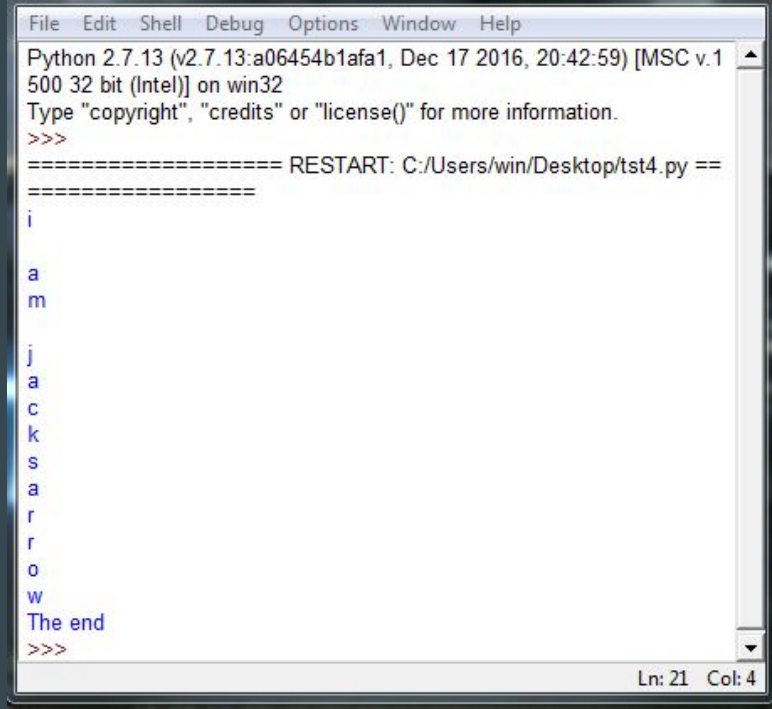
Flowchart of continue



Continue statement example

- This program is same as the above example except the break statement has been replaced with continue.
- We continue with the loop, if the string is “p", not executing the rest of the block. Hence, we see in our output that all the letters except “p" gets printed





Function

What is Python Function?

- In Python, function is a group of related statements that perform a specific task.
- Functions help break our program into smaller and modular chunks. As our program grows larger and larger, functions make it more organized and manageable.
- Furthermore, it avoids repetition and makes code reusable.

Syntax Of Function

```
def  function name(parameters) :
       “” doc string “”
Statement (s)
```

- Keyword def marks the start of function header.
- A function name to uniquely identify it. Function naming follows the same rules of writing identifiers in Python.
- Parameters (arguments) through which we pass values to a function. They are optional.
- A colon (:) to mark the end of function header.
- Optional documentation string (docstring) to describe what the function does.

Syntax Of Function

```
def  function name(parameters) :
        “” doc string “”
Statement (s)
```

- One or more valid python statements that make up the function body. Statements must have same indentation level (usually 4 spaces).
- An optional return statement to return a value from the function.

Example Of a Function

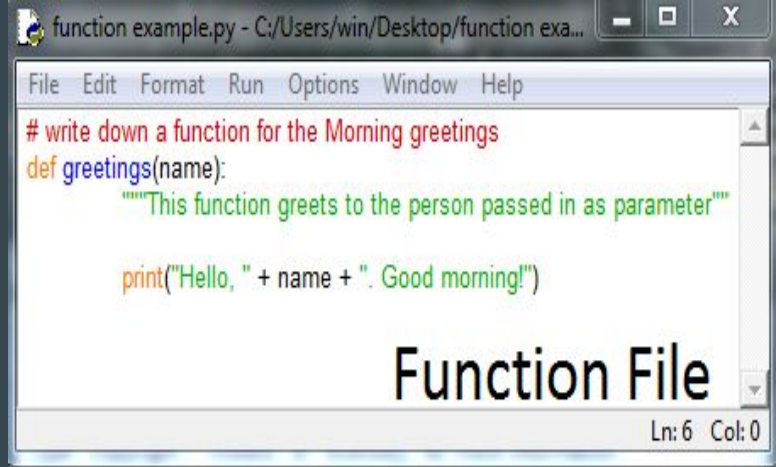
Problem Statement :write a function which gives you a morning Greeting .

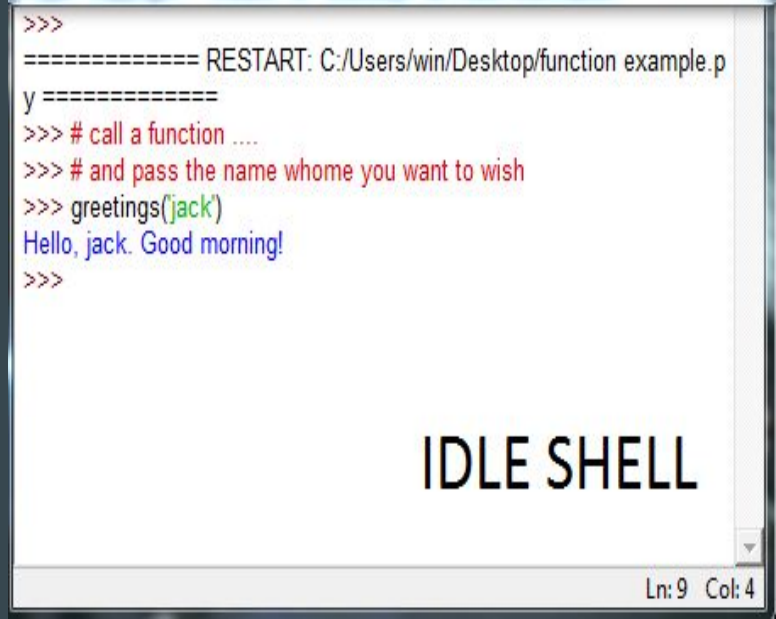
Step 1:

>>Define a Function in new python file

Step 2:

>>Call a function from Python Idle shell

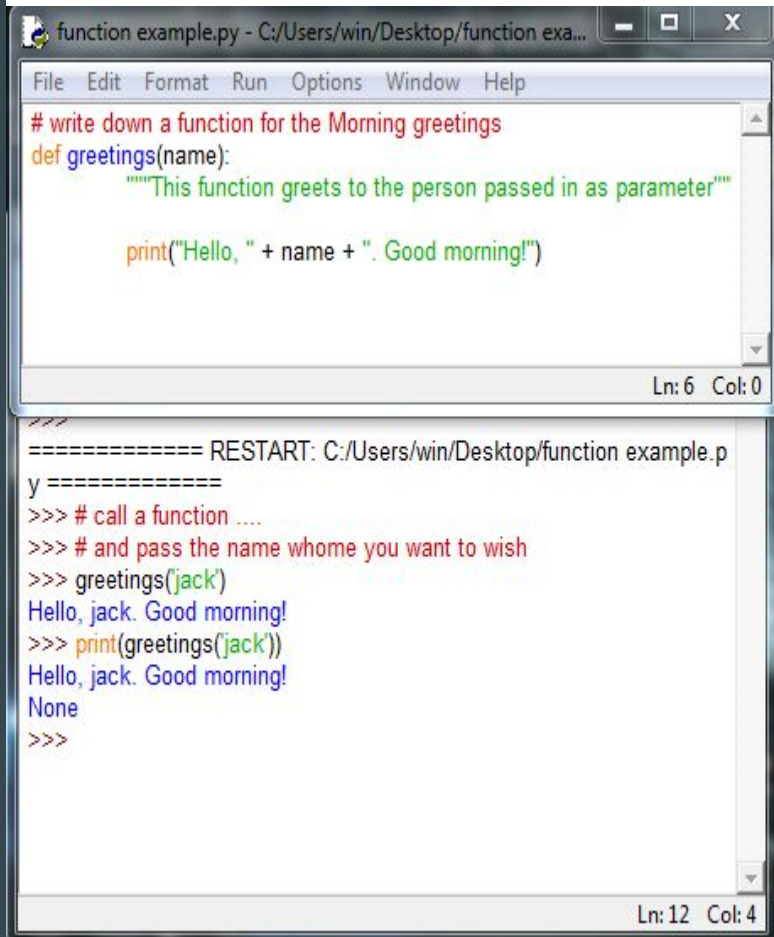




The Return Statement syntax

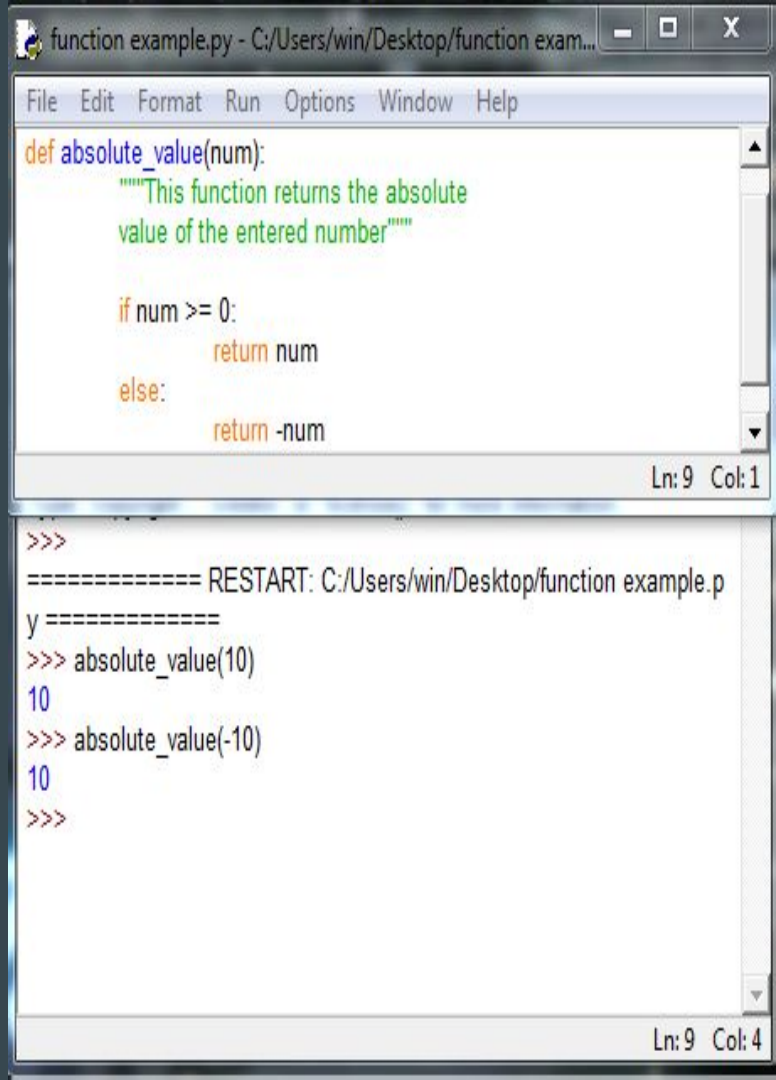
- This statement can contain expression which gets evaluated and the value is returned.
- If there is no expression in the statement or the return statement itself is not present inside a function, then the function will return the None object.
- Here, None is the returned value.

return [expression_list]



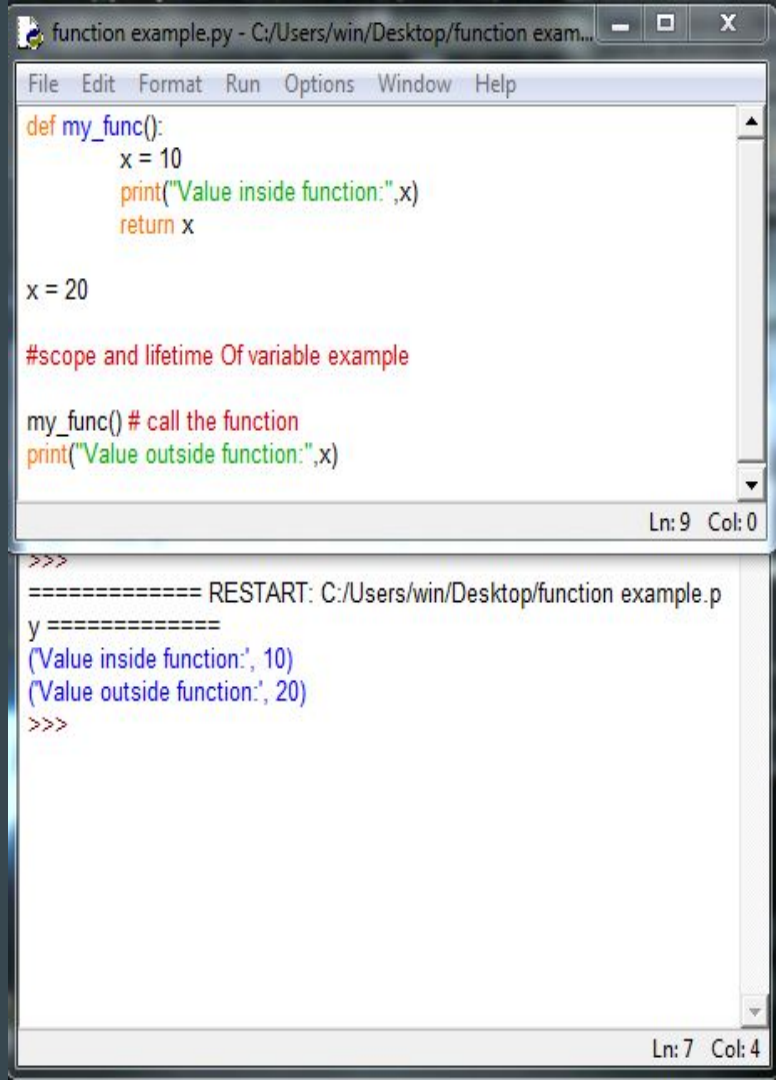
Example Of Return

Problem Statement: Write a function for the absolute number and return it's value.



Scope and Lifetime Of variables

- Scope of a variable is the portion of a program where the variable is recognized. Parameters and variables defined inside a function is not visible from outside. Hence, they have a local scope.
- Lifetime of a variable is the period throughout which the variable exits in the memory. The lifetime of variables inside a function is as long as the function executes.



Scope and Lifetime Of variables

- They are destroyed once we return from the function. Hence, a function does not remember the value of a variable from its previous calls.