

Python Programming

operators, loops, functions

Dr. Sarwan Singh



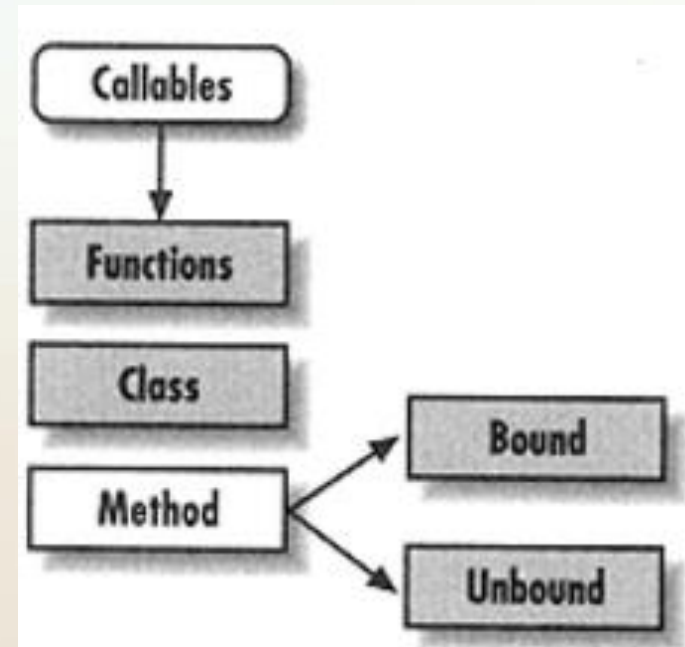
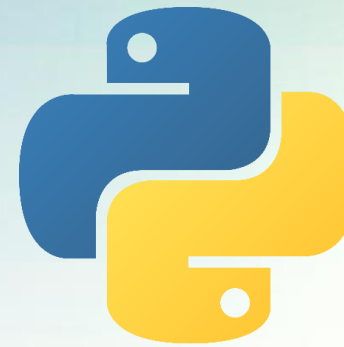
Agenda

- Python –
 - operators
 - decision constructs
 - looping constructs
- hands on – *programming*

Artificial Intelligence

Machine Learning

Deep Learning



Python was conceived in the late 1980s, & implementation began in December 1989 by **Guido van Rossum**



Programming *with* Python



Some companies I know that use python are:

- Google (Youtube)
- Facebook (Tornado)
- Dropbox.
- Yahoo.
- NASA.
- IBM.
- Mozilla.
- Quora :D.

More items...

What top tier companies use Python? - Quora
<https://www.quora.com/What-top-tier-companies-use-Python>



Basic operator

- Arithmetic Operators
- Comparison (Relational) Operators
- Assignment Operators
(+= , -= , *= , /= , %=)
- Logical Operators
(and, or, not)
- Bitwise Operators

Arithmetic Operator

+ Addition
- Subtraction
* Multiplication
/ Division
% Modulus
** Exponent
// Floor division

9//2 -> 4

Comparison Operator

==

!=

<>

>

<

>=

<=

Bitwise Operator

& Binary AND

| Binary OR

^ Binary XOR

~ Binary Ones Complement

<< Binary Left Shift

>> Binary Right Shift



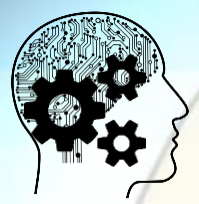
Decision making

```
x = int( input('enter marks'))  
if (x>50) : print('pass')  
else : print('fail')
```

Or

```
x = int( input('enter marks'))  
if (x>50) :  
    print('pass')  
else :  
    print('fail')
```

```
if expression1:  
    statement(s)  
    if expression2:  
        statement(s)  
    elif expression3:  
        statement(s)  
    elif expression4:  
        statement(s)  
    else:  
        statement(s)  
else:  
    statement(s)
```



single line code

```
print("A") if a > b else print("=") if a == b else print("B")
```

or

```
if a > b
```

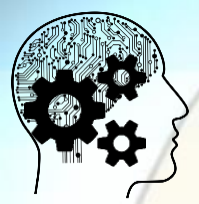
```
    print("A")
```

```
else if a == b
```

```
    print("=")
```

```
else
```

```
    print("B")
```



Exercise

1. Write command to check whether input number is even or odd.
2. Write a command/program to accept marks from user and print the division.
3. Write command/s to return sum of digits of given number.



loops

while expression :
 statements()

```
i=0  
while (i<5) :  
    print (i, 'Jai Ho')  
    i=i+1
```

```
0 Jai Ho  
1 Jai Ho  
2 Jai Ho  
3 Jai Ho  
4 Jai Ho
```

while expression :
 statements()
else :
 statements()

```
i=0  
while (i<5) :  
    print (i, 'Jai Ho')  
    i=i+1  
else :  
    print (i, ' Its over now')
```

```
0 Jai Ho  
1 Jai Ho  
2 Jai Ho  
3 Jai Ho  
4 Jai Ho  
5  Its over now
```




for loop

for iterating Variable in sequence
statement/s

```
In [3]: states=['J&K', 'HimachalPradesh', 'Punjab', 'Delhi']  
for st in states:  
    print (st)
```

```
J&K  
HimachalPradesh  
Punjab  
Delhi
```

```
In [4]: for st in range(len(states)):  
        print (states[st])
```

```
J&K  
HimachalPradesh  
Punjab  
Delhi
```

```
In [5]: for alpha in 'India':  
        print(alpha)
```

```
I  
n  
d  
i  
a
```

for iterating Variable in sequence
statement/s

else:

statement/s

```
In [7]: for st in range(len(states)):  
        print (states[st])  
else :  
        print('-----Its over ----- ')
```

```
J&K  
HimachalPradesh  
Punjab  
Delhi  
-----Its over -----
```



For loop: Example

1. `for n in range(21, 0, -3):`
 `print(n, ", end=")`

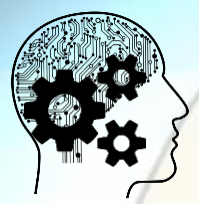
Output: 21 18 15 12 9 6 3

2. `for n in range(1000) :`
 `print(n, end=' ')`

Output: 0, 1, 2, . . . , 999.

3. `sum = 0`
 `for i in range(1, 100):`
 `sum += i`
 `print(sum)`

Output: adds nos from 1 to 99



Iteration : for

`range(10) → 0, 1, 2, 3, 4, 5, 6, 7, 8, 9`

`range(1, 10) → 1, 2, 3, 4, 5, 6, 7, 8, 9`

`range(1, 10, 2) → 1, 3, 5, 7, 9`

`range(10, 0, -1) → 10, 9, 8, 7, 6, 5, 4, 3, 2, 1`

`range(10, 0, -2) → 10, 8, 6, 4, 2`

`range(2, 11, 2) → 2, 4, 6, 8, 10`

`range(-5, 5) → -5, -4, -3, -2, -1, 0, 1, 2, 3, 4`

`range(1, 2) → 1`

`range(1, 1) → (empty)`

`range(1, -1) → (empty)`

`range(1, -1, -1) → 1, 0`

`range(0) → (empty)`



Exercise

1. Write program to check whether given number is prime or not
2. Write a program to find those numbers which are divisible by 7 and multiple of 5, between 1500 and 2700 (both included).
3. Write a Python program to get the Fibonacci series between 0 to 50.
4. Write a program to construct the pattern, using a nested for loop.

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

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

```
1
22
333
4444
55555
666666
7777777
88888888
999999999
```



Exercise

1. Write a program to print the table of given number entered by the user
2. Write a program which can compute the factorial of a given numbers.

$$5 * 1 = 5$$

$$5 * 2 = 10$$

$$5 * 3 = 15$$

$$5 * 4 = 20$$

$$5 * 5 = 25$$

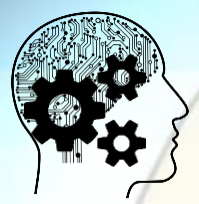
$$5 * 6 = 30$$

$$5 * 7 = 35$$

$$5 * 8 = 40$$

$$5 * 9 = 45$$

$$5 * 10 = 50$$



Loop Control Statements

Break : Terminates loop statement

```
for alpha in 'Greatness':  
    if alpha == 'n':  
        break  
    print ('letter ', alpha)
```

```
letter G  
letter r  
letter e  
letter a  
letter t
```

continue : returns the control to the beginning of the while/for loop

```
for alpha in 'Greatness':  
    if alpha == 'n':  
        continue  
    print ('letter ', alpha)
```

```
letter G  
letter r  
letter e  
letter a  
letter t  
letter e  
letter s  
letter s
```

pass : is used when a statement is required syntactically but you do not want any command or code to execute

```
for alpha in 'Greatness':  
    if alpha == 'n':  
        pass  
    print ('Pass block')  
    print ('letter ', alpha)
```

```
letter G  
letter r  
letter e  
letter a  
letter t  
Pass block  
letter n  
letter e  
letter s  
letter s
```



String

- Strings Are Immutable : once created cannot be changed.

```
: #string concatenation
print (str + ' ' + str1)
#string slicing
print('str ',str)
print('str[1:3]',str[2:8] )
print('str[11:]',str[11:] )
print('str[:11]',str[:11] )
print('str[:-2]',str[:-2] )
print('str[-2]',str[-2] ) #second last str[len(str) -2]
```

```
Incredible India Great
str Incredible India
str[1:3] credib
str[11:] India
str[:11] Incredible
str[:-2] Incredible Ind
str[-2] i
```

```
print('str1 * 3 ',str1 * 3)
print('str1 * 3 ',str1 * 3)
print('str1 * 3 ',str1 * 3)
```

```
str1 * 3    GreatGreatGreat
```

```
#string length and index
for s in range(len(str1)):
    print(str1[s])
```

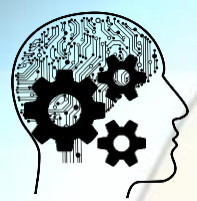
```
G
r
e
a
t
```

```
'e' in str1
```

```
True
```

```
for x in str1:
    print (x)
```

```
G
r
e
a
t
```



String methods

```
str.upper()
```

```
'INCREDIBLE INDIA'
```

```
str.capitalize()
```

```
'Incredible india'
```

```
#string.center(width[, fillchar])  
print(str.center(40))  
print(str.center(40, '-'))
```

```
                Incredible India  
-----Incredible India-----
```

```
#str.count(sub, start= 0,end=len(string))  
print(str.count('In'))
```

```
2
```

```
#str.find(str, beg=0, end=len(string))  
print(str.find('nd'))
```

```
12
```

```
str.join('-*-*-')
```

```
'-Incredible India*Incredible India-Incredible India*'
```

```
str.swapcase()
```

```
'iNCREDIBLE iNDIA'
```

```
str.title()
```

```
'Incredible India'
```

```
str.lower()
```

```
'incredible india'
```

```
about=''' This is multiline  
string and it can span across  
multiple lines'''
```

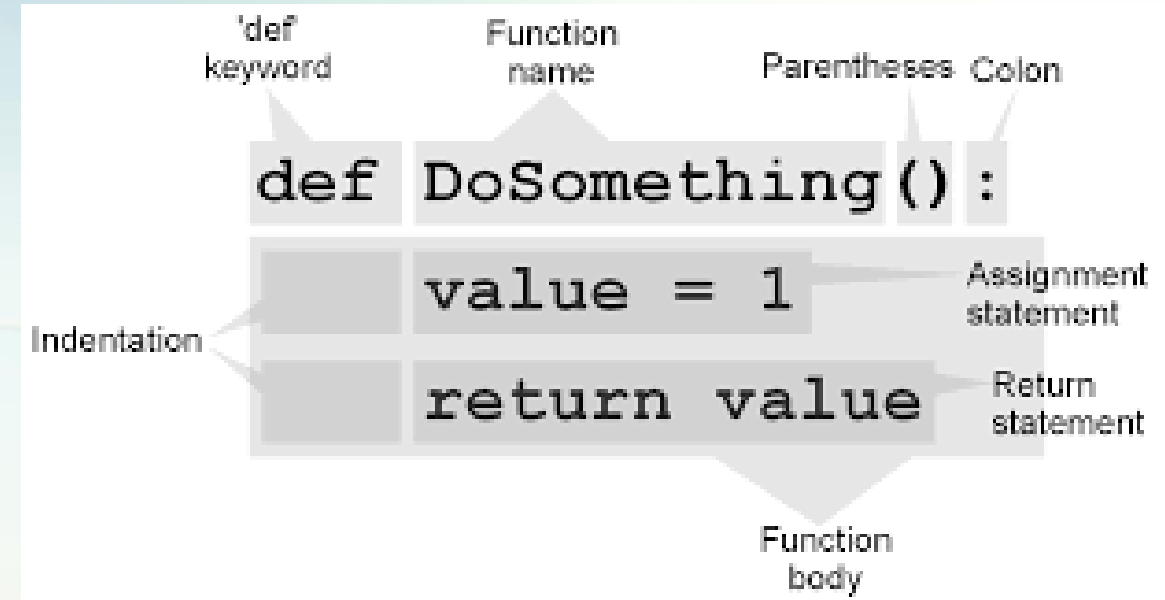
```
about
```

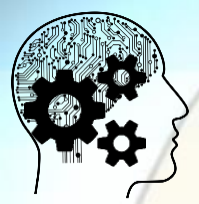
```
' This is multiline\nstring and it can span across\nmultiple lines'
```




Functions

- * Organises related code in blocks, so that it can be efficiently reused
- * Better modularity.





```
def functionname( arg1 , arg2, arg3,  
                 arg4=default_Value ) :  
    """function_docstring"""  
    statements.....  
    return [expression]
```

- Function can be system defined (e.g. print) or user defined
- Function can have variable number of arguments (e.g. print)
- All parameters (arguments) in the Python language are **passed by reference**.

```
def onePlus(a):  
    '''function increments the passed argument by one'''  
    return a+1
```

```
onePlus.__doc__
```

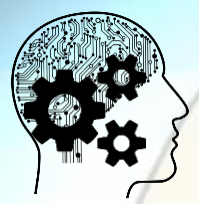
```
'function increments the passed argument by one'
```

```
onePlus(10)
```

```
11
```

```
print (onePlus(21))
```

```
22
```



Best practices

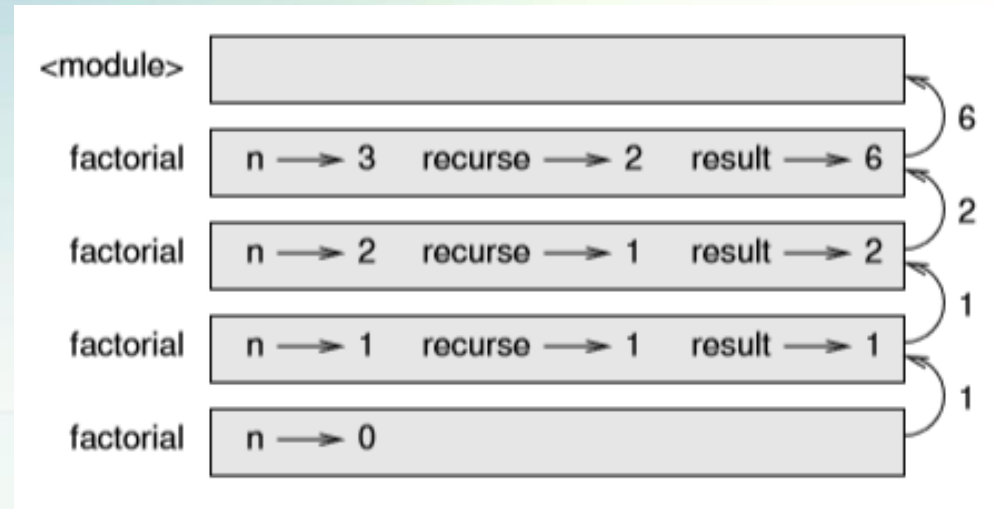
- Function should be written keeping in mind process of **incremental development**.
- **Scaffolding** : code like print statements used for building the program but is not part of the final product
- Initially write individual statements , later consolidate multiple statements in compound statement
- **Composition** is ability of to call function from another function



Recursion

```
def factorial(n):  
    if n == 0:  
        return 1  
    else:  
        recurse = factorial(n-1)  
        result = n * recurse  
    return result
```

Ex: write recursive function to generate/print Fibonacci series



Source : *thinkpython*



Call by value / Reference

```
def changeList(x):  
    print ('Value received      : ',x, ' id(x)=',id(x))  
    x+=[5,6]  
    print ('Value changed       : ',x, ' id(x)=',id(x))  
    return
```

```
arr=[1,2,3,4]  
print ('List before calling :',arr , ' id(a)=',id(arr))  
changeList(arr)  
print ('List after calling  :',arr , ' id(a)=',id(arr))
```

```
List before calling : [1, 2, 3, 4] id(a)= 2214105128328  
Value received      : [1, 2, 3, 4] id(x)= 2214105128328  
Value changed       : [1, 2, 3, 4, 5, 6] id(x)= 2214105128328  
List after calling  : [1, 2, 3, 4, 5, 6] id(a)= 2214105128328
```

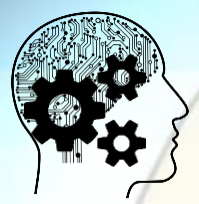
```
def changeMe(x):  
    print ('Value received      : ',x, ' id(x)=',id(x))  
    x=x+20  
    print ('Value changed       : ',x, ' id(x)=',id(x))  
    return
```

```
a=20  
print ('Value before calling :',a , ' id(a)=',id(a))  
changeMe(a)  
print ('Value after calling  :',a , ' id(a)=',id(a))
```

```
Value before calling : 20 id(a)= 1629250304  
Value received      : 20 id(x)= 1629250304  
Value changed       : 40 id(x)= 1629250944  
Value after calling  : 20 id(a)= 1629250304
```

```
arr=[1,2,3,4]  
print ('List before calling :',arr , ' id(a)=',id(arr))  
changeList(arr[:]) #passing a copy ( shallow copy )  
print ('List after calling  :',arr , ' id(a)=',id(arr))
```

```
List before calling : [1, 2, 3, 4] id(a)= 2214105040584  
Value received      : [1, 2, 3, 4] id(x)= 2214105177096  
Value changed       : [1, 2, 3, 4, 5, 6] id(x)= 2214105177096  
List after calling  : [1, 2, 3, 4] id(a)= 2214105040584
```



Function arguments

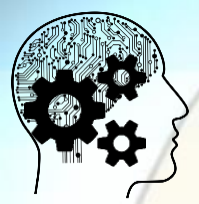
- Keyword argument
- Default argument
- Variable length arguments

```
def functionname([formal_args,] *var_args_tuple ):  
    for var in vartuple:  
        print var
```

```
def fibSeries(n):  
    a, b = 0, 1  
    while b < n:  
        print (b,)  
        a, b = b, a+b
```

```
fibSeries(50)
```

```
1  
1  
2  
3  
5  
8  
13  
21  
34
```



Variable argument passing

```
def printVariables( arg1, *vararg ):  
    print ("Arguments Received: ")  
    print (arg1 )  
    for v in vararg:  
        print v  
    return;
```

Function calling

printVariables(10)

printVariables(70, 60, 50)



Default arguments

```
def studentInfo( name, fee = 3500 ):  
    print ("Name: ", name)  
    print ("Fee : ", fee )
```

Function calling

```
studentInfo('Amrit' )
```

```
studentInfo('Amrit' ,2000)
```

```
studentInfo( fee=5000, name="Amrit" )
```

```
studentInfo( name="Amrit" )
```




Anonymous Functions

- anonymous functions are not declared using the *def* keyword
- uses *lambda* keyword
- Syntax : *lambda* [arg1 [,arg2,.....argn]]:expression
- Example: `sum = lambda arg1, arg2 : arg1 + arg2;`
- Calling: `print "Value of total : ", sum(10, 20)`

```
1 def add(a,b):  
2     return a+b  
3  
4 print(add(10,20))  
5  
6 l=lambda a,b:a+b  
7 print(l(10,20))
```



lambda keyword

1. `lst = [10,2,3,21,99,4]`
2. `Result = list (filter (lambda x:x%2==0, lst))`
3. `print(Result)`
4. `for i in Result : print(i)`

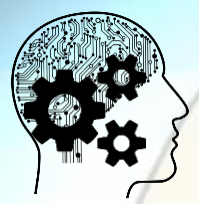
output

`[10,2,4]`

`10`

`2`

`4`



- `Lst = [2,3,4,5]`
- `Result = list (map (lambda n:n*2, Lst))`
- `print(Result)`

[4,6,8,10]

- `from functools import reduce`
- `Lst = [2,3,4,5]`
- `Result = reduce (lambda x,y : x+y, Lst))`
- `print(Result)`

14

- The `functools` module provides tools for working with functions and other callable objects, to adapt or extend them for new purposes without completely rewriting them.



Callable

- Callable in simple words – *some things that can be called*
- Returns true if the passed object appears to be callable else false

```
# callable() example
```

```
def testCallable():  
    return 127
```

```
tst = testCallable  
if (tst) : tst ()
```

```
# an object is created of method
```

```
tst = testCallable
```

```
print(callable(tst)) # returns true
```

```
# a test variable
```

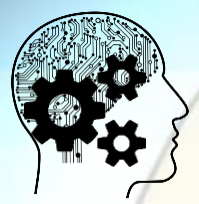
```
num = 4*4
```

```
print(callable(num)) # returns false
```



Exercise

1. Write a function to check the argument passed is part of Fibonacci series or not . `isInFibo(x)` returns true/false
2. Write a function to check the argument passed is prime number. `isPrime(x)` returns true/false
3. Write a program to calculate the arithmetic mean of a variable number of values.
4. Write a function to find letter in a word `def find(word, letter)`
5. Modify above function with third parameter specifying where to start the search `def find (word, letter, start)`

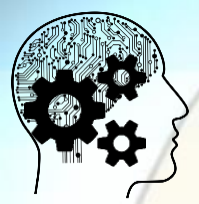


Exercise

1. *ROT13 is a weak form of encryption that involves “rotating” each letter in a word by 13 places. Write a function to encrypt-decrypt passed text according to ROT13.*

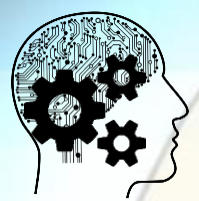
rotCipher(string, 'e') / rotCipher(string, 'd')

1. Write a function to check whether passed string is palindrome or not : *isPalindrome(word)*



Exercise

1. Write command to check whether input number is even or odd.
2. Write a command/program to accept marks from user and print the division.
3. Write command/s to return sum of digits of given number.



loops

while expression :
 statements()

```
i=0
while (i<5) :
    print (i, 'Jai Ho')
    i=i+1
```

```
0 Jai Ho
1 Jai Ho
2 Jai Ho
3 Jai Ho
4 Jai Ho
```

while expression :
 statements()
else :
 statements()

```
i=0
while (i<5) :
    print (i, 'Jai Ho')
    i=i+1
else :
    print (i, ' Its over now')
```

```
0 Jai Ho
1 Jai Ho
2 Jai Ho
3 Jai Ho
4 Jai Ho
5  Its over now
```




for loop

for iterating Variable in sequence
statement/s

```
In [3]: states=['J&K', 'HimachalPradesh', 'Punjab', 'Delhi']  
for st in states:  
    print (st)
```

```
J&K  
HimachalPradesh  
Punjab  
Delhi
```

```
In [4]: for st in range(len(states)):  
        print (states[st])
```

```
J&K  
HimachalPradesh  
Punjab  
Delhi
```

```
In [5]: for alpha in 'India':  
        print(alpha)
```

```
I  
n  
d  
i  
a
```

for iterating Variable in sequence
statement/s

else:

statement/s

```
In [7]: for st in range(len(states)):  
        print (states[st])  
else :  
        print('-----Its over ----- ')
```

```
J&K  
HimachalPradesh  
Punjab  
Delhi  
-----Its over -----
```



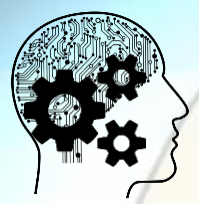
Exercise

1. Write program to check whether given number is prime or not
2. Write a program to find those numbers which are divisible by 7 and multiple of 5, between 1500 and 2700 (both included).
3. Write a Python program to get the Fibonacci series between 0 to 50.
4. Write a program to construct the pattern, using a nested for loop.

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

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

```
1  
22  
333  
4444  
55555  
666666  
7777777  
88888888  
999999999
```



Loop Control Statements

Break : Terminates loop statement

```
for alpha in 'Greatness':  
    if alpha == 'n':  
        break  
    print ('letter ', alpha)
```

```
letter G  
letter r  
letter e  
letter a  
letter t
```

continue : returns the control to the beginning of the while/for loop

```
for alpha in 'Greatness':  
    if alpha == 'n':  
        continue  
    print ('letter ', alpha)
```

```
letter G  
letter r  
letter e  
letter a  
letter t  
letter e  
letter s  
letter s
```

pass : is used when a statement is required syntactically but you do not want any command or code to execute

```
for alpha in 'Greatness':  
    if alpha == 'n':  
        pass  
    print ('Pass block')  
    print ('letter ', alpha)
```

```
letter G  
letter r  
letter e  
letter a  
letter t  
Pass block  
letter n  
letter e  
letter s  
letter s
```



lambda keyword

1. `lst = [10,2,3,21,99,4]`
2. `Result = list (filter (lambda x:x%2==0, lst))`
3. `print(Result)`
4. `for i in Result : print(i)`

output

`[10,2,4]`

`10`

`2`

`4`



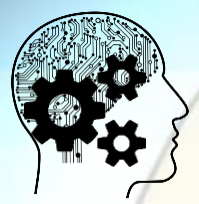
- `Lst = [2,3,4,5]`
- `Result = list (map (lambda n:n*2, Lst))`
- `print(Result)`

[4,6,8,10]

- `from functools import reduce`
- `Lst = [2,3,4,5]`
- `Result = reduce (lambda x,y : x+y, Lst))`
- `print(Result)`

14

- The functools module provides tools for working with functions and other callable objects, to adapt or extend them for new purposes without completely rewriting them.



Callable

- Callable in simple words – *some things that can be called*
- Returns true if the passed object appears to be callable else false

```
# callable() example
```

```
def testCallable():  
    return 127
```

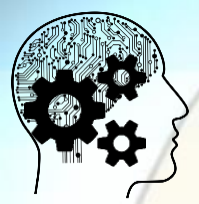
```
# an object is created of method
```

```
tst = testCallable  
print(callable(tst)) # returns true
```

```
# a test variable
```

```
num = 4*4  
print(callable(num)) # returns false
```

```
tst = testCallable  
if (tst) : tst ()
```



Exercise

1. Write a function to check the argument passed is part of Fibonacci series or not . `isInFibo(x)` returns true/false
2. Write a function to check the argument passed is prime number. `isPrime(x)` returns true/false
3. Write a program to calculate the arithmetic mean of a variable number of values.
4. Write a function to find letter in a word `def find(word, letter)`
5. Modify above function with third parameter specifying where to start the search `def find (word, letter, start)`



Exercise

1. *ROT13 is a weak form of encryption that involves “rotating” each letter in a word by 13 places. Write a function to encrypt-decrypt passed text according to ROT13.*

rotCipher(string, 'e') / rotCipher(string, 'd')

1. Write a function to check whether passed string is palindrome or not : *isPalindrome(word)*