1) Methods ⇒ functions (i.e) built into objects.

it will perform specific actions on the object and can also take arguments, like a function.

syntax :-

object.method name (arg1, arg2, ... etc)

↓

variable.

8) Methods and Functions.

2) Functions :- groups together a set of statements so they can be run more than once.

'def' is a built-in keyword telling Python you are about to create a function.

```
def name_of_function ():          # syntax.
    pass
```

eg:-
```
def my_addition_func (arg1, arg2)
    print arg1 + arg2.
```

```
my_addition_func (1, 2)
O/P = 3.
```

eg:-
```
def add_num (num1, num2):
    return num1 + num2
```

```
x = add_num (2, 3)
x
O/P = 5.
```

eg:-
```
def is_prime (num):
    for n in range (2, num):
        if
```

Eg:- def is-prime(num):
    for n in range (2, num):
        if num % n == 0:
            print 'Not Prime'
            break
    else:
        print 'The number is Prime'

is-prime (12)
O/P = Not Prime
is-prime (13)
O/P = The number is Prime

3) Lambda expressions → allows us to create "anonymous" functions
simple def function

Eg:- i) def square (num):
        result = num**2
        return result

square (2)
O/P = 4.

(ii) def square (num):
        return num**2

square (4)
O/P = 14.

(iii) def square(num): return num**2
square(4) # lambda expression.
o/p=16.

(iv) lambda num: num**2. # lambda function

- we usually don't save the lambda expression to a variable
  (i.e) square=lambda num:num**2
  square(10)
  o/p=100.

as we want an anonymous function which means we don't save it we directly run it.

- when we just want to quickly perform some sort of expressions & then don't worry about it later.

(v) adder = lambda x,y: x+y
adder(30, 20)
o/p = 50.

4) Nested Statements and Scope.

- When we create a variable name in Python the name is stored in a name space.
- Variable names also have a scope, the scope determines the visibility of that variable name to other parts of the code.

eg:-
```
x = 25
def printer():
    x = 50   <= # local scope.
    return x.
print x.
print printer().
```

```
print x
o/p = 25
print printer()
o/p = 50.
```

* Local Variables.

- When we declare variables inside a function definition, they are not related in any way to other variables with the same names used outside the function (i.e) variable names are local to the function.
- This is called the scope of the variable. All variables have the scope of the block they are declared in starting from the point of definition of the name.

eg:- (i) x = 50 → # global variable

```
def func(x):
    print 'x is', x.
    x = 2.  → # local changes x to 2
    print 'Changed local x to', x.

func(x)  → # outside the function x is again = 50 (i.e) global value
print 'x is still', x.
```

o/p =

x is 50.
Changed local x to 2.
x is still 50.

(ii)

```
name = 'This is a global name'
def greet():
    # enclosing (nesting) function.
    name = 'Sammy'
    def hello():
        print 'Hello' + name.
    hello().
greet()
```

o/p = . Hello Sammy .

- If we want to assign a value to a name defined at the top level of the program (i.e not inside any kind of scope such as functions or classes), then you have to tell python that the name is not local, but it is global.

classmate
Date
Page

will be done
whatever changes here will affect x globally.

eg:- (iii)

```
x = 50
def func():
    global x    → # this tells we are working an global variable x & not local
    print 'This function is now using the global x!'
    print 'Because of global x is', x.
    x=2
    print 'Ran func(), changed global x to', x.
print 'Before calling func(), x is', x.
func()
print 'Value of x (outside of func()) is', x
```

O/P = Before calling func(), x is :50
This function is now using the global x!
Because of global x is :50.
Ran func(), changed global x to 2.
Value of x (outside of func()) is :2.

- We can use the built-in function globals() and locals() functions to check what are the current locals and global variables.

* Name Assignment – will create or change local names by default.

Name references search (at most) 4 scopes
(i.e) (i) local.
(ii) enclosing functions
(iii) global.
(iv) built-in.

(i) Local scope. – names assigned in any way within a function such as def or lambda, and not declared global in that function.
(ii) Enclosing function locals – name in the local scope of any and all enclosing functions (nested function) from inner to outer
(iii) Global scope: Name assigned at the top level of a module file, or declared global in a def within a file.
(iv) Built-in (Python) – Names preassigned in the built-in names module: such as open, range, SyntaxError, ... etc.