DATE ___/___/___

① => functions are associated with the independent of objects/classes, whereas Method are NOT Associated with objects/classes.

Function
Invocation = function_name() | object.method | class.method()
Method.

* NO access to object/class data | Can access object/class data.

* Def keyword | Part of class definition

② => Arguments are the actual values supplied to the function while it is called. These values are assigned to the corresponding parameters.

Parameters are the variables defined in a function's Signature (definition). They act as placeholders for the values that the function will receive when it is called.

DATE / /

(3)

=> Standard function Definition and call
def
↓
* for passing arguments.

(2) function with default parameters.
* Assign default values to parameters in the function signature.
↓
you can omit arguments for parameters with default values

(3) function with variable positional Arguments (*args)
↓
To accept an arbitrary number of positional arguments.

call: Pass any number of positional arguments.

(4) function with variable keyword Arguments (**kwargs).
↓
To accept variable of keywords.
↓
Pass keyword arguments.

DATE ___/___/___

⑤ Lambda functions (Anonymous functions)

↓

to create small, anonymous functions

↓

Assign the lambda to a variable / call it directly.

⑥ Recursive function :- A function that calls itself

↓

invoke the function with a base case to prevent infinite recursion.

⑦ functions as an argument

↓

Passing a function as an argument to another function.

Call :- Invoke the outer function and pass the inner function as an argument.

⑧ Return Statement :- To return a value from the function.

↓

Capture the return value in a variable for use it directly.

(4)

⇒ The return statement in a python function serves the purpose of sending a value result back to the caller of the function. It allows the function to produce and output that can be used elsewhere in the program.

(5)

⇒ Iterators are the internal process done during the iteration to an iterables, Iterators are basically in core form of the iteration/for-ables which gives the values one-by-one.

(6).

⇒ Generators are a simple way to create iterators using function and the yield keyword instead of returning values. Generators produce values one at a time only when requested, reducing memory usage.

(7)

=> generator function can give the output very fast and it also needs low memory space to do the calculation. The execution time is fast as regular functions, it directly give the calculations and also has an simpler syntax.

(8)

=> lambda function is an anonymous function to create quick output with smaller and simpler syntax, they are used when we want to do small throwaway functions.

(9).

=> Map function helps in mapping each and every element of the iterable, it takes iterable as input and customise it.

(10)

=> map() function applies a given function to each element of an iterable (e.g. list tuple) and returns a map object (an iterator) containing the transformed elements.

=> reduce() function, from the functools module, applies a given function cumulatively to the elements of an iterable, reducing it to a single value.

=> filter() function applies a predicate function that return True / false to each element of an iterable and returns an iterator containing only the elements for which the function return true.

(10)

=> Internal mechanism.

* Initial list: [47, 11, 42, 13].
* lambda function: lambda x, y: x+y
  * This function takes two values x and y and returns their sum.

* first Iteration
  Input x=47, y=77
  operation: 47+11=58.

* Second Iteration
  x = 58 (from first iteration), y=42
  Operation: 58+42=100

* third iteration
  x = 100 (from second iteration), y=13.
  Operation 100+13 = 113 final ans