→ Python functions

a. Script written away from main script
b. Use
    i. Modularity
    ii. reusability

c.
```
def name (parameter list):
        # code block
```

→ Types of functions

	return	
Parameter	Without parameter Without return (not used mostly)	With parameter Without return
	Without parameter With return (very less used)	With parameter With return

→ Document string

a. Optional
b. Tells about function
c. Only for user
d. 1st line after def –abc–()

→ by default, python supports data type based overloading

→ nesting function calls can only be used through returning functions only

→ python used “pass by value” for its primitive data types, but it uses “pass by reference” for data structures

→ pass by value: creates copy of original data, original data is not affected

→ pass by reference:uses reference to original data, original data is affected

→ recursion function in python

a. Not first option
b. Two advantages
    i. Better readability
    ii. Code looks smaller
c. Three disadvantages
    i. Code is complicated
    ii. More execution time
    iii. Consumes more memory space

→ return

a. gives control, data, or both, back to other script
b. Can return more than one element from function, but it returns all the multiple elements as a tuple

→ key-value parameter:

a. mechanism of python to allow parameters in any order, but with precise keys ()this feature is activated from calling

→ default-parameter:

a. default-values are specified in definition, and are accepted, only when an argument value is not specified in the function call.
b. If any specific value needs to be mentioned, use key-value based parameter in function call

→ variable size parameter:

a. Mentioned in function definition
b. All parameters are recorded in a tuple and set
c.

Iterator, Generator & Decorator

→ Iterator:

a. Iterates over a bunch of values, but in one at a time manner
b. Has two methods
    i. iter():
    ii. next()
c. iterator has a limitation of supporting sequence, it is recovered from generator

→ generator:

a. Yield, stop, break manner
b. It is a function, which runs in partial manner everytime next() is called
c. Internally function carries yield statement, which breaks the function & returns the value

→ decorator

a. Works on polymorphic functions/function polymorphism, which is ability of giving alternate identifier to existing functions
b. In this, we have internal old function, which is re-used by external decorator function, and address using function polymorphism

→ lambda functions

a. It is anonymous function, whcih means function without identifier/name
b. It gets attached to a variable, which will be used to call
c. Lambda function cannot be of more then one line, it can only be of one line
d. Throw away functions
e. Anonymous functions
f. Memory efficient

→ loca

a. Local membership:
    i. are variables within function, to be accessed locally(has higher precedence than global)
    ii. If a local function tries to operate on global data, then scope resolution is needed, otherwise error would be caused

b. Global membership:
    i. can be accessed by anybody
c. Membership
d. membership

→

	class	object
	idea	implementation
	virtual	physical
	framework	Code implementation to fill framework
	properties	values
	Methods on properties-access	Operation on values, access to set & reset

→ four pillars of OOP are below

1. Encapsulation:
    a. process of combining data & operations
    b. modularity
    c. Ease of use
    d. re-useability
2. Abstraction:
    a. Hiding complexity and provide simple acccess via methods
3. Inheritance:
    a. Ability to derive property from pre-existing one
    b. Advantages of inheritance:
        i. Standardization
        ii. Enhancement
        iii. Faster development
4. polymorphism:
    a. Ability to exist in many forms
    b. Overload: support multiple data types
    c. Override: ability to re-create pre-existing methods to get a re-code

→ in python class

a. names start with a capital letter
b. can have document string

c. We need method calls for creating properties within object
d. One class can create ‘n’ objects
e. Object has reference, which is passed to every method call using ‘self’ variable
f. Anything performed on self (by self.) will create property in the object whose reference is stored in ‘self’
g. ‘.’ membership operator, indicates membership

→ constructor method

a. A method which automatically gets activated when object is created/initiated
b. constructor method is created as

```
def __init__(self):
      #body
```

→ printer method

a. Method automatically gets triggered when object of class is passed as an argument to print(), it will return only a string
c. Printer method is created as

```
def __str__(self):
      #body
```

It will return string

d. asa