OOPS concept			
Sr.No.	**Syntax/command**	**Output**	**Remarks**
1	__init__(self):		Initilization of a particular variable to class
2	self.name = name		Public variable
3	self._surname = surname		Protected variable (Encapusulation)
4	self.__yob = yob		Private variable (Encapusulation)
5	print(adit._person__name)		You can only access this private variable by appending the class name before the variable
6	import test1		For accessing code from one python file into another
7	from test2 import person		For importing specific class from python file into another
8	from utils.util import person2		For accessing class from a module(py file) inside package(folder)
9	__students = "data science"		This is called **data abstractiion** as we are trying to hide that data behind the local variable. It is the process of hiding the real implementation of an application from the user and emphasizing only on usage of it

10	class ineuron:
	    __students = "data science"
	
	    def students(self):
	        print("print the class of students",ineuron.__students)
	
	i = ineuron()
	i.students()
	print(i._ineuron__students)
			Such data can be accessed only by appending class name before it with single underscore.

	class car :
	
	    def __init__(self, body, engine, tyre):
	        self.body = body
	        self.engine = engine
	        self.tyre = tyre
	
	    def mileage(self):
	        print("Mileage of this car")
	
	class tata(car):
	    pass
			When u want to utilize entire code from one class (parent class) into another class (child class) it Is called **Inheritance.** Here tata class will exactly inherit car class

	class bank :
	
	    def transaction(self):
	        print("Total transaction value ")
	    def account_opening(self):
	        print("This will show you your account opening status")
	    def deposit(self):
	        print("This will show you your deposited amount")
	
	class HDFC_bank(bank):
	
	    def HDFC_to_icici(self):
	        print("This will show all the transactions happend to icici from HDFC")
	
	class icici(HDFC_bank):
	    pass
			Multi level inheritance

	class bank:
	
	    def transaction(self):
	        print("Total transaction value ")
	    def account_opening(self):
	        print("This will show you your account opening status")
	    def deposit(self):
	        print("This will show you your deposited amount")
	    def test(self):
	        print("This is test method from bank")
	
	class HDFC_bank:
	
	    def HDFC_to_icici(self):
	        print("This will show all the transactions happend to icici from HDFC")
	    def test(self):
	        print("This is test method from HDFC bank")
	
	class ineuron_bank:
	
	    def account_status_icici(self):
	        print("Print a account status in icici")
	
	class icici(bank , HDFC_bank, ineuron_bank):
	    pass
			Multple inheritance

11	class ineuron:
	    def __init__(self):
	        self.students1 = "data science"
		data analytics	In run time you can overwrite the value of the variable

	i = ineuron()
	i.students1 = "data analytics"
	print(i.students1)		

12	class ineuron1:
	    def __init__(self):
	        self.__students1 = "data science"
			In run time you cannot change the value of variable that is private but can only be changed if a function is

	i1._ineuron__students1 = "data analystics"
	i1.students1()
			defined to reassign the value. This is called as **encapsulation.** It puts restrictions on accessing variables and methods directly and can prevent the accidental modification of data

	def students_change(self):
	      self.__students1 = "Big data"
	def students_change_to(self, new_value):
	      self.__students1 = new_value		

13	class ineuron:
	
	   def students(self):
	      print("Student details")
	
	class class_type:
	
	   def students(self):
	      print("print the class type of students")
			Polymorphism means multiple forms. It is the property where a single function performs in varied different ways based on the values.

	def ineuron_external(a):
	   a.students()		

	i = ineuron()
	j = class_type()
	ineuron_external(i)
	ineuron_external(j)		

14	a is b		Evaluate whether identifiers a and b are aliases for the same object
	a == b		whether the two identifiers reference equivalent values
14	super(). init (customer, bank, acnt, limit)		Calls the init method that was inherited from the parent class or superclass and is initiated in child class or subclass
15	palette = warmtones		Both alias and subsequently add or remove colors from “palette,” we modify the list identified as warmtones.
16	palette = list(warmtones)		This will create a ***shallow copy*** of warmtones
17	palette = copy.deepcopy(warmtones)		In ***deep copy*** new copy references its own copies of those objects referenced by the original version.