[126]:

```python
# DISCLAIMER : python not supports fully object oriented programming language.

# how to create a class in Python.
# - using class keyword
# - basically, the class name mostly starts with the first letter.
class Sample:
    pass
class Animal:
    pass
```

[ ]:

[127]:

```python
# How to create an object
# - object is also called as instance
class Sample:
    pass

obj1 = Sample()
obj2 = Sample()

# ------------------------------------------

class Ship:
    pass
black_pearl = Ship()
titanic = Ship()
# black_pearl and titanic is an object
```

[128]:

```python
# functions in Python class are called methods
```

[129]:

```python
# How to declare methods inside a class
#  3 types of methods in class
#     - instance method
#     - class method
#     - static method
class Sample:
    def __init__(self): # special method, called constructor,   By default we have make constructor as instance method.
        pass            # executes while object is creating.

    def fn1(self): # instance method    : only access by objects/instance
        pass

    @classmethod
    def fn2(cls):     # class method       : can access by class. which mean common to all objects.
        print(cls)

    @staticmethod
    def fn3():        # static method      : considrcommon to object
        pass
```

[130]:

```python
# 2 types of variables in class
# - class variable
# - instance variable

class Sample:
    v2 = 100 # class variable
    def __init__(self,v1):
        self.v1 = v1 # instance variable

    def fn1(self,a,b):
        print(a+b+self.v1)

    @classmethod
    def fn2(cls,x,y):
        print(x*y*cls.v2)

    @staticmethod
    def fn3(a,b,c):
        print(a+b+c)

obj = Sample(10)
obj.fn1(2,3)
obj.fn2(2,3)
obj.fn3(1,2,3)

print("----------------------------------------")

# Sample.fn1(2,3) # ERROR: as need an object/instance to call instance method.
Sample.fn2(2,3)
Sample.fn3(1,2,3)
# How output came.
```

```
15
600
6
----------------------------------------
600
6
```

[131]:

```python
# How to declare the variables in class
#  2 types of variables in class
#     - instance variable
#     - class variable
class Sample:
    b = 123 # class variable : created inside class
          #  common to all objects.
    def fn(self):
        # You can create variables in any functions.
        # Mostly they are created inside function.
        self.a = 10   # instance variable  : created using lass_name
```

[132]:

```python
# constructor
# instance variable
class Dog:
    breed = "german-sheperd"
    def __init__(self, name, age):
        self.name = name
        self.age = age
d1 = Dog("shiro",5)
d2 = Dog("scooby",10)

# ------------------------------------------------
#  Things to be known before proceeding.
#
#   - HOW TO ACCESS variables / methods
#     [IMPORTANT] using dot(.)

#   - class variables are common to all objects.
#   - can be accessed using, "self", "object", "class_name"
#
#   - instance variables are unique to its object.
#   - can be accessed using, "self", "object"

#   [ NOTE ] : self is nothing but refers to object.

print(d1.breed, d1.name, d1.age)
print(d2.breed, d2.name, d2.age)
print(Dog.breed)
# print(Dog.name) # not possible because,
#    name, age is a instance variable, only access by instance.
```

```
german-sheperd shiro 5
german-sheperd scooby 10
german-sheperd
```

[ ]:

[133]:

```python
# -----------------
#   INHERITANCE
# -----------------

class Parent:
    a = 10
    b = 20

class Child(Parent): # by keeping parent class name in paranthesis
    pass

obj = Child()
print(obj.a)
print(Child.a)
```

```
10
10
```

[134]:

```python
# type of inheritance
# 1. Single Inheritance: A class inherits from one superclass/Parentclass.
class Parent:
    def __init__(self):
        self.value = "Parent"

    def display(self):
        print(self.value)

class Child(Parent):
    def __init__(self):
        super().__init__() # always using this line,
        # becuase, if you not using this, parent constructor will not get called
        self.value = "Child"

obj = Child()
obj.display()  # Output: Child
```

```
Child
```

[135]:

```python
# 2. Multiple Inheritance: A class inherits from more than one superclass.
class ClassA:
    def method_a(self):
        print("ClassA method")

class ClassB:
    def method_b(self):
        print("ClassB method")

class ClassC(ClassA, ClassB):
    def __init__(self):
        pass
obj = ClassC()
obj.method_a()  # Output: ClassA method
obj.method_b()  # Output: ClassB method
```

```
ClassA method
ClassB method
```

[136]:

```python
# 3. Multilevel Inheritance: A class inherits from a superclass, which itself inherits from another superclass.
```

[137]:

```python
class Grandparent:
    def __init__(self):
        self.grandparent_value = "Grandparent"

class Parent(Grandparent):
    def __init__(self):
        super().__init__()
        self.parent_value = "Parent"

class Child(Parent):
    def __init__(self):
        super().__init__() # you have to use this. if you want to call parent class constructor.
        # if not, python will only call child constructor.
        #
        self.child_value = "Child"

obj = Child()
print(obj.grandparent_value)  # Output: Grandparent
print(obj.parent_value)       # Output: Parent
print(obj.child_value)        # Output: Child
```

```
Grandparent
Parent
Child
```

[138]:

```python
# 4. Hierarchical Inheritance: Multiple classes inherit from the same superclass.
class Parent:
    def __init__(self):
        self.value = "Parent"

class Child1(Parent):
    pass

class Child2(Parent):
    pass

obj1 = Child1()
obj2 = Child2()
print(obj1.value)  # Output: Parent
print(obj2.value)  # Output: Parent
```

```
Parent
Parent
```

[139]:

```python
# 5. Hybrid Inheritance: A combination of two or more types of inheritance.
class ClassA:
    def method_a(self):
        print("ClassA method")

class ClassB(ClassA):
    def method_b(self):
        print("ClassB method")

class ClassC:
    def method_c(self):
        print("ClassC method")

class ClassD(ClassB, ClassC):
    pass

obj = ClassD()
obj.method_a()  # Output: ClassA method
obj.method_b()  # Output: ClassB method
obj.method_c()  # Output: ClassC method
```

```
ClassA method
ClassB method
ClassC method
```

[ ]:

[ ]:

[ ]:

[140]:

```python
# access specifiers

class Parent:
    a = 10     # public     :  can access in class, subclass, object
    _b = 20   # protected :  can access in class, subclass
    __c = 30 # private    :  can access in class
    def __init__(self):
        print(self.a)
        print(self._b)
        print(self.__c)
class Child(Parent):
    def __init__(self):
        super().__init__()
        print(self.a)
        print(self._b)
        # print(self.__c) # not accessable
obj = Child()
print(obj.a)
print(obj._b) # still accessable. as python partially implements oops concept
# print(obj.__c) # not accessable
```

```
10
20
30
10
20
10
20
```

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[141]:

```python
# Data abstraction
# is a concept in object-oriented programming that focuses on
# hiding the implementation details of an object
# and exposing only the necessary features to the outside world.
# This helps in reducing complexity and increases the modularity of the code.

# In Python, data abstraction can be achieved using abstract classes and abstract methods provided by the abc module.

from abc import ABC, abstractmethod

class Animal(ABC):
    '''
     Here ,
     - you cannot create object for Animal class
     - If you inherit this Animal class,
     -    then first you have write some logic for below abstractmethod ("make_sound")
    '''
    @abstractmethod
    def make_sound(self):
        pass


# here abstract method tells that if you inherit

class Dog(Animal):
    def make_sound(self):
        return "Bark"

class Cat(Animal):
    def make_sound(self):
        return "Meow"

# Creating instances of the concrete classes
dog = Dog()
cat = Cat()

print(dog.make_sound())  # Output: Bark
print(cat.make_sound())  # Output: Meow
```

```
Bark
Meow
```

[ ]:

[ ]:

[ ]:

[142]:

```python
# Encapsulation definition with example
# Encapsulation is one of the fundamental principles of object-oriented programming.
# It refers to the binding of data (attributes) and methods (functions) in a single place.

class Parent:
    a = 10      # Public    : can be accessed in class, subclass, and object
    _b = 20     # Protected : can be accessed in class and subclass
    __c = 30    # Private   : can be accessed only in class

    def __init__(self):
        print(self.a)      # Accessing public attribute
        print(self._b)     # Accessing protected attribute
        print(self.__c)    # Accessing private attribute

class Child(Parent):
    def __init__(self):
        super().__init__()    # Calling the parent class constructor
        print(self.a)         # Accessing public attribute
        print(self._b)        # Accessing protected attribute
        # print(self.__c)     # Not accessible (uncommenting this will cause an error)

# Creating an instance of Child class
obj = Child()

# Accessing attributes through the object
print(obj.a)      # Public attribute, accessible
print(obj._b)     # Protected attribute, accessible but not recommended
# print(obj.__c) # Private attribute, not accessible (uncommenting this will cause an error)

# Encapsulation helps to protect the internal state of an object and prevents unintended
# interference and misuse. By controlling access to the data through methods and access specifiers,
# we can ensure that the object's data remains consistent and valid.
```

```
10
20
30
10
20
10
20
```

[ ]:

[ ]:

[ ]:

[ ]:

[143]:

```python
# polymorphism
# 2 ways we can achieve
#     1. overloading
#     2. overriding

class Parent:
    def fn(self):
        print("inside parent")

class Child(Parent):
    def fn(self): # this function overrides fn() in parent class
        print("inside child")

    def logic(self,a,b=10,*args): # method overloading
        print(a,b,*args)

obj = Child()
obj.fn()

# overloading means, same method will respond diff for diff values
obj.logic(1)
obj.logic(1,2)
obj.logic(1,2,3)
```

```
inside child
1 10
1 2
1 2 3
```

[ ]:

[ ]:

[ ]:

[ ]:

[144]:

```python
# Implement a calculator using class and objects


class Calculator:
    def add(self, a, b):
        return a + b

    def subtract(self, a, b):
        return a - b

    def multiply(self, a, b):
        return a * b

    def divide(self, a, b):
        if b != 0:
            return a / b
        else:
            return "Error! Division by zero."

# Creating an instance of the SimpleCalculator class
calc = Calculator()

# Performing some calculations
print("Addition:", calc.add(10, 5))           # Output: 15
print("Subtraction:", calc.subtract(10, 5))   # Output: 5
print("Multiplication:", calc.multiply(10, 5)) # Output: 50
print("Division:", calc.divide(10, 5))         # Output: 2.0
print("Division by zero:", calc.divide(10, 0)) # Output: Error! Division by zero.
```

```
Addition: 15
Subtraction: 5
Multiplication: 50
Division: 2.0
Division by zero: Error! Division by zero.
```

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]: