OOPS - Object oriented programming

Class	Classification of real world entity
Object	Represents the real world entity
Constructor	entity/function/default method by which you can give data/information to class.

Advantages of OOPs

Clarity of code.
Proper segmentation of the code
Reusability of the code
Cleanness inside code
Structure inside code

Object-Oriented Design Principles

Modularity
Modularity refers to an organizing principle in which different components of a software system are divided into separate functional units.

Abstraction
The notion of abstraction is to distill a complicated system down to its most fundamental parts.

Encapsulation
Encapsulation refers to using software system that not reveal the internal details of their respective implementations Eg. Single underscore character (e.g., secret) are assumed to be nonpublic

Software Development

Traditional software development involves several phases. Three major steps are:
1. Design
2. Implementation
3. Testing and Debugging

Design

Algorithms in a way that is intended for human eyes only are called **pseudo-code**.

Class diagram.
A standard approach to explain and document the design using class diagrams to express the organization of a program

Class:	CreditCard	
Fields:	customer	balance
	bank	limit
	account	
Behaviors:	get customer( )	get balance()
	get bank()	get limit( )
	get account()	charge(price
	make payment(amount)	

Coding Style and Documentation

The main principles that we adopt are as follows:

• Python code blocks are typically indented by 4 spaces. It is strongly recommended that tabs be avoided, as tabs are displayed with differing widths across systems, and tabs and spaces are not viewed as identical by the Python interpreter

• Use meaningful names for identifiers

> **Classes** (other than Python's built-in classes) should have a name that serves as a singular noun, and should be capitalized (e.g., CreditCard).
>
> **Functions**, including member functions of a class, should be lowercase. be separated by underscores. (e.g., makepayment)
>
> **Names** that identify an individual **object** (e.g., a parameter, instance variable, or local variable) should be a lowercase noun (e.g., price).
>
> **Identifiers** that represent a value considered to be a **constant** are traditionally identified using all capital letters and with underscores to separate words (e.g., MAX SIZE).

• Use comments that add meaning to a program and explain ambiguous or confusing constructs

Documentation

Python provides integrated support for embedding formal documentation directly in source code using a mechanism known as a **docstring.**
By convention, those string literals should be delimited within triple quotes (”””).

Testing and Debugging

Testing is the process of experimentally checking the correctness of a program

Debugging is the process of tracking the execution of a program and discovering the errors in it

1. Testing

There are two main testing strategies,
top-down and bottom-up,

Top-down testing proceeds from the top to the bottom of the program hierarchy

if name == __main__ : *# perform tests...*
to test the functionality of the functions and classes specifically defined in that module.

More robust support for automation of unit testing is provided by Python's **unittest module**.
This framework allows the grouping of individual test cases into larger test suites, and provides support for executing those suites, and reporting or analyzing the results of those tests using **regression testing**

2. Debugging

The simplest debugging technique consists of using **print statements** to track the values of variables during the execution of the program.

Other approach is using **debugger** The basic functionality provided by a debugger is the insertion of **breakpoints** within the code

Inheritance

Inheritance allows a new class to be defined based upon an existing class as the starting point. In object-oriented terminology, the existing class is typically described as the base class, parent class, or superclass, while the newly defined class is known as the subclass or child class.

Protected Members

Protected or Private access modes.
Members that are declared as **protected** are accessible to subclasses, but not to the general public, while members that are declared as **private** are not accessible to either.

Abstract Base Classes

A class is called abstract base class if its only purpose is to serve as a base class through inheritance.
Eg. Progression class, which serves as a base class for three distinct subclasses: ArithmeticProgression, GeometricProgression,and FibonacciProgression

Nested Classes

```
class A: # the outer class
   class B: # the nested class
```
nest one class definition within the scope of another class.

Dot operator

Python interpreter begins a name resolution process, described as follows:
1. The instance namespace is searched; if the desired name is found, its associated value is used.
2. Otherwise the class namespace, for the class to which the instance belongs,is searched; if the name is found, its associated value is used.
3. If the name was not found in the immediate class namespace, the search continues upward through namespace the inheritance hierarchy, checking the class for each ancestor (commonly by checking the superclass class, then its superclass class, and so on). The first time the name is found, its associate value is used.
4. If the name has still not been found, an AttributeError is raised

Inheritance	Multiple	class bank:
		class HDFC_bank:
		class icici(bank , HDFC_bank):
		    pass
			In case both hdfc_bank() and bank() class have the same defined function then that function will be executed based on the order mentioned in icici() i.e function which is mentioned first in the order will be executed.

	Multi-layer	class bank :
		class HDFC_bank(bank):
		class icici(HDFC_bank):
		    pass
			when same function is mentioned in both bank() and HDFC_bank(bank) then function from HDFC_bank(bank) will overrite the function from bank().

	class objects modules		

packages
constructor
inheritance
private public protected
abstraction
encapsulation
polymorphism