OOP in Python - Summary

1. Class and Object

A class is a blueprint for creating objects. An object is an instance of a class.

Syntax:

```
class Person:
    def __init__(self, name):
        self.name = name

p = Person("Alice")
print(p.name)
```

2. Encapsulation and Access Modifiers

Encapsulation restricts direct access to variables.

Access Modifiers in Python:

- Public: self.var
- Protected: self._var (convention)
- Private: self.__var (name mangling)

Use getters and setters to access private data safely.

3. Inheritance

Inheritance allows one class to acquire the properties and methods of another.

Types: Single, Multilevel, Hierarchical, Multiple

```
class Animal:
    def speak(self): print("Animal speaks")

class Dog(Animal):
    def speak(self): print("Dog barks")
```

OOP in Python - Summary

4. Polymorphism

Polymorphism allows same method to behave differently for different classes.

```
def sound(animal): animal.speak()
sound(Dog())  # Dog barks
sound(Cat())  # Cat meows
```

5. Abstraction

Abstraction hides complexity using abstract base classes.

```
from abc import ABC, abstractmethod
class Shape(ABC):
    @abstractmethod
    def area(self): pass
```

6. Property Decorator

@property is used to create getters/setters in Pythonic way.

```
class Employee:
    @property
    def salary(self): return self.__salary

    @salary.setter
    def salary(self, val): self.__salary = val
```