🐍 Python OOP Cheat Sheet 🐍

---

1. What is Object-Oriented Programming (OOP)?

- **OOP** is a programming paradigm based on the concept of "objects" that combine **data (attributes)** and **methods (functions)**.
- Key principles:
  - **Encapsulation**: Group related data and methods.
  - **Inheritance**: Reuse and extend existing code.
  - **Polymorphism**: Use the same interface for different data types.
  - **Abstraction**: Hide complex implementation details.

---

2. Classes and Objects 🏰

Define a Class

```
class Dog:
    def __init__(self, name, breed):  # Constructor
        self.name = name
        self.breed = breed
```

Create an Object

```
my_dog = Dog("Buddy", "Golden Retriever")
print(my_dog.name)  # Output: Buddy
```

---

3. Methods in a Class 📜

Instance Methods

- Operate on the **object instance** and can access instance attributes.

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

    def bark(self):  # Instance method
        print(f"{self.name} says Woof!")
```

Class Methods 🏫

- Use `@classmethod` decorator.
- Operate on the **class itself**, not on individual objects.

```
class Dog:
    species = "Canis lupus"  # Class attribute

    @classmethod
    def get_species(cls):  # Class method
        return cls.species
```

Static Methods ⚙️

- Use `@staticmethod` decorator.
- Do **not** operate on instance or class. Acts like a regular function in the class.

```
class Dog:
    @staticmethod
    def sound():
        print("Dogs bark!")
```

---

4. Inheritance 👪

- Reuse and extend existing classes.

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

    def speak(self):
        return "I make a sound."

class Dog(Animal):  # Inherit from Animal
    def speak(self):
        return "Woof!"

dog = Dog("Buddy")
print(dog.speak())  # Output: Woof!
```

---

5. Abstract Classes 🔍

- Use the `abc` module for defining **abstract classes**.
- Cannot be instantiated.
- Force subclasses to implement specific methods.

```python
from abc import ABC, abstractmethod

class Animal(ABC):
    @abstractmethod
    def speak(self):
        pass

class Dog(Animal):
    def speak(self):
        return "Woof!"
```

---

6. Polymorphism 🎮

- Objects of different classes can share the same interface.

```python
class Cat:
    def speak(self):
        return "Meow!"

class Dog:
    def speak(self):
        return "Woof!"

animals = [Cat(), Dog()]
for animal in animals:
    print(animal.speak())
# Output:
# Meow!
# Woof!
```

---

7. Special (Magic) Methods 🧒

- Begin and end with double underscores (`__`).

Method	Purpose
`__init__`	Initialize attributes for a new object.
`__str__`	String representation of the object.
`__len__`	Length representation (for `len()`).
`__getitem__`	Indexing (`obj[key]`).
`__add__`	Overload `+` operator.

```python
class Vector:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __add__(self, other):
        return Vector(self.x + other.x, self.y + other.y)

v1 = Vector(2, 3)
v2 = Vector(1, 4)
result = v1 + v2  # Overloads `+` operator
print(result.x, result.y)  # Output: 3, 7
```

---

8. Encapsulation 🔒

- Control access to attributes using **private** (`_attr`) or **protected** (`__attr`) attributes.

```python
class Dog:
    def __init__(self, name):
        self._name = name  # Protected attribute

    def get_name(self):
        return self._name  # Getter

    def set_name(self, name):
        self._name = name  # Setter
```

9. Key OOP Concepts with Examples 🧠

1. Single Inheritance

```python
class Parent:
    def greet(self):
        print("Hello from Parent!")

class Child(Parent):
    pass

child = Child()
child.greet()  # Output: Hello from Parent!
```

2. Multiple Inheritance

```python
class A:
    def say_a(self):
        print("A says hello!")

class B:
    def say_b(self):
        print("B says hello!")

class C(A, B):  # Inherit from A and B
    pass

c = C()
c.say_a()  # Output: A says hello!
c.say_b()  # Output: B says hello!
```

3. Multilevel Inheritance

```python
class Grandparent:
    def message(self):
        print("Message from Grandparent")

class Parent(Grandparent):
    pass

class Child(Parent):
    pass

child = Child()
child.message()  # Output: Message from Grandparent
```

---

10. Best Practices for OOP in Python 

1. Use **meaningful names** for classes and methods.
2. Use **inheritance sparingly**; prefer composition if possible.
3. Keep methods **short and focused**.
4. Use **private and protected attributes** to avoid accidental modifications.
5. Add **docstrings** to classes and methods.
