Python OOP Cheatsheet

1 Classes and Objects

- **Class**: A template for creating objects, like a blueprint for a toy.
- **Object Class**: The parent class for all Python classes, giving them basic features.
- **Object / Instance**: A thing made from a class, like a toy built from the blueprint.
- **Instantiation**: The process of creating an object from a class.
- **self**: A way to refer to the object itself inside a class.
- **cls**: A way to refer to the class itself in class methods.
- **Meta Class**: A special class that controls how other classes are created.
- **Constructor**: A special function that sets up a new object.
- **Destructor**: A special function that cleans up when an object is deleted.

2 Attributes

- **Instance Variable**: A piece of data unique to each object.
- **Class Variable / Static Variable**: A piece of data shared by all objects of a class.
- **Public Variable**: Data anyone can see or change.
- **Protected Variable**: Data for use inside the class or its children. (convention: single underscore)
- **Private Variable**: Data hidden from outside the class. (convention: double underscore)

3 Methods

- **Instance Method**: A function that works with a specific object.
- **Class Method**: A function that works with the class itself.
- **Static Method**: A function in a class that doesn't need an object or class.
- **Abstract Method**: A function that must be written in child classes.

4 Inheritance and Polymorphism

- **Inheritance**: A way for a class to get features from another class.
- **Parent Class / Superclass / Base Class**: The parent class that shares its features.
- **Child Class / Subclass / Derived Class**: A class that gets features from a parent class.
- **super()**: A way to call functions from the parent class.
- **Abstract Class**: A class you can't make objects from, used as a template.
- **Polymorphism**: When child classes change how parent functions work.
- **Method Resolution Order (MRO)**: The order Python looks for functions in classes.
- **Multiple Inheritance / Diamond Inheritance**: A class getting features from more than one parent.

5 Encapsulation

- **Encapsulation**: Keeping data and functions together, hiding some details.
- **Access Modifiers**: Rules about who can see or change data (public, protected, private).
- **Property Decorators**: Special tools to control how data is used.

6 Composition and Aggregation

- **Composition**: A class that includes another class as a part. (e.g. car and engine)
- **Aggregation**: A class that uses another class but doesn't own it. (e.g. team and player)

7 Callable

- **__call__**: A special function that lets you use an object like a function.
- **callable()**: A way to check if something can be used like a function.

8 Iterable

- **__iter__**: A special function that starts a loop.
- **__next__**: A special function that gets the next item in a loop.

9 Decorators

- **Function Decorator**: A tool that adds extra steps to a function.
- **Class Decorator**: A tool that adds extra features to a class.

10 Exception Handling

- **Custom Exception**: A special error you create for your program.
- **try...except**: A way to catch and handle errors safely.

11 Dunder Methods

Dunder methods are special functions that let your classes work with Python's built-in features, like printing or looping.

- **__init__**: Sets up a new object, like a constructor.
- **__str__**: Defines how an object looks when printed as text.
- **__call__**: Lets an object act like a function when called.
- **__iter__**: Makes an object usable in a loop.
- **__next__**: Gets the next item in a loop from an object.
- **__del__**: Cleans up an object when it's deleted, like a destructor.

Made by Abbas Asad