OOP Classes

Classes are the most important part of OOP as they allow us to meet all 4 principles of OOP.

Important things to know

- Keywords and Terms
- Naming convention
- Creating attributes
  - Outside of the method
  - Inside a method
- Creating methods
- Access Modifiers

Terms

`class`

- A blueprint for an object
- defines the methods and attributes required for an object to correctly operate

`object`

- An instance of a class
- All variables in Python are objects
- Think of a custom class like a variable
  - Represents a single item
  - Stores data
  - Can be passed around

`method`

- A fancy term for a function that exists within a class
- Used to perform operations
- takes the `self` keyword as a parameter in order to access attributes and other methods within the class

`attribute`

- A fancy term for a variable that exists within a class
- Used to store data about the object
- Needs to be created using the `self` keyword if it is being created within a method, else it can be defined normally

Keywords

`dunder`

- Double underscore
- Methods that perform operations based on certain actions that are performed on the class
    - `"__init__"`
        - Known as a constructor
        - Runs when the object is created
        - Used for setting up the class level dependencies and attributes
    - `"__str__"`
        - Returns a string representation of the object when the object is passed into a `str()` or `print()` function
- `self`
    - Denotes that a class wide method or attribute is being accessed
    - a method can not call any attributes or methods without the `self` keywords
    - attributes can not be defined in a method without the keyword
    - attributes can not be modified within a method withou the keyword
- `Access modifiers private vs public`
    - `public`
        - All methods and attribute are public by default
        - Public methods and attributes can be accessed from the object
    - `private`
        - Need to be set to private using the double underscore at start of the name `__`
        - Can only be accessed within the class
        - Add abstraction to the code, preventing access to certain methods that the user doesn't need to know about, or attributes they shouldn't be able to change

Creation and Naming

- We use the class keyword to let Python know that we are creating a class
- The name of the class should be written in PascalCase, where every words starts with an uppercase character
- We can leave out the parentheses if we are not inheriting from anything

```
# With Parentheses
class MyClass():
    ...

# Without Parentheses
class MyClass:
    ...
```

Creating attributes

- We can create attributes directly in the class
    - We wouldn't need to use the `self` keyword
- We can create attributes in a method
    - the `__init__()` method is the advised one since it runs on when the object is created, allowing you to know that an attribute 100% exists
    - We need to use teh `self` keyword to let Python know that it is a class level variable
        - If we don't, the variable will only exist within the method and can't be accessed by other methods.

```
# In the class
class MyClass():
    attribute_one = None
    attribute_two = None

# In the constructor
class MyClass():

    def __init__(self) -> None:
        self.attribute_one = None
        self.attribute_two = None

# Passing values at init
class MyClass():

    def __init__(self, attribute_one, attribute_two) -> None:
        self.attribute_one = attribute_one
        self.attribute_two = attribute_two


# Passing values at init wih default values
class MyClass():

    def __init__(self, attribute_one, attribute_two = None) -> None:
        self.attribute_one = attribute_one
        self.attribute_two = attribute_two
```

Creating Methods

- We can create methods the same way we create a function
- We need to pass the `self` keyword as a parameter when creating our methods so that we can access class level features
- RULES
    - Methods should be able to work with any type of application, not just console applications
        - Methods should not have a print
        - Methods should not have an input
    - Instead
        - Input is handled by the operation calling the method and passes the values as paramters
        - Output is received by the calling operation, the method will return the output value.
- We can make use of the `__str__` dunder to print values when we call the object in a print
- We can make use of the `__add__` dunder method to add do custom addition operations between objects

```
# Create the class
class MyClass():
    hour = None
    minute = None

    def get_class_name(self):
        return "MyClass"

    def set_time(self, hour, minute):
        self.hour = hour
        self.minute = minute

    def __str__(self) -> str:
        return f"{self.hour}:{self.minute}"

    def __add__(self, other):
        return (self.hour + other.hour, self.minute + other.minute)
```

```
# Create the first class
my_class_one = MyClass()
my_class_one.set_time(7, 30)

# Create the second class
my_class_two = MyClass()
```

Public and Private

- Private attributes and methods are declared using `__`
- We use these to hide features that we don't want the user to have access too

```
# Add the two objects together
```

```
import sqlite3



class MyClass():
    __connection_string = None # PRIVATE

    def __init__(self, connection_string, attribute_one) -> None:
        self.__connection_string = connection_string
        self.attribute_one = attribute_one

    def write(self, data):
        conn = self.__create_database_connection()
        cursor = conn.cursor()
        cursor.execute("INSERT INTO data VALUES (?)", (data,))

        conn.close()

    # PRIVATE
    def __create_database_connection(self):
        return sqlite3.connect(self.__connection_string)
```