Welcome to SYEP 2023 Workshop

Community Agreements

- **One person!**

- **Be present and participate.**
  - Use the chat if you can’t unmute or feel shy to participate verbally.
- **Respect the chat.**
- **Include others in the conversation.** Support and encourage others to speak.
- **Promote a comfortable atmosphere.**
- **Maintain appropriate boundaries.**
- **Respect others’ opinions.** Even when you disagree, do so respectfully.
- **Respect others’ differences.** This includes race, gender, religion, culture, sexuality, etc.

Agenda
Week 4

1

Recap of last week's lesson

3

Object Oriented Programming

5

Variables and logical operators

2

Python Libraries

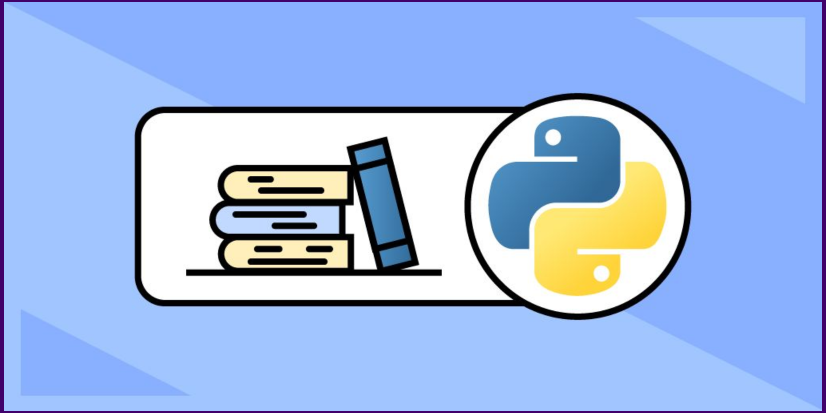
4

Group projects

Python Libraries 2.

Python Libraries:

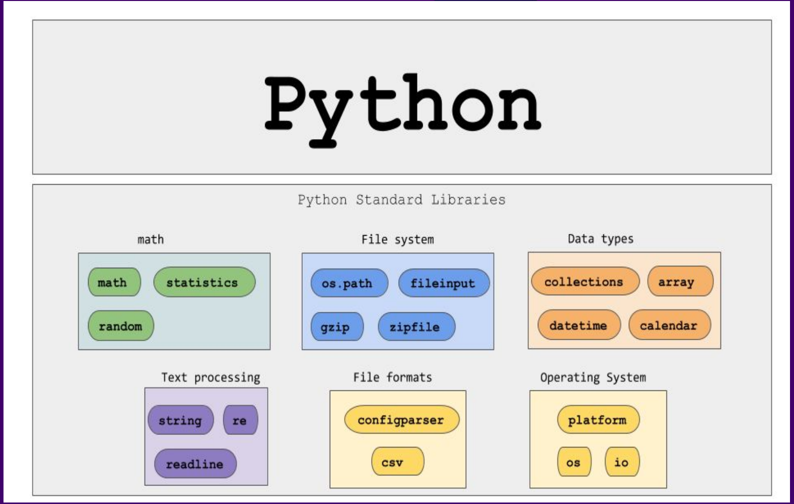
In Python, libraries are collections of pre-written code and functions that extend the language's capabilities. They provide a set of tools and functionalities that developers can use to perform specific tasks without having to write everything from scratch. Python's extensive library ecosystem is one of its greatest strengths, making it a popular language for various domains, from web development to data analysis and machine learning.



Two types of libraries:

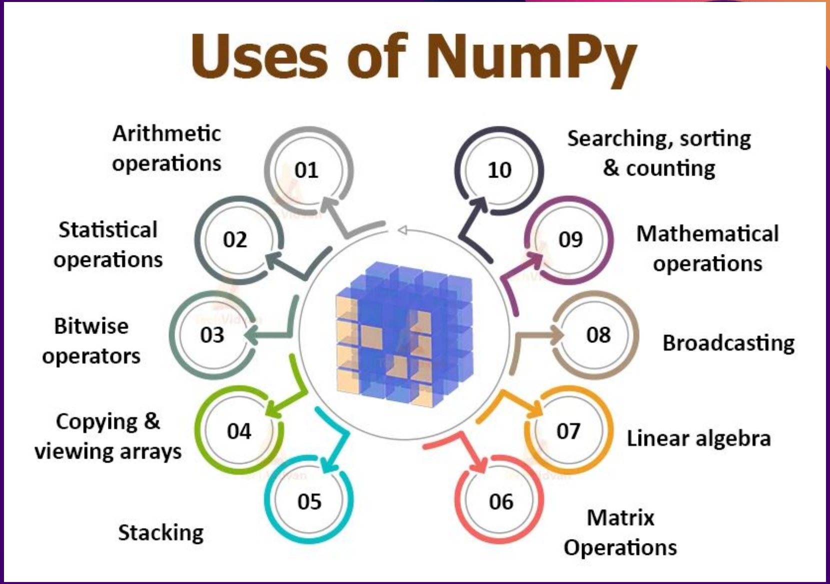
Standard Library: The Python Standard Library comes included with the Python interpreter and contains a vast collection of modules that provide functionalities for tasks like file I/O, working with data structures, handling network communications, interacting with the operating system, and more. These libraries do not require any additional installation because they are part of the Python distribution. Some examples of modules in the standard library include:

- ❖ os: For interacting with the operating system (operations, environment variables).
- ❖ math: For mathematical operations and functions.
- ❖ datetime: For working with dates and times.
- ❖ json: For working with JSON data.
- ❖ urllib: For working with URLs and making HTTP requests.



External Libraries (Third-Party Libraries): Python has a vast ecosystem of third-party libraries that are not part of the standard library but can be installed separately using package managers like pip. These external libraries cover a wide range of functionalities and domains, allowing developers to work on specialized tasks efficiently. Some of the most popular third-party libraries in Python include:

- NumPy: A library for numerical and mathematical operations, particularly for working with arrays and matrices.
- Pandas: A library for data manipulation and analysis, especially for working with structured data.
- Matplotlib and Seaborn: Libraries for creating data visualizations and plots.
- Requests: A library for making HTTP requests to interact with web APIs.
- Django and Flask: Web frameworks for building web applications.
- TensorFlow and PyTorch: Libraries for deep learning and machine learning.



In- depth of Libraries:

Examples of libraries:

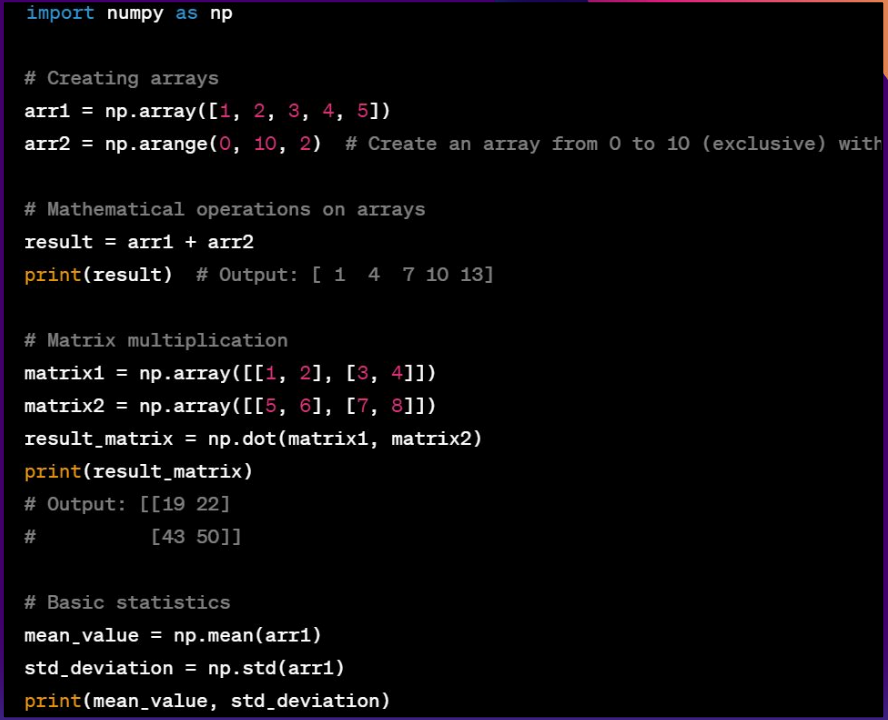
Numpy, Pandas example:
https://colab.research.google.com/drive/1Sh1PSXkt90O66vpIk-WSIk-hoST3XQAz#scrollTo=jBucuzTqnm1B

Data Visualization example:
https://colab.research.google.com/drive/1Sh1PSXkt90O66vpIk-WSIk-hoST3XQAz

Dataframes:
https://colab.research.google.com/drive/16WUIKZ3uUXk6F_NKpmbuFA_xutJvOdeT#scrollTo=_Bz0hl43_3aQ

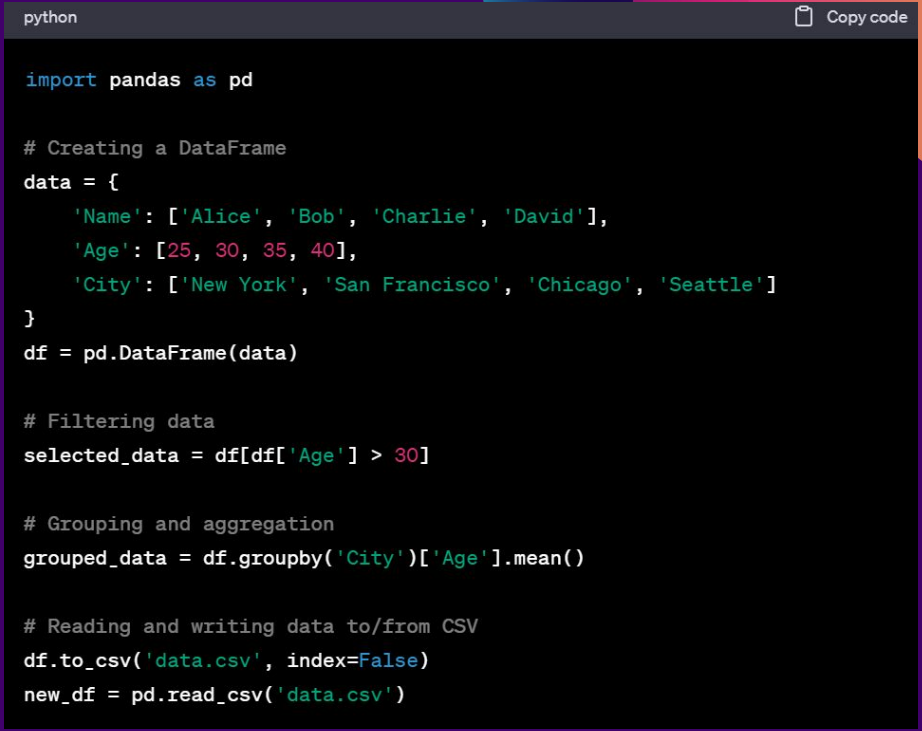
In- class activity:

Using NumPy:
NumPy is a powerful library for numerical computing, particularly for working with arrays and matrices.



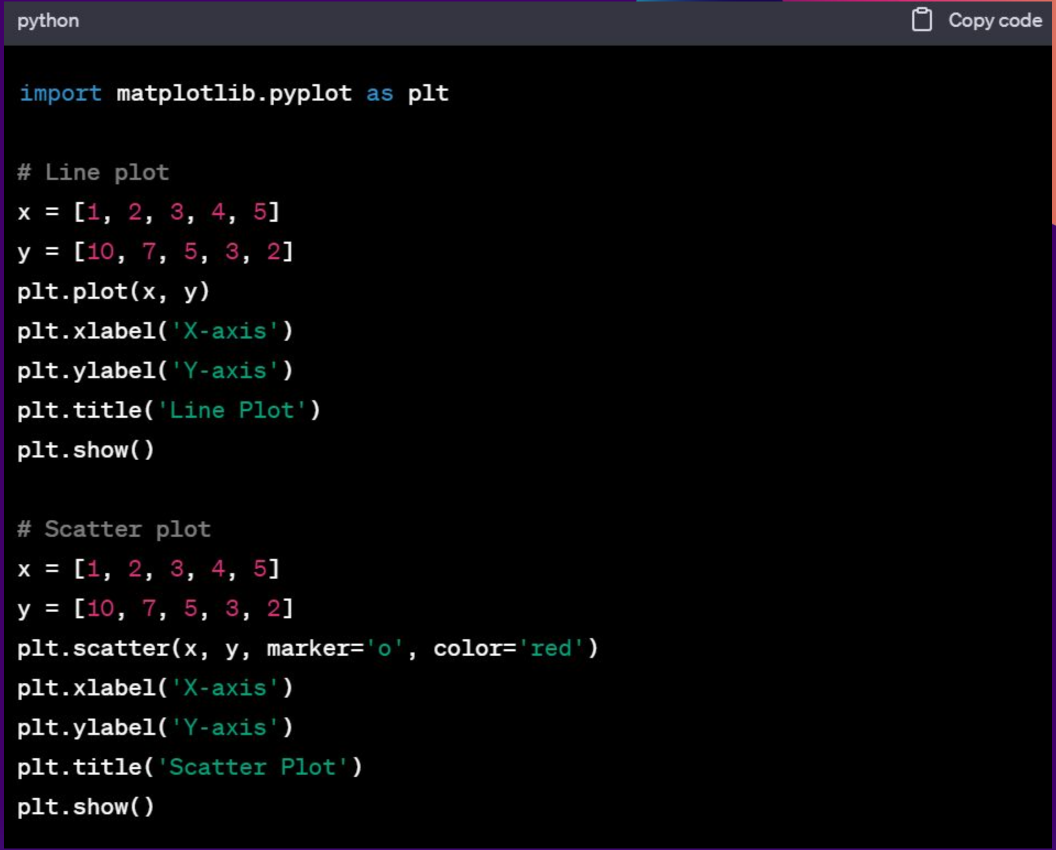
```
import numpy as np

# Creating arrays
arr1 = np.array([1, 2, 3, 4, 5])
arr2 = np.arange(0, 10, 2)  # Create an array from 0 to 10 (exclusive) with

# Mathematical operations on arrays
result = arr1 + arr2
print(result)  # Output: [ 1  4  7 10 13]

# Matrix multiplication
matrix1 = np.array([[1, 2], [3, 4]])
matrix2 = np.array([[5, 6], [7, 8]])
result_matrix = np.dot(matrix1, matrix2)
print(result_matrix)
# Output: [[19 22]
#          [43 50]]

# Basic statistics
mean_value = np.mean(arr1)
std_deviation = np.std(arr1)
print(mean_value, std_deviation)
```

Using Pandas:
Pandas is a library for data manipulation and analysis, particularly with structured data.



```python
import pandas as pd

# Creating a DataFrame
data = {
    'Name': ['Alice', 'Bob', 'Charlie', 'David'],
    'Age': [25, 30, 35, 40],
    'City': ['New York', 'San Francisco', 'Chicago', 'Seattle']
}
df = pd.DataFrame(data)

# Filtering data
selected_data = df[df['Age'] > 30]

# Grouping and aggregation
grouped_data = df.groupby('City')['Age'].mean()

# Reading and writing data to/from CSV
df.to_csv('data.csv', index=False)
new_df = pd.read_csv('data.csv')
```

Using Matplotlib:
Matplotlib is a library for creating data visualizations and plots.



```python
import matplotlib.pyplot as plt

# Line plot
x = [1, 2, 3, 4, 5]
y = [10, 7, 5, 3, 2]
plt.plot(x, y)
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Line Plot')
plt.show()

# Scatter plot
x = [1, 2, 3, 4, 5]
y = [10, 7, 5, 3, 2]
plt.scatter(x, y, marker='o', color='red')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Scatter Plot')
plt.show()
```

3.

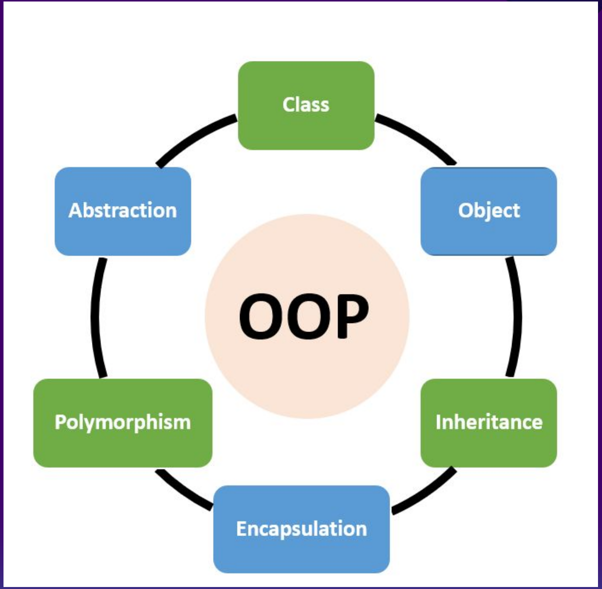
OOP
Object Oriented Programming

Object Oriented Programming

In Python, object-oriented Programming (OOPs) is a programming paradigm that uses objects and classes in programming. It aims to implement real-world entities like inheritance, polymorphisms, encapsulation, etc. in the programming. The main concept of OOPs is to bind the data and the functions that work on that together as a single unit so that no other part of the code can access this data.

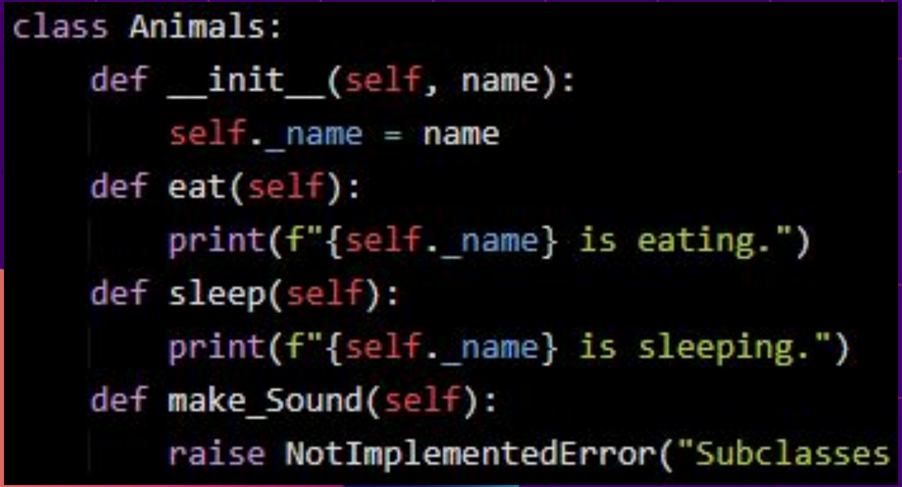
OOPs Concepts in Python

- Class
- Objects
- Inheritance
- Polymorphism
- Encapsulation
- Data Abstraction



Python Class

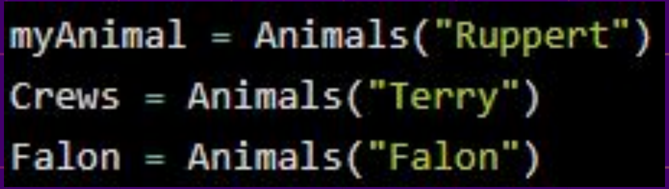
A class is a collection of objects. A class contains the blueprints or the prototype from which the objects are being created. It is a logical entity that contains some attributes and methods.



```
class Animals:
    def __init__(self, name):
        self._name = name
    def eat(self):
        print(f"{self._name} is eating.")
    def sleep(self):
        print(f"{self._name} is sleeping.")
    def make_Sound(self):
        raise NotImplementedError("Subclasses
```

Python Objects

The object is an entity that has a state and behavior associated with its class. You create objects using constructors.



```
myAnimal = Animals("Ruppert")
Crews = Animals("Terry")
Falon = Animals("Falon")
```

Creating a Class

The Python self

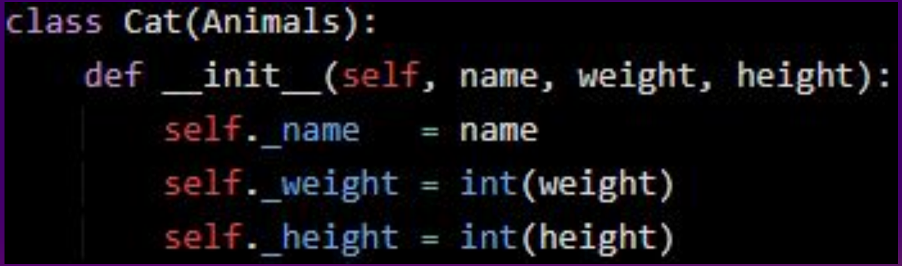
- Class methods must have an extra first parameter in the method definition.

- If we have a method that takes no arguments, then we still have to have one argument.

The Python __init__ Method

- The __innit__ method runs as soon as an object of a class is instantiated.

- The method is useful to do any initialization you want to do with your object.



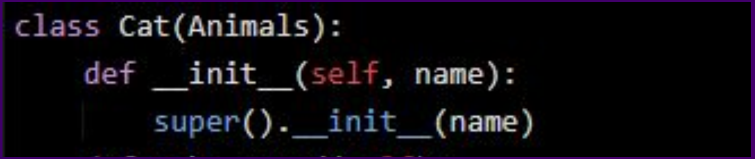
```python
class Cat(Animals):
    def __init__(self, name, weight, height):
        self._name   = name
        self._weight = int(weight)
        self._height = int(height)
```

Attributes, Methods, and Inheritance

Inheritance

Inheritance is the capability of one class to derive or inherit the properties from another class.

The class that derives properties is called the derived class or **child** class and the class from which the properties are being derived is called the base class or **parent** class.

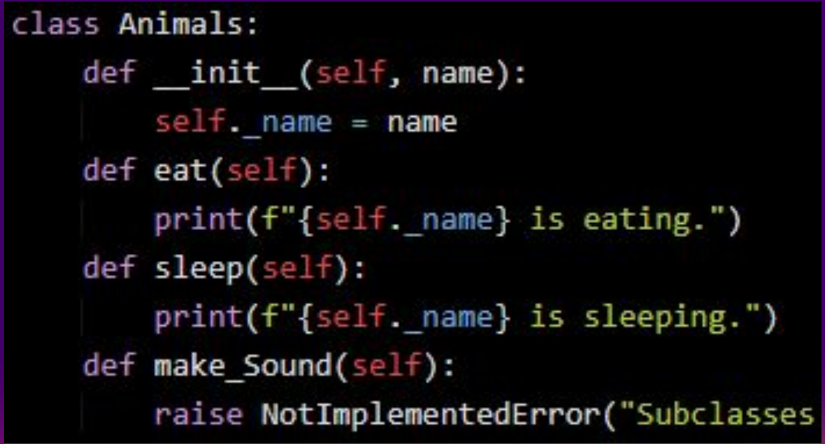


```
class Cat(Animals):
    def __init__(self, name):
        super().__init__(name)
```

Class Attributes and Methods

Class attributes are properties you assign to a class, which then forms an object.

Class methods are much like class attributes, where the method simply belongs to the class and nowhere else.



```
class Animals:
    def __init__(self, name):
        self._name = name
    def eat(self):
        print(f"{self._name} is eating.")
    def sleep(self):
        print(f"{self._name} is sleeping.")
    def make_Sound(self):
        raise NotImplementedError("Subclasses
```

Polymorphism and Encapsulation

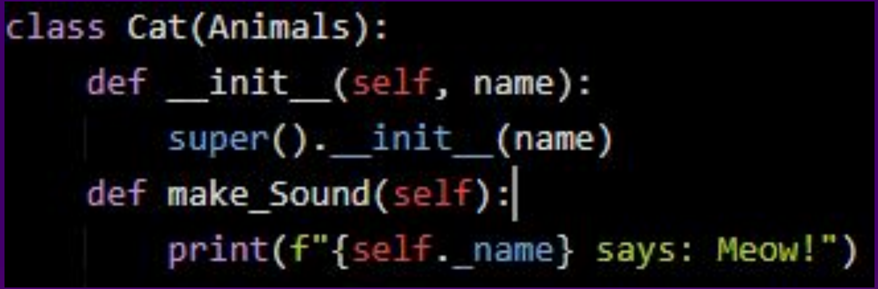
Encapsulation

Encapsulation is one of the fundamental concepts in object-oriented programming (OOP). It describes the idea of wrapping data and the methods that work on data within one unit.

This puts restrictions on accessing variables and methods directly and can prevent the accidental modification of data.

Polymorphism

- Polymorphism simply means having many forms.
- Subclasses can override methods defined in their parent class to provide specific behavior while still inheriting other methods from the parent class.

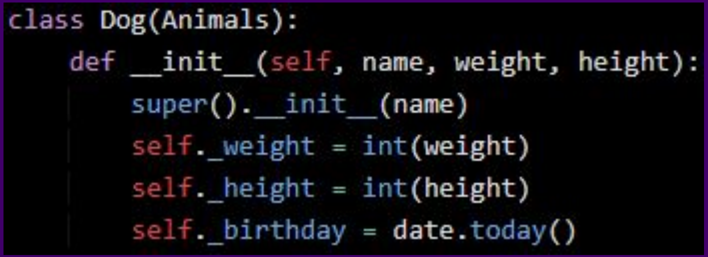


```python
class Cat(Animals):
    def __init__(self, name):
        super().__init__(name)
    def make_Sound(self):
        print(f"{self._name} says: Meow!")
```

Encapsulation, Getter and Setter methods

Private Variables

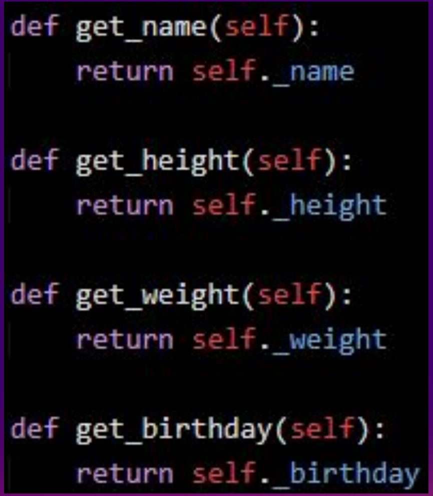
To prevent accidental change, an object's variable can only be changed by an object's method. Those types of variables are known as private variables.



```
class Dog(Animals):
    def __init__(self, name, weight, height):
        super().__init__(name)
        self._weight = int(weight)
        self._height = int(height)
        self._birthday = date.today()
```

Getters

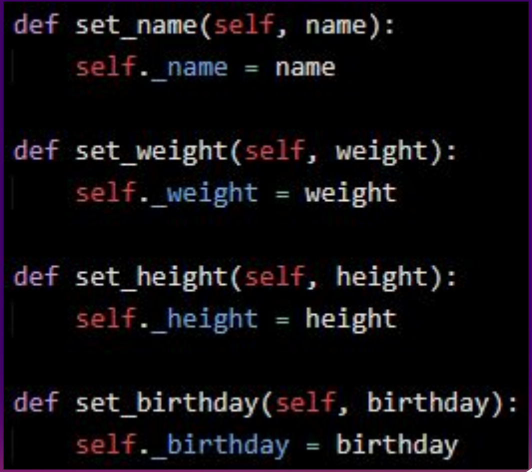
Getter methods simply retrieves the attribute you want



```
def get_name(self):
    return self._name

def get_height(self):
    return self._height

def get_weight(self):
    return self._weight

def get_birthday(self):
    return self._birthday
```

Setters

Setter methods simply sets or modifies the attribute



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

def set_weight(self, weight):
    self._weight = weight

def set_height(self, height):
    self._height = height

def set_birthday(self, birthday):
    self._birthday = birthday
```

Group Projects 4.

Work on your group projects!

Recap: Any questions? Clarifications?