In Partial Fulfillment of the Requirements for the

CS 223 - Object-Oriented Programming

"Four Principles of Object-Oriented Programming"

Presented to:

Dr. Unife O. Cagas
Professor V

Presented by:

Richie R. Burdeos
BSCS 2A2 Student

"Animal Sounds and Habitats"

Project Title

Project Description

In the world of programming, abstraction allows us to encapsulate shared characteristics while enabling unique behavior among different classes. This code snippet demonstrates a simple system to model various animals and their distinct attributes, focusing on their sounds and natural habitats. The code defines an abstract base class Animal with common properties for all animals, including name and species. This class also contains methods to define sound, habitat, and a general info method to represent the animal's basic details. From Animal, three subclasses are derived: Mammal, Bird, and Reptile. These subclasses extend the Animal class and implement their unique sounds and habitats, reflecting the common characteristics of these broader categories. Further, specific animal classes are created to represent individual species: Elephant for mammals, Parrot for birds, and Snake for reptiles. Each of these classes overrides the sound and habitat methods to reflect their unique traits. For instance, elephants are known for trumpeting and inhabit savannahs or forests, while parrots mimic sounds and are found in tropical rainforests. To bring it all together, the show_animal_info function takes a list of animal instances and prints their information, including name, species, sound, and habitat. The code snippet then creates instances of these specific animal classes and demonstrates the use of the show_animal_info function to display their characteristics. This simple yet robust structure allows you to easily add more animal species by extending the appropriate subclass and defining the unique behaviors of each. It serves as a great starting point for modeling various animal behaviors and their natural habitats in a structured way.

Objectives:

1. Establish a foundational abstract class Animal that encapsulates common properties and methods shared by all animal types, allowing for consistent inheritance and polymorphism across different subclasses.

2. Implement derived classes for common categories of animals, such as Mammal, Bird, and Reptile, which can serve as a base for specific species and define generic behaviors for each category.

3. Create subclasses for specific animal species, such as Elephant, Parrot, and Snake, each providing unique implementations of sound and habitat, demonstrating polymorphic behavior.

4. Illustrate the flexibility of polymorphism by overriding abstract methods like sound and habitat in specific animal classes to reflect their distinct characteristics.

5. Develop a utility function (show_animal_info) to display the name, species, sound, and habitat of a list of animal objects, facilitating easy interaction and presentation of animal data.

6. Ensure that the code structure is designed to be extensible, allowing new animal subclasses and specific species to be added without altering existing code. This modular approach supports future growth and code maintenance.

7. The code serves as a foundational platform for exploring animal characteristics, promoting educational purposes, and allowing further exploration of object-oriented programming concepts such as inheritance and abstraction.

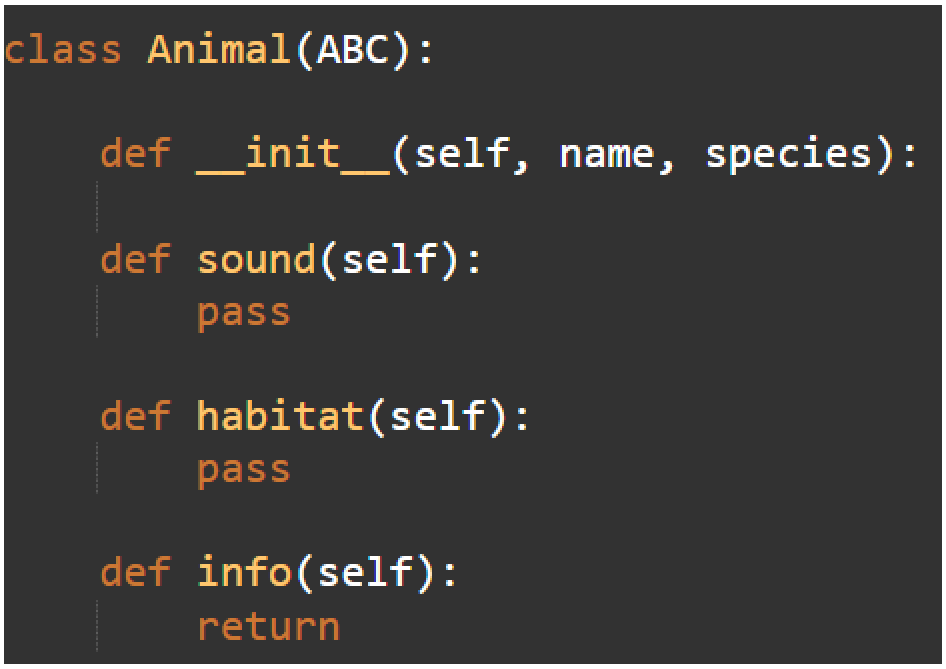
Importance and Contribution of the Project

This project demonstrates the use of object-oriented programming principles in Python, including inheritance, polymorphism, and abstraction. It also showcases the importance of using descriptive naming conventions, type hints, and docstrings to make the code more readable and maintainable. The project's contribution lies in its ability to model real-world entities (animals) using Python classes, and to demonstrate how to use inheritance to create a hierarchy of classes that share common attributes and behaviors. This can be useful in a variety of applications, such as simulations, games, or educational software.

Four Principles of Object-Oriented Programming with code

Class:

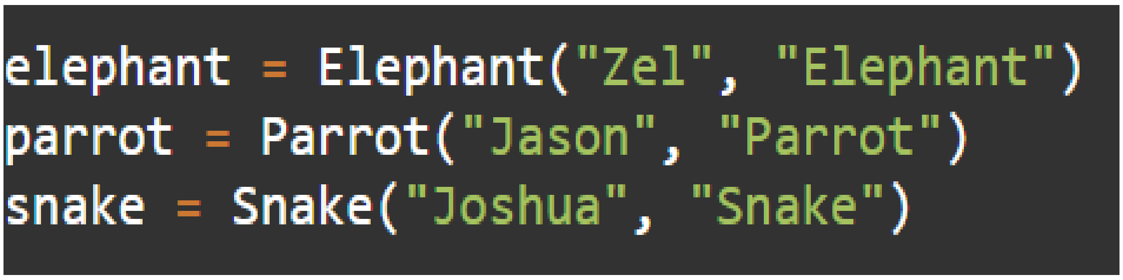
A class is a blueprint or template for creating objects. It defines properties and methods that the objects created from the class will have. In this code, Animal, Mammal, Bird, Reptile, Elephant, Parrot, and Snake are classes. Each class represents a specific concept or category of animals with shared attributes.



```
class Animal(ABC):

    def __init__(self, name, species):

    def sound(self):
        pass

    def habitat(self):
        pass

    def info(self):
        return
```

Object:

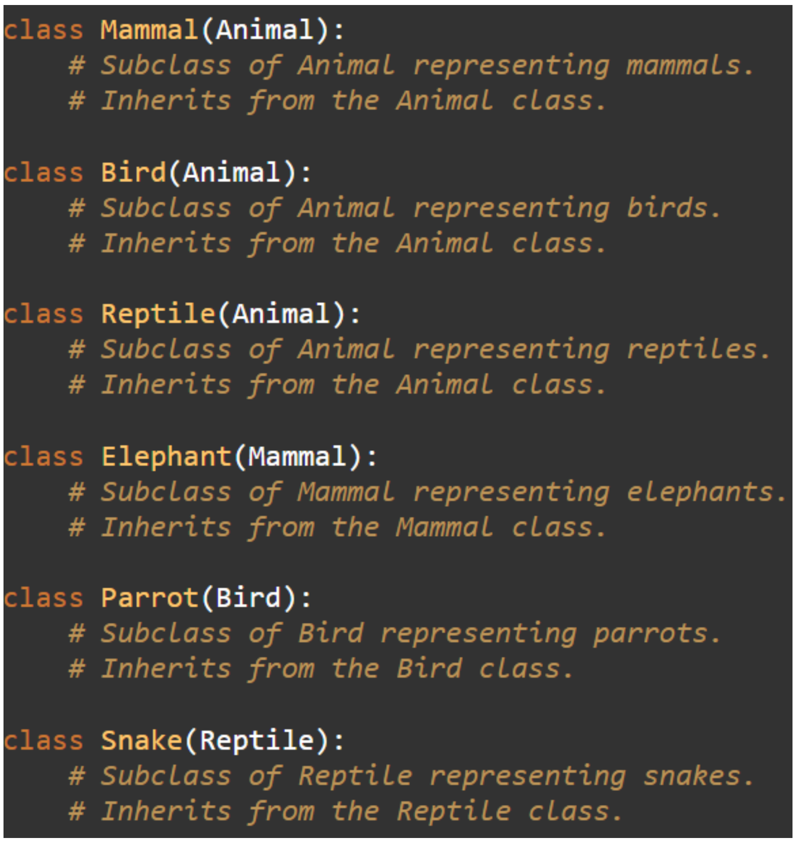
An object is an instance of a class. It embodies the properties and behavior defined by its class. elephant, parrot, and snake are objects created from the Elephant, Parrot, and Snake classes, respectively. These objects represent specific animals with unique characteristics.



```
elephant = Elephant("Zel", "Elephant")
parrot = Parrot("Jason", "Parrot")
snake = Snake("Joshua", "Snake")
```

Inheritance:

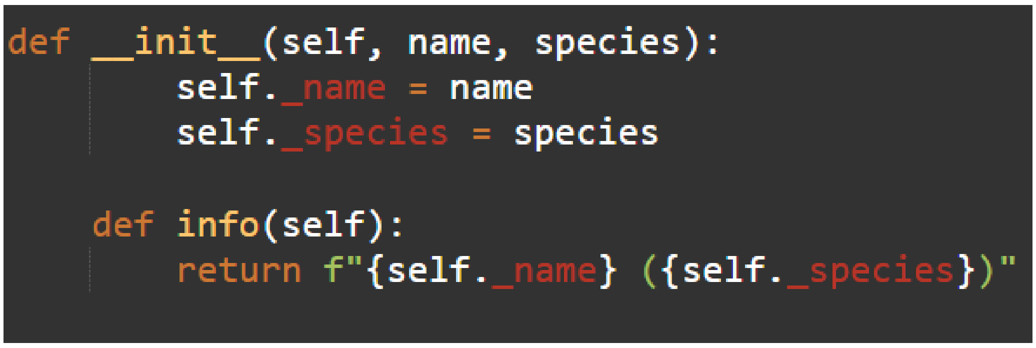
Inheritance allows a class to inherit properties and methods from another class, promoting code reuse and establishing relationships between classes. Mammal, Bird, and Reptile inherit from the Animal base class. This relationship signifies that these subclasses share common features but can also have unique behavior. Elephant, Parrot, and Snake further inherit from these subclasses.



```
class Mammal(Animal):
    # Subclass of Animal representing mammals.
    # Inherits from the Animal class.

class Bird(Animal):
    # Subclass of Animal representing birds.
    # Inherits from the Animal class.

class Reptile(Animal):
    # Subclass of Animal representing reptiles.
    # Inherits from the Animal class.

class Elephant(Mammal):
    # Subclass of Mammal representing elephants.
    # Inherits from the Mammal class.

class Parrot(Bird):
    # Subclass of Bird representing parrots.
    # Inherits from the Bird class.

class Snake(Reptile):
    # Subclass of Reptile representing snakes.
    # Inherits from the Reptile class.
```

Encapsulation:

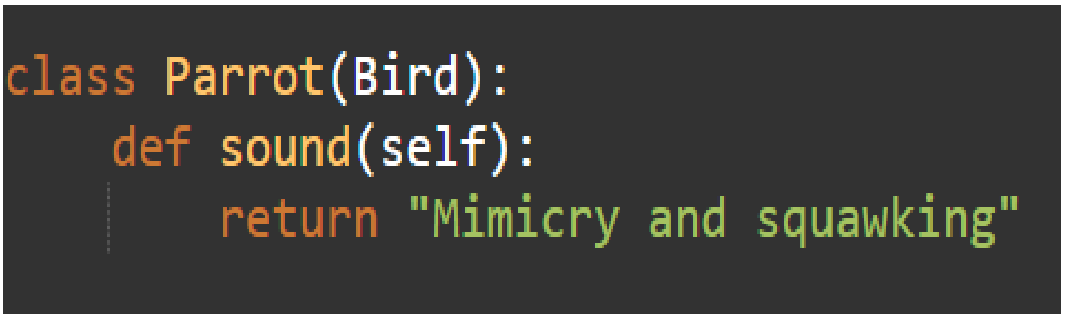
Encapsulation restricts direct access to certain components of an object, enhancing security and modularity. It provides controlled access to internal data. The Animal class encapsulates its attributes _name and _species, which are not directly accessible outside the class. The info method provides a controlled way to access these attributes.



```
def __init__(self, name, species):
        self._name = name
        self._species = species

    def info(self):
        return f"{self._name} ({self._species})"
```

Polymorphism:

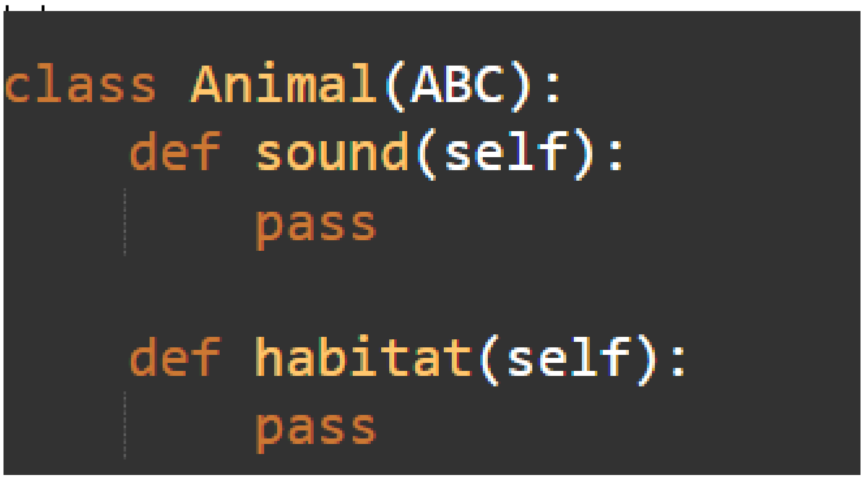
Polymorphism allows different classes to implement the same method in different ways, enabling flexible and dynamic behavior. The sound and habitat methods demonstrate polymorphism. Although they are defined in the Animal class, each subclass provides its unique implementation. For example, Elephant returns "Trumpeting" for sound, while Parrot returns "Mimicry and squawking".



```
class Parrot(Bird):
    def sound(self):
        return "Mimicry and squawking"
```

Abstraction

Abstraction provides a simplified view of a complex system by focusing on essential features while hiding unnecessary details. The Animal class is abstract, providing a generic representation of an animal with abstract methods for sound and habitat. Subclasses must implement these methods, allowing flexibility in how each subclass



```
class Animal(ABC):
    def sound(self):
        pass

    def habitat(self):
        pass
```

Hardware and Software Used

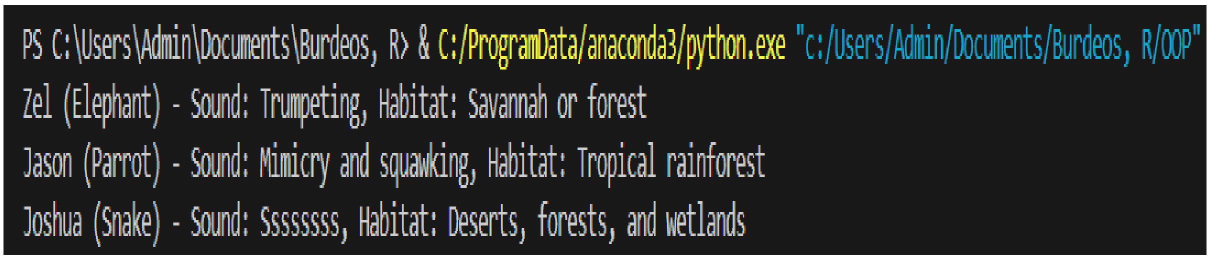
Hardware:

- Laptop
- Cellphone

Software:

- Visual Studio Code
- Online GDB

Output:



```
PS C:\Users\Admin\Documents\Burdeos, R> & C:/ProgramData/anaconda3/python.exe "c:/Users/Admin/Documents/Burdeos, R/OOP"
Zel (Elephant) - Sound: Trumpeting, Habitat: Savannah or forest
Jason (Parrot) - Sound: Mimicry and squawking, Habitat: Tropical rainforest
Joshua (Snake) - Sound: Sssssss, Habitat: Deserts, forests, and wetlands
```

Description:

The given code defines a hierarchy of classes for animals, mammals, birds, reptiles, elephants, parrots, and snakes. Each class has methods for sound and habitat, which are overridden in the subclasses to provide specific information for each animal type. The show_animal_info function takes a list of animals and prints their name, species, sound, and habitat. In the example, an elephant, a parrot, and a snake are created and added to a list. The show_animal_info function is then called with this list, resulting in the output shown above. Each animal's name, species, sound, and habitat are printed on separate lines.

Code Documentation:

from abc import ABC

# Import the Abstract Base Class (ABC) from the "abc" module to create abstract classes.

class Animal(ABC):

  # Create an abstract base class named "Animal".

  def __init__(self, name, species):

    # Constructor to initialize the name and species of the animal.

 self._name = name

    # Set a private instance variable "_name" to hold the animal's name.

```
self._species = species

        # Set a private instance variable "_species" to hold the animal's species.


  def sound(self):

        # Abstract method for the sound the animal makes. This method is intended to
be overridden in subclasses.

        pass


    def habitat(self):

        # Abstract method for the habitat of the animal. This method is intended to be
overridden in subclasses.

        pass


    def info(self):

        # Method to return a formatted string with the animal's name and species.

        return f"{self._name} ({self._species})"



class Mammal(Animal):

    # Create a subclass of "Animal" called "Mammal".


    def sound(self):

        # Override the "sound" method to return a generic mammal sound.

        return "Generic mammal sound"


    def habitat(self):

        # Override the "habitat" method to return a generic description for mammal
habitats.


  return "Varies for mammals"
```

```
class Bird(Animal):
    # Create a subclass of "Animal" called "Bird".

def sound(self):
        # Override the "sound" method to return bird-specific sounds.
        return "Chirping and singing"

    def habitat(self):
        # Override the "habitat" method to return typical habitats for birds.
        return "Nests and trees"

class Reptile(Animal):
    # Create a subclass of "Animal" called "Reptile".

    def sound(self):
        # Override the "sound" method to return reptile-specific sounds.
        return "Hiss or growl"



  def habitat(self):
        # Override the "habitat" method to return typical habitats for reptiles.
        return "Desert or tropical regions"

class Elephant(Mammal):
    # Create a subclass of "Mammal" called "Elephant".

    def sound(self):
 # Override the "sound" method to return the sound elephants make.
        return "Trumpeting"
    def habitat(self):
        # Override the "habitat" method to return the common habitats for elephants.
 return "Savannah or forest"


class Parrot(Bird):
    # Create a subclass of "Bird" called "Parrot".


    def sound(self):
        # Override the "sound" method to return the sound parrots make.
        return "Mimicry and squawking"


    def habitat(self):
        # Override the "habitat" method to return the common habitat for parrots.
        return "Tropical rainforest"


class Snake(Reptile):
    # Create a subclass of "Reptile" called "Snake".


def sound(self):
        # Override the "sound" method to return the sound snakes make.
        return "Sssssss"


    def habitat(self):
        # Override the "habitat" method to return the typical habitat for snakes.
        return "Deserts, forests, and wetlands"


def show_animal_info(animals):
# Function that takes a list of animals and displays their information.
```

```
for animal in animals:
# Loop through each animal in the given list.

 print(f"{animal.info()} - Sound: {animal.sound()}, Habitat: {animal.habitat()}")
 # Print the animal's information, including name, sound, and habitat.

elephant = Elephant("Zel", "Elephant")
# Create an instance of "Elephant" with name "Zel" and species "Elephant".
parrot = Parrot("Jason", "Parrot")
# Create an instance of "Parrot" with name "Jason" and species "Parrot".
snake = Snake("Joshua", "Snake")
# Create an instance of "Snake" with name "Joshua" and species "Snake".
animal_list = [elephant, parrot, snake]
# Create a list containing the created animal instances.

show_animal_info(animal_list)
# Call the "show_animal_info" function with the list of animals to display their information.
```

User Guide:

1. Defining New Animals:

- To define a new type of animal, create a subclass of either Mammal, Bird, or Reptile.
- Implement the sound() and habitat() methods with specific behaviors for the new animal type.

2. Creating Instances:

- To create an instance of a specific animal, instantiate the corresponding subclass with a name and species.

3. Displaying Animal Information:

- Use the show_animal_info function to display information about a list of animals. Pass a list of animal instances as an argument to this function.

4. Customizing Behavior:

- You can customize the sound and habitat of each animal by overriding the sound() and habitat() methods in their respective subclasses.

5. Extending Functionality:

- You can extend the functionality of this code by adding more methods to the Animal class or its subclasses, depending on your requirements.

References:

"Python Tutorial" - W3Schools. [Online]. Available: https://www.w3schools.com/python/default.asp

"Python Programming Language" - GeeksforGeeks. [Online]. Available: https://www.geeksforgeeks.org/python-programming-language/#python-oops-concepts

"ChatGPT - Conversational AI Chatbots for Developers" - ChatGPT. [Online]. Available: https://chatgpt.com/c/ae2acb6c-efef-4e31-b0b6-20aeeb454f30