Computer Science

Summer Springboard

SUMMER SPRINGBOARD
Look Inward. Go Upward

Day 8: OOP & Real World

OOP Review Discussion

- Before hopping into today's content, let's make sure we remember what was covered yesterday
- In groups or as a class, discuss the following topics and their uses:
  - Classes
  - Objects
  - Attributes
  - Methods

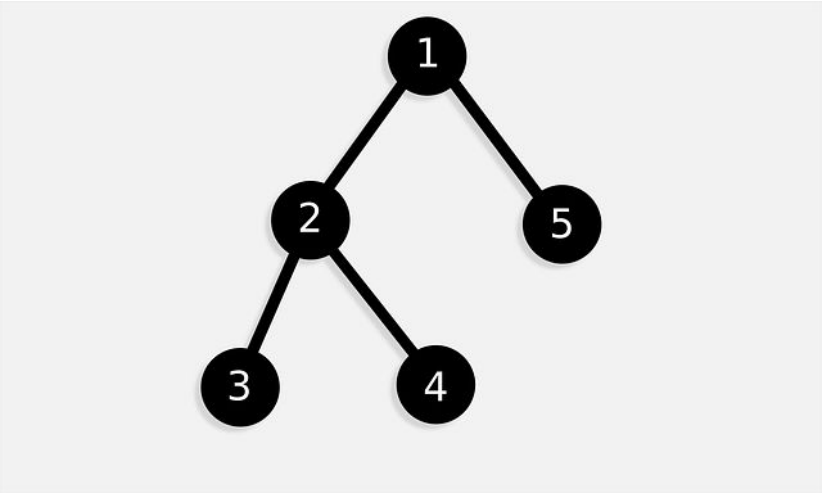
OOP Continued

Inheritance and Polymorphism

- Inheritance and polymorphism are key concepts in OOP that allow for more efficient code reuse and flexibility
- Inheritance enables a new class to inherit attributes and methods from an existing class
- Polymorphism allows methods to do different things based on the object it is acting upon

Inheritance

- Inheritance allows a **child** / **subclass** to inherit attributes and methods from a **parent** / **superclass**
  - Think of this as different hierarchies of classes



Real-Life Inheritance Examples

- Let’s think of a way we could use inheritance to describe a relationship between objects
- One example is Fruit
  - Apples, Bananas, Oranges, etc are fruits
  - Each of the above fruits then have subtypes; for example, there are Gala Apples, Granny Smith Apples, etc
  - Here, Fruit is the superclass / parent and Apple, Banana, Orange, etc are subclasses / children of Fruit
  - Further, Gala, Granny Smith, etc are subclasses of Apple

Real-Life Inheritance Examples (Cont'd)

- Another example we could think of is Pet
  - Cats, Dogs, Snakes could be Pets
  - Each of the above pets then have subtypes; for example, for Dogs there are Golden Retrievers, Dalmations, Pugs, etc
  - Here, Pet is the superclass / parent and Cat, Dog, Snake, etc are subclasses / children of Fruit
  - Further, Golden Retriever, Dalmatian, Pug are subclasses of Dog

Real-Life Inheritance Examples (Cont'd)

- Now, come up with your own ideas!
- In groups or pairs, think of some real-life examples of things you could represent in a class structure with many superclasses / subclasses.

Inheritance Example

- Here is a code example of inheritance using an **Animal** class
- Note that the first parameter for **Dog** and **Cat** is **Animal**, meaning that both **Dog** and **Cat** inherit properties from **Animal**

```
295  class Animal:
296      def __init__(self, name):
297          self.name = name
298
299      def speak(self):
300          pass
301
302  class Dog(Animal):
303      def speak(self):
304          return "Woof!"
305
306  class Cat(Animal):
307      def speak(self):
308          return "Meow!"
```

Inheritance Example (Cont'd)

- What properties are inherited exactly?
- Well, in our **Animal** example, **Dog** and **Cat** inherit the **speak()** method from **Animal**
  - Animals do not all sound the same, so this method is not defined in the **Animal** class itself; rather, the subclasses define it as needed

Adding an Attribute

- Below is an example of adding an attribute to a subclass
  - Note that we access the superclass with **super()**

```
379  class Person:
380      def __init__(self, name, age):
381          self.name = name
382          self.age = age
383
384  class Student(Person):
385      def __init__(self, name, age, student_id):
386          super().__init__(name, age)
387          self.student_id = student_id
```

Inheritance Key Points

- Subclasses can override or extend the functionalities of the superclass
- Python supports multiple inheritance, meaning that a child class can inherit from more than one parent class (example on next page!)

Multiple Inheritance Example

```python
class Employee:
    def __init__(self, name, salary):
        self.name = name
        self.salary = salary

    def get_salary(self):
        return f"{self.name}'s salary is {self.salary}"

class Student:
    def __init__(self, name, student_id):
        self.name = name
        self.student_id = student_id

    def get_student_id(self):
        return f"{self.name}'s student ID is {self.student_id}"

# Multiple Inheritance
class TeachingAssistant(Employee, Student):
    def __init__(self, name, salary, student_id, courses):
        Employee.__init__(self, name, salary)
        Student.__init__(self, name, student_id)
        self.courses = courses

    def get_courses(self):
        return f"{self.name} is teaching {', '.join(self.courses)}"

# Creating an instance of TeachingAssistant
ta = TeachingAssistant("Alex", 30000, "TA12345", ["Math", "Physics"])
print(ta.get_salary())
print(ta.get_student_id())
print(ta.get_courses())
```

Inheritance Wrap-up

- As we have seen, inheritance is super cool!
- We can use it to model real-world relationships, while keeping our code cleaner and more reusable
- This coding practice is widely used and is useful in many applications

Inheritance Practice Problems

- Navigate to today's Google Colab notebook and work up to the "***PAUSE***"

SUMMER SPRINGBOARD
Look Inward. Go Upward

Polymorphism

- Polymorphism allows objects of different classes to be treated as objects of a common superclass
- Closely related to inheritance

Polymorphism Examples

- Below is an example of polymorphism
  - We create a list of **Animal** objects called **animals**
  - Then, we call the **speak()** method of each **Animal** object, which behaves differently depending on the object's class

```
343  animals = [Dog("Buddy"), Cat("Whiskers")]
344
345  for animal in animals:
346      print(animal.speak())
```

Polymorphism Examples (Cont'd)

- Here is an example of polymorphism with fantasy character classes

```
class Character:
    def attack(self):
        raise NotImplementedError("Each character must have its own unique attack method")

class Wizard(Character):
    def attack(self):
        return "Casting a powerful spell!"

class Knight(Character):
    def attack(self):
        return "Swinging a mighty sword!"

class Archer(Character):
    def attack(self):
        return "Shooting an arrow!"

def character_attack(char):
    print(f"{char.__class__.__name__} attacks: {char.attack()}")

# Creating instances of different character types
wizard = Wizard()
knight = Knight()
archer = Archer()

# Characters attacking
characters = [wizard, knight, archer]
for char in characters:
    character_attack(char)
```

Polymorphism Wrap-Up

- Polymorphism promotes flexibility and integration between components
- Enables the implementation of functions that can utilize objects of different classes through a common interface

Polymorphism Practice Problems

- Now, for our last practice problem set (sad times :(very sad), navigate to today's Google Colab notebook and work through the rest of the problems

SUMMER SPRINGBOARD
Look Inward. Go Upward

Programming in the Real-World

Examine Real-World Applications

- In this section, our goal is to explore how the programming concepts we've learned apply to real-world applications
- The format will be as follows: a slide will be shown with a real-world application; this will be a slide for guesses and discussion. The next slide will show specific details about what the applications likely use in their code; this can also be discussed

Real-World Example 1

- Our first example will be Ride-Sharing Apps, such as Uber and Lyft
- How do you think the coding concepts we have learned show up in these apps?
- Discuss in groups or as a class

Real-World Example 1 (Cont'd)

- Here are some likely uses of coding concepts in these apps:
  - Variables and Data Types: Store user profiles, location data, and trip details
  - Conditionals and Loops: Determine the closest available driver, calculate fares based on distance and traffic conditions
  - Functions: Modularize tasks like fare calculation, route optimization, and payment processing
  - Libraries: Use mapping and payment processing libraries for geolocation services and transactions
  - Objects: Model users, drivers, rides, and payment information as objects with specific attributes and behaviors

Real-World Example 2

- Our next example will be Minecraft
- How do you think the coding concepts we have learned show up in Minecraft?
- Discuss in groups or as a class

Real-World Example 2 (Cont'd)

- Here are some likely uses of coding concepts in Minecraft:
  - Variables and Data Types: Store information about player health, inventory items, block types, and world coordinates
  - Conditionals: Determine the outcomes of player interactions with the environment (e.g., if a player hits water or lava)
  - Functions: Modularize actions such as breaking blocks, crafting items, and spawning mobs
  - Objects: Players, mobs, and blocks can be modeled as objects with specific attributes (health, type, position) and methods (move, attack, break).

Real-World Example 3

- Our next example will be YouTube
- How do you think the coding concepts we have learned show up in YouTube?
- Discuss in groups or as a class

Real-World Example 3 (Cont'd)

- Here are some likely uses of coding concepts in YouTube:
  - Variables and Data Types: Store details about videos (title, description, length), user accounts (username, preferences), and comments
  - Conditionals and Loops: Determine video recommendations based on user history and preferences, looping through datasets to find matches.
  - Functions and Libraries: Use functions for video processing, encoding, and streaming. External libraries are integrated for data analysis, machine learning for recommendations, and handling video formats.

Discussion - What do you want?

SUMMER
SPRINGBOARD
Look Inward. Go Upward.

Discussion Time!

- Since we are taking the time to talk about the real-world, let's talk about our goals!
  - What would you like to use computer science for in the future?
  - What kinds of projects do you see yourself building with Python?
  - Where do you see yourself taking your first steps in computer science?

Final Project Development Time

SUMMER
SPRINGBOARD
Look Inward. Go Upward.

Final Project

- As with yesterday, let's take some time to work on your final project while in class!
- Use this time to make meaningful progress on your project and ask your professor any important questions you may have
- You could also use this time to try to implement objects into your project