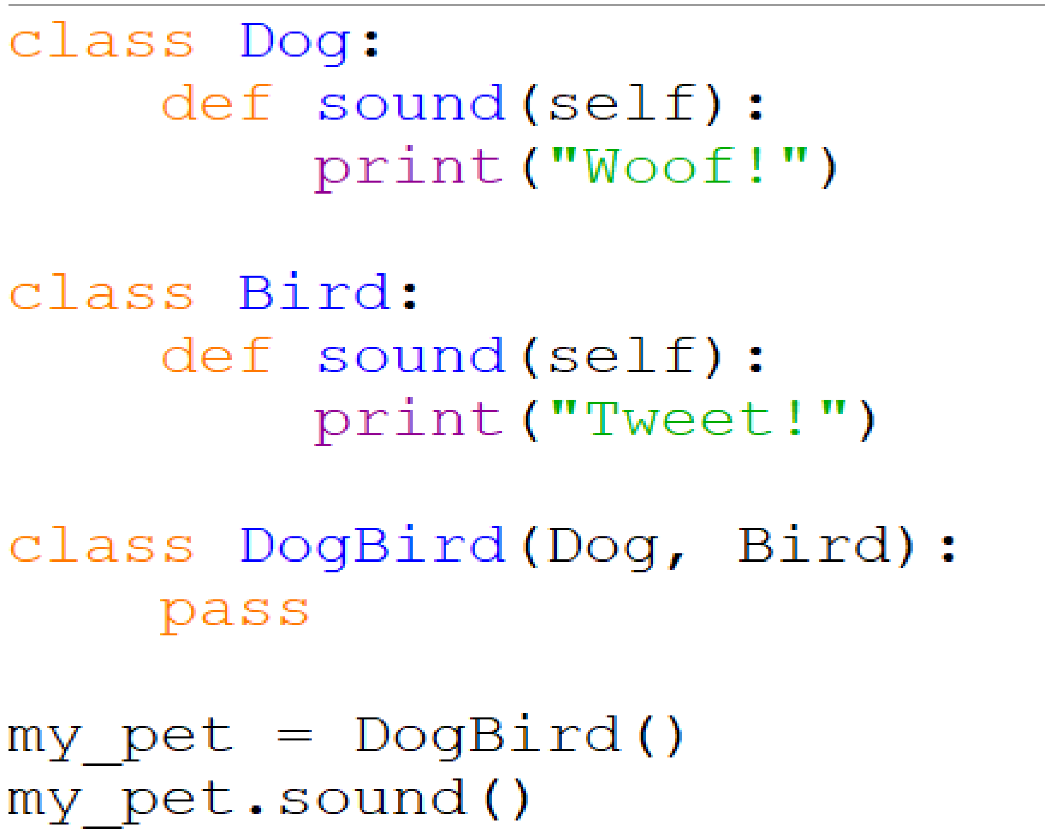
CSE 2040 Programming IV Lecture #33

What will we learn today

- Inheritance
  - Multiple inheritance
  - Diamond Problem
  - Method Resolution Order- MRO
- Polymorphism
  - Duck Typing Philosophy of Python

Multiple inheritance

- Multiple inheritance in Python allows a class to inherit attributes and methods from more than one parent class. This means that a child class can inherit from multiple parent classes.



```python
class Dog:
    def sound(self):
        print("Woof!")

class Bird:
    def sound(self):
        print("Tweet!")

class DogBird(Dog, Bird):
    pass

my_pet = DogBird()
my_pet.sound()
```

Method Resolution Order (MRO)

- Python uses MRO to resolve method calls in the context of multiple inheritance. It's important to note that MRO is calculated dynamically based on the inheritance hierarchy at runtime, and it helps avoid ambiguity in method resolution.
- Classname.mro()

Method Resolution Order (MRO)

```python
'''Demonstrating MRO: hierarchy '''
class A:
    def greet(self):
        print("Greetings from class A")

class B(A):
    def greet(self):
        print("Greetings from class B")
        super().greet()

class C(A):
    def greet(self):
        print("Greetings from class C")
        super().greet()

class D(B, C):
    pass

obj_D = D()
obj_D.greet()
```

Incorrect usage of super() can lead to unexpected behavior, especially in cases of multiple inheritance.

Cont’d

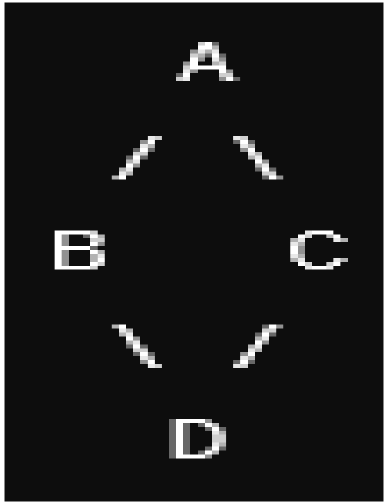
This demonstrates how the super() function is used to delegate method calls to the next class in the method resolution order, allowing for cooperative multiple inheritance and ensuring that all classes in the inheritance hierarchy get a chance to execute their relevant code.

Diamond problem in Python

```python
class A:
    def greet(self):
        print("Greetings from class A")

class B(A):
    def greet(self):
        print("Greetings from class B")

class C(A):
    def greet(self):
        print("Greetings from class C")

class D(B, C):  # Diamond problem
    pass

obj_D = D()
obj_D.greet()
```

Cont’d

When obj_D.greet() is called, it creates a diamond-shaped inheritance structure like this:



```
    A
  /   \
B       C
  \   /
    D
```

In this scenario, Python faces ambiguity when resolving the greet() method for class D. Since both B and C inherit from A, there are two paths to reach class A when calling greet() on D.

Cont’d

- To solve this problem uses order of methods in the inheritance hierarchy
- The order of resolution is calculated based on the order of inheritance specified when defining the subclass.

Python Polymorphism

The word ”polymorphism” means ”many forms”, and in programming it refers to methods/functions/operators with the same name that can be executed on many objects or classes.

1. Function Polymorphism

```
x = ”Hello World! ”
print (len (x) )
```

Output: 12

```
mytuple = ("apple", "banana", "cherry")

print(len(mytuple))
```

Output: 3

```
thisdict = {
  ”brand”: ”Ford”,
  ”model”: ”Mustang”,
  ”year”: 1964
}
print (len (thisdict) )
```

Output: 3

Polymorphism with Duck typing

Polymorphism is often used in Class methods, where we can have multiple classes with the same method name.

For example, say we have three classes: Car, Boat, and Plane, and they all have a method called move():

Lecture33_polymorphism1.py On LMS

Inheritance Class Polymorphism

If we use the example above and make a parent class called Vehicle, and make Car, Boat, Plane child classes of Vehicle, the child classes inherits the Vehicle methods, but can override them:

- Child classes inherits the properties and methods from the parent class.
- In the example above you can see that the Car class is empty, but it inherits brand, model, and move () from Vehicle.
- The Boat and Plane classes also inherit brand, model, and move () from Vehicle, but they both override the move () method.

- Because of polymorphism we can execute the same method for all classes.

inheritance-class-polyphism.py On LMS

Inheritance and type of methods

Inherited Method	Overloaded Method	Overridden Method
• Same name	• Same name	• Same name
• Same parameters	• Different parameters	• Same parameters
• Same implementation	• Different implementation	• Different implementation

Duck Typing

```
x = 12000
  print(type(x))

  x = 'Dynamic Typing'
  print(type(x))

  x = [1, 2, 3, 4]
  print(type(x))
```

```
<class 'int'>
<class 'str'>
<class 'list‘>
```

duck typing is to provide support for dynamic typing in Python programming

Credit: www.javapoint.com

Duck Typing Philosophy of Python

Lecture33polymorphismE.g1

Lecture33polymorphismE.g2

References

- Dr. R. Nageswara Rao, Core Python Programming, Second Edition, 2018