Digital Career Institute

Python Course - OOP Concepts

Digital Career Institute

Inheritance

Inheritance

Inheritance

Inheritance allows classes to **inherit** features of other classes.

Parent classes extend attributes and behaviors to child classes.

If basic attributes and behaviors are defined in a parent class, child classes can be created extending the functionality of the parent class, and adding additional attributes and behaviors.

The benefits of inheritance are that programs can create a generic parent class, and then create more specific child classes as needed.

This simplifies overall programming, because instead of recreating the structure of the generic class multiple times, child classes automatically gain access to functionalities within their parent class.

Inheritance uses a parent-child relationship (IS-A relationship).

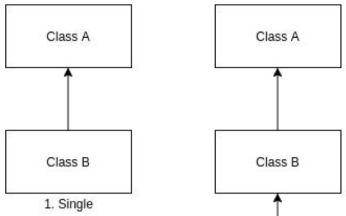
Inheritance - What is Inherited

A **class** can **extend one or multiple classes**. A class can also **extend a class that already extends another class**, thus creating a multilevel inheritance.

What is inherited?

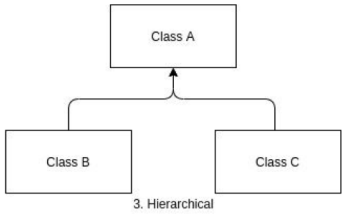
1. Methods.
2. Attributes.

Inheritance - Types

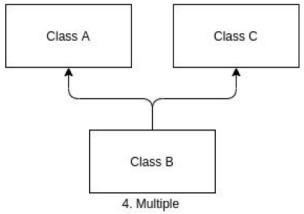


1. Single

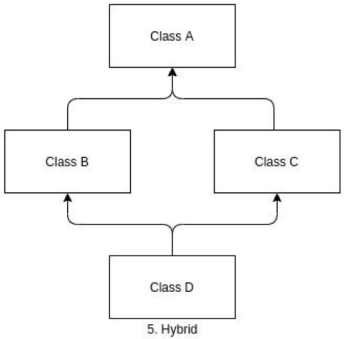
2. Multi level



3. Hierarchical



4. Multiple



5. Hybrid

1. **Single**: When a *class* ***extends*** *from another class* (either concrete or abstract).
2. **Multi level**: When a *class* ***extends*** *from another class that already extends another class*.
3. **Hierarchical**: When two or more different classes **extend** from another class.
4. **Multiple**: *W*hen a class **extends** multiple interfaces.
5. **Hybrid**: *A* mix of *multi level, hierarchical and multiple* inheritance types.

Inheritance - super()

Digital Career Institute

The *super()* built-in function

The **`super()`** built-in function in Python plays a major role in Inheritance because we can use it to **refer to parent class methods and fields**.

In other words, we can use the **`super()`** built-in function to call methods and data members of the immediate parent class.

- Whenever we create an instance of the child class, then **the instance of the parent class is created implicitly**. A reference can be obtained by calling the **super()** built-in function.
- If we have the **same method name in a child as well as parent** class then the **`super()`** built-in function is used to call the parent class method.
- **`super().__init__(...)`** is used to invoke the super-class's **constructors**

Inheritance - super()

```
>>> class Animal:
...   def __init__(self, name):
...     self.name = name
...   def display(self):
...     print(f"Name: {self.name}")
...
>>> class Fish(Animal):
...   def display(self):
...     print("Type: Fish")
...     super().display()
...
```

```
>>> my_fish = Fish('Trout')
>>> my_fish.display()
Type: Fish
Name: Trout
```

Invoking the display() method of the parent class with the super() built-in function

Inheritance - super()

```
>>> class Animal:
...   def __init__(self, name):
...     self.name = name
...     print(f"Name: {self.name}")
...
>>> class Bird(Animal):
...   def __init__(self, name, flying):
...     super().__init__(name)
...     self.can_fly = can_fly
...     print(f"Flying: {self.can_fly}")
...
```

```
>>> my_bird = Bird('Penguin', False)
Name: Penguin
Flying: False
```

Calling the constructor of the parent class with the super() built-in function.

Inheritance - self

self

As the name defines, **`self` refers to the current object** and it is a **reference variable**. It is used to **refer to the current object inside a method or a constructor**.

`self` is used in various contexts as given below:

- To refer to the **instance variables and methods of current class**
- Is automatically passed as the first **argument in all method calls**
- Can be used to **return the current class instance**
- It's mostly used for **ambiguity in variable names inside the same scope**

Note: In Python, the use of *self* is merely a convention. It is not a keyword. You can use other terms instead. The term that is used within a method is the name of the first argument of that method.

Inheritance - self

```
>>> class Person:
...     def __init__(self, name, age):
...         self.name = name
...         self.age = age
...
...     def show(self):
...         print(f"{self.name}
{self.age}")
...
...     def info(self):
...         print(self)
```

```
>>> me = Person('Thomas', 35)
>>> me.show()
Thomas 35
>>> me.info()
<__main__.Person at 0x7f9598cb2f70>
>>> print(me)
<__main__.Person at 0x7f9598cb2f70>
```

self is a reference to the object itself.

Inheritance - super() & self

	`super()`	`self`
Definition	Refers to the immediate parent class instance	Refers to the current class instance
Invoke	Can be used to invoke immediate parent class method	Can be used to invoke current class method
Constructor	`super().__init__()` references the constructor of the immediate parent class	`self.__init__()` references the constructor of the current class
Override	When invoking a superclass version of an overridden method the `super()` `built-in function` should be used	When invoking a current version of an overridden method the `self` keyword can used

Inheritance - multiple and super()

Digital Career Institute

super() and single inheritance

In very simple cases, one can reference the parent class directly and achieve the same result:

```
class Higher:
          def
__init__(self):

          print('Higher')

class Lower(Higher):
          def
__init__(self):

          print('Lower')

          super().__init__()

>>> a = Lower()
```

```
class Higher:
          def __init__(self):

          print('Higher')

class Lower(Higher):
          def __init__(self):

          print('Lower')

          Higher.__init__(sel
f)

>>> a = Lower()
Lower
```

Inheritance - multiple and super()

```
class Top:
        def __init__(self):
                print('Top')

class Middle1(Top):
        def __init__(self):

        print('Middle1')

        Top.__init__(self)

class Middle2(Top):
        def __init__(self):

        print('Middle2')

        Top.__init__(self)

class Bottom(Middle1, Middle2):
        def __init__(self):

        print('Bottom')
```

multiple inheritance without super()

```
>>> a = Bottom()
Bottom
Middle1
Top
Middle2
Top
```

Note: Top is initialized twice

Inheritance - multiple and super()

```
class Top:
        def __init__(self):
            print('Top')

class Middle1(Top):
        def __init__(self):
            print('Middle1')
            super().__init__()

class Middle2(Top):
        def __init__(self):
            print('Middle2')
            super().__init__()

class Bottom(Middle1, Middle2):
        def __init__(self):
            print('Bottom')
            super().__init__()
```

multiple inheritance with super()

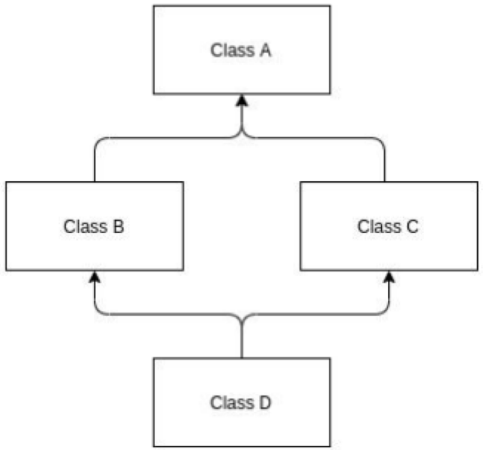
```
>>> a = Bottom()
Bottom
Middle1
Middle2
Top
```

Note: the usage of super() ensures that Top is only initialized once.

Inheritance - multiple and super()

The diamond-shape problem

- If one goes up in the hierarchy of classes, eventually all classes will derive from object.
- Therefore, whenever using multiple inheritance, all the parents will have the same ancestors if one goes up the hierarchy sufficiently high.
- In order to prevent calling the same method on an ancestor several time, one can use the **super()** built-in function.



At the core of the lesson

Digital Career Institute

Inheritance

- Inheritance is one of the key features of OOP that allows us to create a new class from an existing class.
- The new class that is created is known as subclass (child or derived class) and the existing class from where the child class is derived is known as superclass (parent or base class).
- In Python, inheritance is performed by using the parent class as a parameter when creating the child class.
- In Python, inheritance is an is-a relationship. That is, we use inheritance only if there exists an is-a relationship between two classes.
- Python allows multiple inheritance and the super() built-in methods allows us to avoid the diamond-shape problem.

Documentation

Documentation

Digital Career Institute

1. The Python tutorial on classes (docs.python.org)
2. Object-Oriented Programming (OOP) in Python 3 (realpython.com)
3. Java Classes and Objects (w3schools.com)
4. Python Object Oriented Programming (programiz.com)
5. self in Python class (geeksforgeeks.org)
6. Python super() (programiz.com)
7. Python’s super() considered super! (rhettinger.wordpress.com)
8. Function and Variables Names to Constants (python.org)


THANK
YOU