Prof R Madana Mohana

OBJECT ORIENTED PROGRAMMING (USING PYTHON)

INHERITANCE

Introduction | Polymorphism (Overriding)

https://www.youtube.com/c/RASINENIMADANAMOHANA

OUTLINE

Introduction to Inheritance

Inheriting classes

Polymorphism and Method Overriding

Inheritance - Introduction

- Reusability is an important feature of Object Oriented Programming (OOP).
- Reusing an existing piece of code has many benefits.
- It not only saves effort and cost required to build a software product, but also enhances its reliability.

Inheritance - Introduction

- Instead of reinventing the same code that is already available, it makes sense in reusing existing code.
- Python permits two code reuse mechanisms:

1. Containership (also called composition or complex objects)
2. Inheritance

Inheritance - Introduction

- In both mechanisms we can reuse existing classes and create new enhanced classes based on them.
- We can reuse existing classes even if their source code is not available.

Inheritance - Introduction

Which to use and when?

- **Containership** should be used when the two classes have a '**has-a**' relationship.
- **For example**, a College has Professors. So College class's object can contain one or more Professor class's object(s).

Inheritance - Introduction

Which to use and when?

- **Inheritance** should be used when the two classes have a '**like-a** or **is-a**' relationship.
- **For example**, a **Button** is like a **Window**. So **Button class** can inherit features of an existing class called **Window**.

Inheritance - Introduction

Inheritance:

- The technique of creating a **new class** from an **existing class** is called ***inheritance***.
- The old or existing **class** is called the ***base class*** or ***super class*** or ***parent class*** and the new class is known as the ***derived class*** or ***sub-class*** or ***child class***.

Inheritance - Introduction

Inheritance:

- The ***derived classes*** are created by first inheriting the data and methods of the ***base class*** and then adding new specialized data and functions in it.
- In this process of **inheritance**, the ***base class*** remains unchanged.
- The concept of **inheritance** is used to implement the **is-a** relationship.

Inheritance - Introduction

Inheritance:

For example, teacher **IS-A** person, student **IS-A** person; while both teacher and student are a person in the first place, both also have some distinguishing features. So all the common traits of teacher and student are specified in the ***Person class*** and specialized features are incorporate in two separate classes- ***Teacher*** and ***Student***.

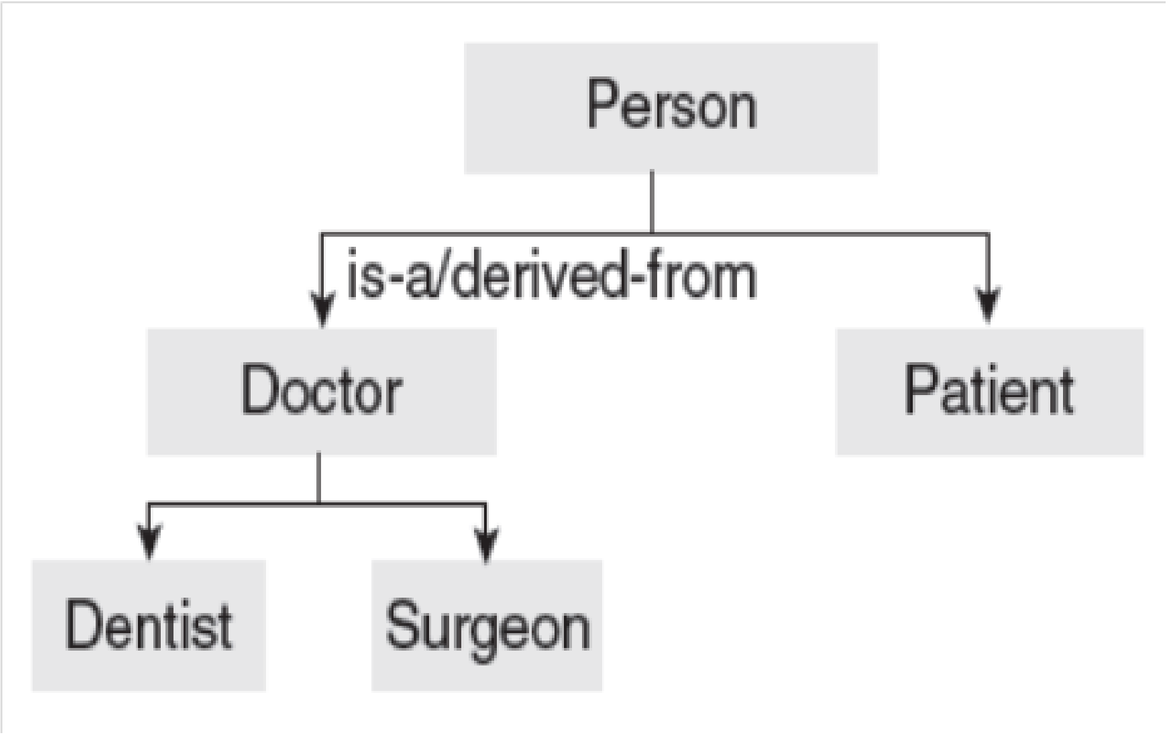
Inheritance - Introduction

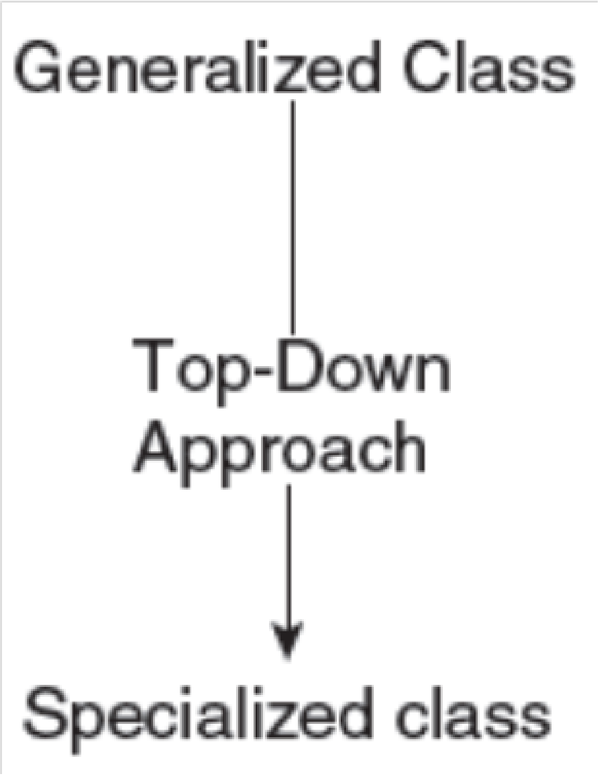
Inheritance:

- ***Inheritance*** which follows a top-down approach to problem solving.
- In top-down approach, generalized classes are designed first and then specialized classes are derived by inheriting/extending the generalized classes.

Inheritance - Introduction

Inheritance:





Inheriting Classes in Python

- In **python**, a derived class can ***inherit*** base class by just mentioning the base in the bracket after the derived class name.
- Syntax to ***inherit*** a base class into the derived class:

```
class derived-classname(base classname):
    list of variables
    list of methods
```

Inheriting Classes in Python

Example: Program to demonstrate the use of inheritance

```
class Person: # Here, Person is Base or Master or
Parent class
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def display(self):
        print("Name:", self.name)
        print("Age:", self.age)
```

Inheriting Classes in Python

Example: Program to demonstrate the use of inheritance

```
class Teacher(Person): # Here, Teacher is derived class-1 and
Person is Base class
    def __init__(self, name, age, experience, research_area):
        Person.__init__(self, name, age)
        self.experience = experience
        self.research_area = research_area
    def displaydata(self):
        Person.display(self)
        print("Experience:", self.experience)
        print("Research Area:", self.research_area)
```

Inheriting Classes in Python

Example: Program to demonstrate the use of inheritance

```
class Student(Person): # Here, Student is derived class-2 and
Person is Base class
    def __init__(self, name, age, course, year_semester):
        Person.__init__(self, name, age)
        self.course = course
        self.year_semester = year_semester
    def displaydata(self):
        Person.display(self)
        print("Course:", self.course)
        print("Year & Semester:", self.year_semester)
```

Inheriting Classes in Python

Example: Program to demonstrate the use of inheritance

```
print("------TEACHER DETAILS------:")
T = Teacher("Dr. R. Madana Mohana", 42, 17, "Artificial
Intelligence & Machine Learning")
T.displaydata()
print("------STUDENT DETAILS----:")
S = Student("RAM", 20, "B.E AI&DS-2", "I Year, II
Semester")
S.displaydata()
```

Inheriting Classes in Python

Example: Program to demonstrate the use of inheritance

OUTPUT:

```
------TEACHER DETAILS------:
Name: Dr. R. Madana Mohana
Age: 42
Experience: 17
Research Area: Artificial Intelligence & Machine Learning
------STUDENT DETAILS----:
Name: RAM
Age: 20
Course: B.E AI&DS-2
Year & Semester: I Year, II Semester
```

Inheriting Classes in Python

- In the above program, classes **Teacher** and **Student** are both **inherited** from class **Person**.
- Therefore, the ***inherited classes*** have all the features (attributes and methods) of the ***base class***.
- A ***derived class*** is instantiated in the same way as any ***other class*** is.

Inheriting Classes in Python

- To create an object of the ***derived class***, just write the ***derived class*** name followed by an empty brackets as in **DerivedClassName()**.
- When we use **.__base__** attribute with class name, the **base** (or the ***parent***) **class** of the specified ***class*** is returned.
- Therefore, **print(Student.__base__)** will print (**<class '__main__.Person'>**)

Polymorphism & Method Overriding

- **Polymorphism** refers to having several different forms.
- It is one of the key features of **Object Oriented Programming (OOP)**.
- It enables the programmers to assign a different meaning or usage to a **variable**, **function**, or an **object** in different contexts.

Polymorphism & Method Overriding

- While **inheritance** is related to **classes** and their **hierarchy**, **polymorphism**, on the other hand, is related to methods.
- When **polymorphism** is applied to a **function** or **method** depending on the given parameters, a particular form of the **function** can be selected for execution.
- In **Python**, **method overriding** is one way of implementing **polymorphism**.

Polymorphism & Method Overriding

- In **Python**, method overriding is the process of defining a method in a subclass that has the same name and same parameters as a method in its superclass, but with different implementation.
- This allows the subclass to provide its own implementation of a method that is already defined in its superclass.

Polymorphism & Method Overriding

Example of how to override a method in a Python class:

```
class Animal:
    def make_sound(self):
        print("The animal makes a sound.")
class Cat(Animal):
    def make_sound(self):
        print("Meow")
class Dog(Animal):
    def make_sound(self):
        print("Woof")
```

Polymorphism & Method Overriding

Example of how to override a method in a Python class:

```
# create objects of the classes
animal = Animal()
cat = Cat()
dog = Dog()
# calling the methods
animal.make_sound() #output: The animal makes a sound.
cat.make_sound()      # output: Meow
dog.make_sound()      # output: Woof
```

Polymorphism & Method Overriding

Example of how to override a method in a Python class:

OUTPUT

```
The animal makes a sound.
Meow
Woof
```

Polymorphism & Method Overriding

Example of how to override a method in a Python class:

In this example, we have defined an **Animal** class with a **make_sound()** method that simply prints a generic message. We then define two subclasses, **Cat** and **Dog**, which both inherit from **Animal** class and override the **make_sound()** method with their own implementation. When we create objects of these classes and call the **make_sound()** method, the appropriate implementation is used based on the type of object.

Polymorphism & Method Overriding

Example: Program to demonstrate the use of inheritance

```
class Person: # Here, Person is Base or Master or
Parent class
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def display(self):
        print("Name:", self.name)
        print("Age:", self.age)
```

Polymorphism & Method Overriding

Example: Program to demonstrate the use of inheritance

```
class Teacher(Person): # Here, Teacher is derived class-1 and
Person is Base class
    def __init__(self, name, age, experience, research_area):
        Person.__init__(self, name, age)
        self.experience = experience
        self.research_area = research_area
    def displaydata(self):
        Person.display(self)
        print("Experience:", self.experience)
        print("Research Area:", self.research_area)
```

Polymorphism & Method Overriding

Example: Program to demonstrate the use of inheritance

```
class Student(Person): # Here, Student is derived class-2 and
Person is Base class
    def __init__(self, name, age, course, year_semester):
        Person.__init__(self, name, age)
        self.course = course
        self.year_semester = year_semester
    def displaydata(self):
        Person.display(self)
        print("Course:", self.course)
        print("Year & Semester:", self.year_semester)
```

Polymorphism & Method Overriding

Example: Program to demonstrate the use of inheritance

```
print("------TEACHER DETAILS------:")
T = Teacher("Dr. R. Madana Mohana", 42, 17, "Artificial
Intelligence & Machine Learning")
T.displaydata()
print("------STUDENT DETAILS----:")
S = Student("RAM", 20, "B.E AI&DS-2", "I Year, II
Semester")
S.displaydata()
```

Polymorphism & Method Overriding

- In the *above program*, notice that **__init__()** method was defined in all the three classes. When this happens, the method in the derived class **overrides** that in the base class.
- This means that **__init__()** in **Teacher** and **Student** gets preference over the **__init__()** method in the **Person class**.

Polymorphism & Method Overriding

- Thus, ***Method Overriding*** is the ability of a class to change the implementation of a method provided by one of its ancestors. It is an important concept of Object Oriented Programming (OOP) since it exploits the power of ***inheritance***.

Polymorphism & Method Overriding

- Observe another thing that when we **override** a base method, we extend the functionality of the base class method.
- This is done by calling the method in the base class method from the derived class method and also adding additional statements in the derived class method.

Polymorphism & Method Overriding

- Instead of writing **Person.__init__(self, name, age)**, we could have also written **super().__init__(self, name, age)**.
- Here, **super()** is a built-in function that denotes the **base class**. So when we invoke a method using the **super()** function, then parent version of the method is called.
- In **Python**, every class is **inherited** from the base class object.

Polymorphism & Method Overriding

- In case of **multiple inheritance** (*a class derived from more than one base class*), we need to invoke the **super()** function in **__init__()** method of every class.

Polymorphism & Method Overriding

Example: Program to demonstrate the issue of invoking __init__() case of multiple inheritance.

```
class Base1(object):
    def __init__(self):
        print("Base1 Class..")
class Base2(object):
    def __init__(self):
        print("Base2 Class..")
class Derived(Base1, Base2):
    pass
D = Derived()
```

Polymorphism & Method Overriding

Example: Program to demonstrate the issue of invoking __init__() case of multiple inheritance.

OUTPUT:

```
Base1 Class..
```

Polymorphism & Method Overriding

- In the ***above method***, an object of **derived class** is made.
- Since there is no **__init__()** method in the derived class, the **__init__()** method of the first class gets called.
- But since, there is no call to **super()** function in the **__init__()** method of **Base1** class, no further **__init__()** method is invoked.

Polymorphism & Method Overriding

Example: Program to demonstrate the call of super() from __init__() of a base class

```
class Base1(object):
    def __init__(self):
        print("Base1 Class..")
        super(Base1, self).__init__()
class Base2(object):
    def __init__(self):
        print("Base2 Class..")
class Derived(Base1, Base2):
    pass
D = Derived()
```

Polymorphism & Method Overriding

Example: Program to demonstrate the call of super() from __init__() of a base class

```
OUTPUT:
Base1 Class..
Base2 Class..
```

Polymorphism & Method Overriding

Example: Program to call the __init__() methods of all the classes.

```
class Base1(object):
    def __init__(self):
        print("Base1 Class..")
        super(Base1, self).__init__()
class Base2(object):
    def __init__(self):
        print("Base2 Class..")
class Derived(Base1, Base2):
    def __init__(self):
        super(Derived, self).__init__()
        print("Derived Class..")
D = Derived()
```

Polymorphism & Method Overriding

Example: Program to call the __init__() methods of all the classes.

```
OUTPUT:
Base1 Class..
Base2 Class..
Derived Class..
```

Polymorphism & Method Overriding

isinstance() and issubclass() methods:

- **isinatance()** and **issubclass()** are two built-in functions which are very useful in **Python** to check instances.
- The **isinstance()** function returns **True** if the object is an instance the class or other classes derived from it.
- Similarly, the **issubclass()** checks for class inheritance.

Polymorphism & Method Overriding

Example: Program to demonstrate isinstance() and issubclass() methods

```
class Person: # Here, Person is Base or Master or Parent
class
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def display(self):
        print("Name:", self.name)
        print("Age:", self.age)
```

Polymorphism & Method Overriding

Example: Program to demonstrate isinstance() and issubclass() methods

```
class Teacher(Person): # Here, Teacher is derived class-1 and Person
is Base class
    def __init__(self, name, age, experience, research_area):
        Person.__init__(self, name, age)
        self.experience = experience
        self.research_area = research_area
    def displaydata(self):
        Person.display(self)
        print("Experience:", self.experience)
        print("Research Area:", self.research_area)
```

Polymorphism & Method Overriding

Example: Program to demonstrate isinstance() and issubclass() methods

```
class Student(Person): # Here, Student is derived class-2 and
Person is Base class
    def __init__(self, name, age, course, year_semester):
        Person.__init__(self, name, age)
        self.course = course
        self.year_semester = year_semester
    def displaydata(self):
        Person.display(self)
        print("Course:", self.course)
        print("Year & Semester:", self.year_semester)
```

Polymorphism & Method Overriding

Example: Program to demonstrate isinstance() and issubclass() methods

```
print("------TEACHER DETAILS------:")
T = Teacher("Dr. R. Madana Mohana", 42, 17, "Artificial
Intelligence & Machine Learning")
T.displaydata()
print("------STUDENT DETAILS----:")
S = Student("RAM", 20, "B.E AI&DS-2", "I Year, II
Semester")
S.displaydata()
```

Polymorphism & Method Overriding

Example: Program to demonstrate isinstance() and issubclass() methods

```
print("----DEMO ON isinatnce() and issubclass() methods---:")
print("T is a Teacher?", isinstance(T, Teacher))
print("T is a Person?", isinstance(T, Person))
print("T is an Integer?", isinstance(T, int))
print("T is an Object?", isinstance(T, object))
print("Person is a subclass of Teacher?", issubclass(Person,
Teacher))
print("Boolean is a subclass of int?", issubclass(bool, int))
```

Example: Program to demonstrate isinstance() and issubclass() methods

OUTPUT:

```
------TEACHER DETAILS------:
Name: Dr. R. Madana Mohana
Age: 42
Experience: 17
Research Area: Artificial Intelligence & Machine Learning
------STUDENT DETAILS----:
Name: RAM
Age: 20
Course: B.E AI&DS-2
Year & Semester: I Year, II Semester
```

Polymorphism & Method Overriding

Example: Program to demonstrate isinstance() and issubclass() methods

OUTPUT:

```
------DEMO ON isinatnce() and issubclass() methods------:
T is a Teacher? True
T is a Person? True
T is an Integer? False
T is an Object? True
Person is a subclass of Teacher? False
Boolean is a subclass of int? True
```