Python Inheritance

https://www.w3schools.com/python/python_inheritance.asp

Наследование позволяет нам определить класс, который наследует все методы и свойства другого класса.
Родительский класс — это наследуемый класс, также называемый базовым классом.
Дочерний класс — это класс, который наследуется от другого класса, также называемого производным классом.

Любой класс может быть родительским классом, поэтому синтаксис аналогичен созданию любого другого класса:

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname
  def printname(self):
    print(self.firstname, self.lastname)
#Use the Person class to create an object, and then execute the printname method:
x = Person("John", "Doe")
x.printname()
```

Чтобы создать класс, наследующий функциональность от другого класса, отправьте родительский класс в качестве параметра при создании дочернего класса:

```
class Student(Person):
  pass
```

Теперь класс Student имеет те же свойства и методы, что и класс Person.

Используйте `Student` класс для создания объекта, а затем выполните `printname` метод:

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname
  def printname(self):
    print(self.firstname, self.lastname)
class Student(Person):
  pass
x = Student("Mike", "Olsen")
x.printname()
```

```
Mike Olsen
```

Add the __init__() Function

Итак, мы создали дочерний класс, который наследует свойства и методы своего родителя.

Мы хотим добавить `__init__()` функцию в дочерний класс (вместо `pass` ключевого слова).

```
class Student(Person):
  def __init__(self, fname, lname):
    #add properties etc.
```

Когда вы добавите`__init__()` функцию, дочерний класс больше не будет наследовать родительскую `__init__()` функцию.

Дочерняя `__init__()` функция переопределяет наследование родительской `__init__()` функции.

Чтобы сохранить наследование родительской `__init__()` функции, добавьте вызов родительской `__init__()` функции:

```
class Student(Person):
  def __init__(self, fname, lname):
    Person.__init__(self, fname, lname)
```

Теперь мы успешно добавили функцию `__init__()` и сохранили наследование родительского класса, и мы готовы добавить функциональность в `__init__()` функцию.

Use the super() Function

В Python также есть `super()` функция, которая заставляет дочерний класс наследовать все методы и свойства своего родителя:

```
class Student(Person):
  def __init__(self, fname, lname):
    super().__init__(fname, lname)
```

Используя `super()` функцию, вам не нужно использовать имя родительского элемента, он автоматически наследует методы и свойства от своего родителя.

Add Properties

Add a property called `graduationyear` to the `Student` class:

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
```

```
    self.lastname = lname
  def printname(self):
    print(self.firstname, self.lastname)
class Student(Person):
  def __init__(self, fname, lname):
    super().__init__(fname, lname)
    self.graduationyear = 2019
x = Student("Mike", "Olsen")
print(x.graduationyear)

2019
```

В приведенном ниже примере год `2019` должен быть переменной и передаваться в `Student` класс при создании объектов ученика. Для этого добавьте еще один параметр в функцию __init__():

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname
  def printname(self):
    print(self.firstname, self.lastname)
class Student(Person):
  def __init__(self, fname, lname, year):
    super().__init__(fname, lname)
    self.graduationyear = year
x = Student("Mike", "Olsen", 2019)
print(x.graduationyear)

2019
```

Add Methods

Добавьте метод, вызываемый `welcome` в `Student` класс:

```
class Student(Person):
  def __init__(self, fname, lname, year):
    super().__init__(fname, lname)
    self.graduationyear = year
  def welcome(self):
    print("Welcome", self.firstname, self.lastname, "to the class of", self.graduationyear)
```

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname
  def printname(self):
    print(self.firstname, self.lastname)
class Student(Person):
```

```
  def __init__(self, fname, lname, year):
    super().__init__(fname, lname)
    self.graduationyear = year
  def welcome(self):
    print("Welcome", self.firstname, self.lastname, "to the class of", self.graduationyear) #- THIS
x = Student("Mike", "Olsen", 2019)
x.welcome()
```

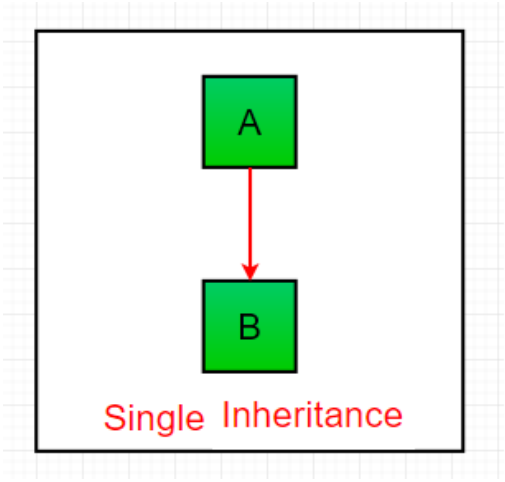
Если вы добавите в дочерний класс метод с тем же именем, что и у функции в родительском классе, наследование родительского метода будет переопределено.

Types of inheritance Python

https://www.geeksforgeeks.org/types-of-inheritance-python/

1. **Single Inheritance:**

Одиночное наследование позволяет производному классу наследовать свойства от одного родительского класса, что обеспечивает возможность повторного использования кода и добавление новых функций в существующий код.

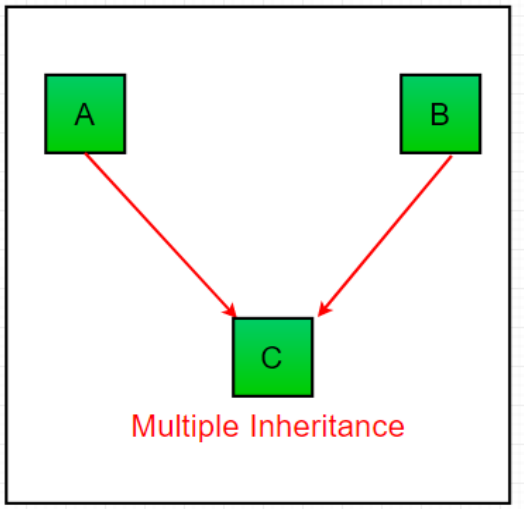



```python
# Python program to demonstrate
# single inheritance

# Base class
class Parent:
    def func1(self):
        print("This function is in parent class.")

# Derived class


class Child(Parent):
    def func2(self):
        print("This function is in child class.")


# Driver's code
object = Child()
object.func1()
object.func2()
```

```
Output:
This function is in parent class.
This function is in child class.
```

2. **Multiple Inheritance:**

Когда класс может быть получен более чем из одного базового класса, этот тип наследования называется множественным наследованием. При множественном наследовании все функции базовых классов наследуются производным классом.

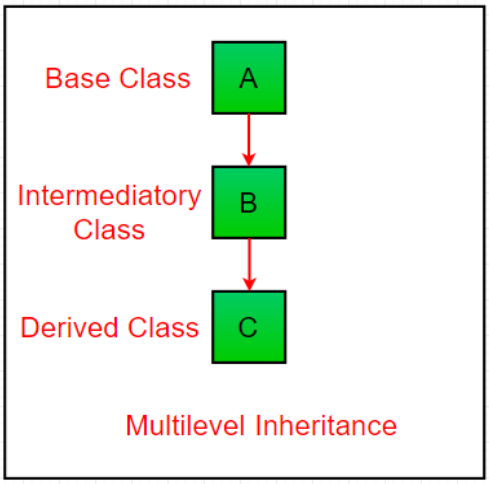



```python
# Python program to demonstrate
# multiple inheritance

# Base class1
class Mother:
    mothername = ""

    def mother(self):
        print(self.mothername)

# Base class2


class Father:
    fathername = ""

    def father(self):
        print(self.fathername)

# Derived class


class Son(Mother, Father):
    def parents(self):
        print("Father :", self.fathername)
        print("Mother :", self.mothername)


# Driver's code
s1 = Son()
s1.fathername = "RAM"
s1.mothername = "SITA"
s1.parents()
```

```
Output:
Father : RAM
Mother : SITA
```

3. **Multilevel Inheritance:**

При многоуровневом наследовании свойства базового класса и производного класса далее наследуются в новый производный класс. Это похоже на отношения, представляющие ребенка и дедушку.

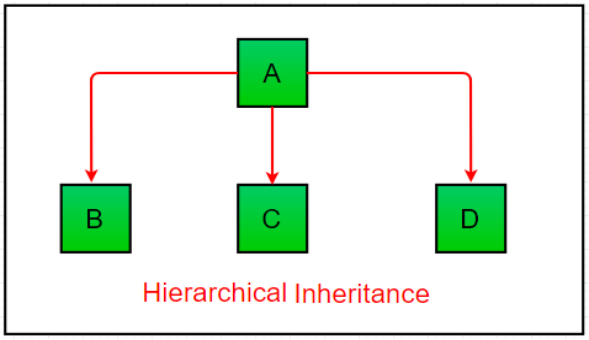



```
# Python program to demonstrate
# multilevel inheritance

# Base class


class Grandfather:

    def __init__(self, grandfathername):
        self.grandfathername = grandfathername

# Intermediate class


class Father(Grandfather):
    def __init__(self, fathername, grandfathername):
        self.fathername = fathername

        # invoking constructor of Grandfather class
        Grandfather.__init__(self, grandfathername)

# Derived class


class Son(Father):
    def __init__(self, sonname, fathername, grandfathername):
        self.sonname = sonname

        # invoking constructor of Father class
        Father.__init__(self, fathername, grandfathername)

    def print_name(self):
        print('Grandfather name :', self.grandfathername)
        print("Father name :", self.fathername)
        print("Son name :", self.sonname)


#  Driver code
s1 = Son('Prince', 'Rampal', 'Lal mani')
print(s1.grandfathername)
s1.print_name()
```

```
Output:
Lal mani
Grandfather name : Lal mani
Father name : Rampal
Son name : Prince
```

4. **Hierarchical Inheritance:**

Когда из одной базы создается более одного производного класса, этот тип наследования называется иерархическим наследованием. В этой программе у нас есть родительский (базовый) класс и два дочерних (производных) класса.

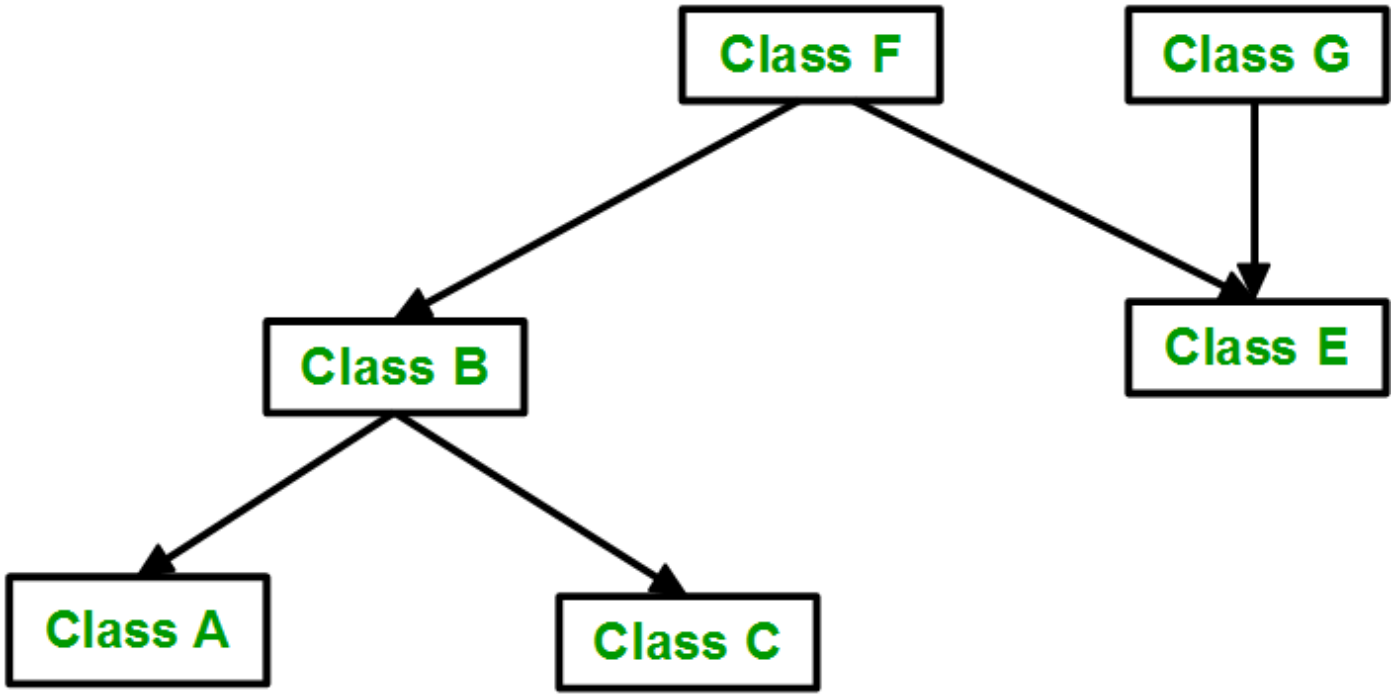



```
# Python program to demonstrate
# Hierarchical inheritance


# Base class
class Parent:
    def func1(self):
        print("This function is in parent class.")

# Derived class1


class Child1(Parent):
    def func2(self):
        print("This function is in child 1.")

# Derivied class2


class Child2(Parent):
    def func3(self):
        print("This function is in child 2.")


# Driver's code
object1 = Child1()
object2 = Child2()
object1.func1()
object1.func2()
object2.func1()
object2.func3()
```

```
Output:
This function is in parent class.
This function is in child 1.
This function is in parent class.
This function is in child 2.
```

5. **Hybrid Inheritance:**

Наследование, состоящее из нескольких типов наследования, называется гибридным наследованием.




```
# Python program to demonstrate
# hybrid inheritance


class School:
    def func1(self):
        print("This function is in school.")


class Student1(School):
    def func2(self):
        print("This function is in student 1. ")


class Student2(School):
    def func3(self):
        print("This function is in student 2.")


class Student3(Student1, School):
    def func4(self):
        print("This function is in student 3.")


# Driver's code
object = Student3()
object.func1()
object.func2()
```

```
Output:
This function is in school.
This function is in student 1.
```