# Chapter – 16 Inheritance

**Contents**



## Chapter-16 Inheritance

တစ်ခါတစ်လေ coding လုပ်နေတုန်း အရင်တုန်းက တွေ့ဘူးခဲ့တဲ့ object တစ်ခုက အခု လုပ်ချင်တာနဲ့ ခပ်ဆင်ဆင်တူညီနေတာမျိုး သို့မဟုတ် လုံးဝ တူညီနေတာမျိုး ကြုံဘူးမှာပါ။ ရှိပြီးသား class တစ်ခုက အခု လုပ်ချင်နေတာနဲ့ တူညီနေတယ်ဆိုရင် လုပ်နိုင်တဲ့နည်းတွေက

(၁) အသစ် class တစ်ခု အစမှ စရေးခြင်း

(၂) ရှိပြီးသား class ကို ကူးထည့်ခြင်း(copy and past လုပ်ခြင်း)

(၃) Inheritance လုပ်ခြင်း

အကောင်းဆုံးနည်းသည် inheritance လုပ်ခြင်း ဖြစ်သည်။ ရှိပြီးသား class တစ်ခုကို အခြေခံပြီး အနည်းငယ် ထပ်ထည့်ခြင်း(additions)၊ ပြောင်းလဲခြင်း(changing) တို့ပြုလုပ်နိုင်သည်။ ကုဒ်ကို ပြန်လည် အသုံးပြုရန် အကောင်းဆုံးနည်းလမ်း(excellent way to reuse code) ဖြစ်သည်။

Inheritance ဆိုသည်မှာ ရှိပြီးသား class တစ်ခုမှ ကုဒ်များအားလုံးကို ကော်ပီကူးစရာမလိုဘဲ ၄င်းကုဒ်များ အလိုအလျောက် အသုံးပြုပြီး class အသစ် တစ်ခု တည်ဆောက်လိုက်ခြင်း ဖြစ်သည်။ class အသစ်အတွက် ထပ်ထည့်ရန် သို့မဟုတ် ပြောင်းလဲရန်လိုအပ်သည့် အရာကိုသာ လုပ်ပေးရန်((ကုဒ်များကို ထည့်ပေးရန်) လိုသည်။

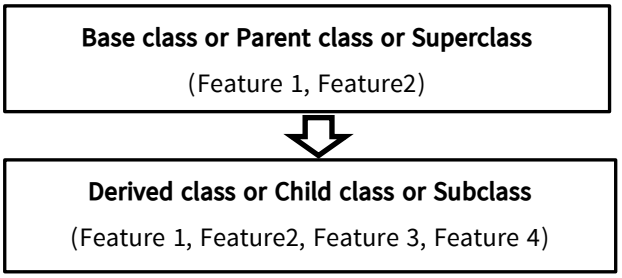
OOP ၏အခြေခံသဘောတရား လေးခု ဖြစ်သည့် Abstraction, Encapsulation, Inheritance နှင့် Polymorphism တို့အနက်မှ Inheritance အကြောင်းကို လေ့လာကြပါစို့။

ပုံမှန်အားဖြင့် new class သည် parent class မှ အရာအားလုံးကို အမွေဆက်ခံ (inherit လုပ်)သည်။

**16. Inheritance**

Inheritance ဆိုသည်မှာ class တစ်ခုသည် တခြား class များမှ ဂုဏ်သတ္တိများ(properties) ၊ feature များကို လက်ခံယူနိုင်ခြင်း ဖြစ်သည်။ ရှိပြီးသား class တစ်ခု သို့မဟုတ် class များမှ ဂုဏ်သတ္တိများ(properties) ၊ feature များကို ယူ၍ နောက်ထပ် class တစ်ခုကို အလွယ်တကူ တည်ဆောက်နိုင်သည်။ ဟုခေါ်သည်။

- အသစ်ဖြစ်ပေါ်လာမည့် class ကို derived class သို့မဟုတ် child class ဟုခေါ်သည်။
- ရှိပြီးသား class ကို base class သို့မဟုတ် parent class ဟုခေါ်သည်။




**16.1 Add a Method**

Child class ထဲတွင် parent class မှ feature များသာမက feature အသစ်များ method များ ထပ်ထည့်နိုင်သည်။ parent class တွင် မရှိသည့် method များကိုလည်း child class ထဲတွင် ထပ်ထည့်နိုင်သည်။

Inheritance လုပ်ခြင်းကြောင့် ကုဒ်များကို အသစ်ထပ်ရေးစရာ မလိုဘဲ ပြန်လည်အသုံးပြု(Code reusability) နိုင်သည်။

Class တစ်ခုက child class ဖြစ်နေတယ်ဆိုရင်ဘယ် parent class ကနေ Derived လုပ်ထားသလဲဆိုတာက သိနိုင်ဘို့ child class မှာ parent class ကို ဖော်ပြဘို့လိုသည်။ ထိုမှသာ မည်သူ၏ child class ဖြစ်ကြောင်း အလွယ်တကူ သိနိုင်လိမ့်မည်။ အောက်မှာ class တစ်ခု inheritance လုပ်ပုံကို ဥပမာတွင် child class ကို define လုပ်သည့်အခါ () ထဲတွင် parent class နာမည်ထည့်ပေးရသည်။ `class Child(Parent):`

**Inheritance Example-1**

```
class Parent:
    def func1(self):
        print('This is a parent function.')


class Child(Parent): #မည်သည့် parent class မှ inheritance လုပ်ထားကြောင်း ဖော်ပြ
    def func2(self):
        print('This is a child function.')

ob = Child()
ob.func1()
ob.func2()
```

```
This is a parent function.
This is a child function.
```

(1) child class က parent class မှာ ရေးထားတဲ့ **func1** ကို ဘာမှ ရေးစရာမလိုပဲ ပြန်သုံးလို့ရသည်။

**16.2 Override a Method**

Parent class တွင် ရှိသည့် method အားလုံးကို child class က override လုပ်နိုင်သည်။ Parent class ၏ __init__() method ကိုလည်း override လုပ်နိုင်သည်။

**16.3 Override __init __() method**

**__init __()** method ဆိုတာ class တစ်ခုအတွင်း၌ `object` လေးတွေ တည်ဆောက်ဘို့ လိုသည့် အခါတိုင်း ခေါ်သုံးသည့် method တစ်ခုဖြစ်သည်။ Child class က parent class မှာ ရေးထားတဲ့ __init __() method ကို လှမ်းပြီးတော့ access လုပ်လို့မရပါဘူး။ ထို့ကြောင့် Child class __init __() method က parent class မှာရှိတဲ့ __init __() method ကို access လုပ်ဘို့ **`Parent.__init__()`** ကုဒ်တစ်ကြောင်းထည့်ပေး ရသည်။

```
class Parent:
    def __init__(self, fage , fname):
        self.name = fname
        self.age = fage

    def view(self):
            print( self.age, self.name)

class Child(Parent):              # () ထဲတွင် Parent class ကို ထည့်ပေးရသည်။
    def __init__(self, fage , fname):
        Parent.__init__(self, fage , fname)    #Parent class __init__
        self.last_name = "Python"

    def view(self):
            print( self.age, self.name, self.last_name)

ob = Child(99 ,'Easy')
ob.view()
```

```
99 Easy Python
```

**`Parent.__init__(self, fage , fname)`** သည် child class က __init __() method သည် parent class မှာရှိသည့် __init __() method ကို access လုပ်ရန်အတွက် ကုဒ်ဖြစ်သည်။ **`help(Parent)`** ဖြင့်၏ အချက်အလက်များကို ကြည့်နိုင်သည်။ Parent class ၏ အချက်အလက်များကို ကြည့်နိုင်သည်။

```
help(Child)
```

```
Help on class Child in module __main__:

class Child(Parent)
 |  Child(fage, fname)
 |
 |  Method resolution order:
 |      Child
 |      Parent
 |      builtins.object
 |
 |  Methods defined here:
 |
```

```
 |  __init__(self, fage, fname)
 |      Initialize self. See help(type(self))for accurate signature.
 |
 |  view(self)
```

**16.1 Type of inheritance**

- Single
- Multiple
- Multilevel
- Hierarchical

**16.2.1 Single**

Child class တစ်ခု နှင့် parent class တစ်ခုသာ ပါဝင်သည်။ အထက်တွင် ဖော်ပြခဲ့ပြီးသည့် Inheritance Example-1 သည် single inheritance ဥပမာ ဖြစ်သည်။

**16.2.2 Multiple**

တစ်ခုထက်ပိုများသည့် parent class များ ပါဝင်သည်။ အောက်တွင် Parent1 နှင့် Parent2 တို့မှ inheritance လုပ် ထားသည့် Child class ကို ဖော်ပြထားသည်။ Child class ကို define လုပ်သည့်အချိန်တွင် parent (၂)ခုစလုံးကို () ထဲတွင် ဖော်ပြပေးရသည်။ `class Child(Parent1, Parent2):`

```
class Parent1:
    def func1(self):
        print('This is a parent1 class function1.')

class Parent2:
    def func2(self):
        print('This is a parent2 class function2.')

class Child(Parent1, Parent2):    # parent (၂)ခုစလုံးကို ဖော်ပြပေးရသည်။
    def func3(self):
        print('This is a child class function3.')

ob = Child()
ob.func1()
ob.func2()
ob.func3()
```

```
This is a parent1 class function1.
This is a parent2 class function2.
This is a child class function3.
```

Parent1 class မှ function1 နှင့် parent2 class မှ function2 တို့၏ ကုဒ်ကို ထပ်ရေးစရာမလိုဘဲ ထိစရာမလိုဘဲ အသုံးပြုခွင့် ရရှိသည်။

### 16.2.3 Multilevel

Child class တစ်ခုသည် parent class အဖြစ် ဆောင်ရွက်သည့်အခါ သို့မဟုတ် child class တစ်ခုသည် တစ်ချိန်တည်း၌ တခြား class တစ်ခုအတွက် child class အဖြစ်တည်ရှိနေပြီး၊ တခြား class တစ်ခုအတွက် parent class အဖြစ် တည်ရှိနေသည့်အခါမျိုး ဖြစ်သည်။

Parent1 class → Parent2 class → Child class

Parent2 class သည် Parent1 class မှ function1ကို inheritance လုပ်သည်။ Child class သည် Parent2 class မှ function2 ကို inheritance လုပ်သည်။ ထို့ကြောင့် Child class သည် level မတူသည့် Parent1 class များကို inheritance လုပ်နေခြင်း ဖြစ်သည်။

```
class Parent1:
    def func1(self):
        print('This is a parent1 class function1.')

class Parent2(Parent1):     # Parent1 ဆီမှ inheritance လုပ်သည်။
    def func2(self):
        print('This is a parent2 class function2.')

class Child(Parent2):       # Parent2 ဆီမှ inheritance လုပ်သည်။
    def func3(self):
        print('This is a child class function3.')

ob = Child()
ob.func1()
ob.func2()
ob.func3()
```

```
This is a parent1 class function1.
This is a parent2 class function2.
This is a child class function3.
```

Parent2 တွင် ရှိသည့် object သည် သူ့၏ func2 နှင့် Parent1 ထံမှ func1 ကိုသာ ရထားသည်။

```
ob_1 = Parent2()
ob_1.func1()
ob_1.func2()
```

```
This is a parent1 class function1.
This is a parent2 class function2.
```

### 16.2.4 Hierarchical

Inheritance တစ်မျိုးထက် ပိုများလျှင် Hierarchical ဟုသတ်မှတ်သည်။ တစ်ချိန်တည်း Single Inheritance နှင့် Multiple Inheritance တို့ ဖြစ်ပေါ်နေသည့်အခါမျိုး ဖြစ်သည်။

**16.2.5 Super function**

အထက်တွင် child class သည် parent ကို override လုပ်သည့်နည်းနှင့် add method ကို ဖော်ပြခဲ့ဖြစ်သည်။ ထိုသို့ add method နှင့် override method ကို မသုံးဘဲ parent method ကို တိုက်ရိုက် ခေါ်သည့်နည်း (call directly)ကိုလည်း အသုံးပြုနိုင်သည်။ Super function ဆိုသည်မှာ parent class ကို တိုက်ရိုက် ခေါ်သည့်နည်း (call directly) ဖြစ်သည်။

**Super() Example-1 ( without __init__() method)**

အောက်တွင် super() ကို သုံးသည့် ဥပမာတစ်ခု ပြထားသည်။ ထိုဥပမာတွင် __init__() method သုံးရန် မလိုပါ။ အဘယ်ကြောင့်ဆိုသော် Child class ထဲသို့ parameter အသစ်(parent class ထဲတွင် မပါသည် parameter) ထပ်ထည့်ရန် မလိုသောကြောင့်ဖြစ်သည်။ parameter အသစ် ထပ်ထည့်ရန် မလိုပါက parent class မှ __init__() method ကို override လုပ်ရန် မလိုပါ။

```
class Parent:
    def func1(self):
        print('This is function1.')

class Child(Parent):
    def func2(self):
        super().func1()               #Super
        print('This is function2.')

ob = Child()
ob.func2()
```

```
This is function1.
This is function2.
```

**Super() Example-2 ( with __init__() method)**

အောက်တွင် super() ကို သုံးသည့် ဥပမာတွင် parameter အသစ် (parent class ထဲတွင် မပါသည် parameter) ထပ်ထည့်ရန်လိုသည်။ ထို့ကြောင့် __init__() method ကို သုံးသည်။ parameter အသစ် ထပ်ထည့်ရန် လိုအပ်ပါက parent class မှ __init__() method ကို override လုပ်ရန် လိုသည်။

```
class Person():                # or class Person:
    def __init__(self, name):
        self.name = name

# additional email parameter ထည့်ရန် subclass ထဲတွင် __init__() ခေါ်သည်။
class EmailPerson(Person):
    def __init__(self, name, email):
        super().__init__(name)
        self.email = email

sturdent1 = EmailPerson('Lin Htet', 'lin_htet@python.com')
print(sturdent1.name , sturdent1.email)
```

```
Lin Htet lin_htet@python.com
```

- super() သည် parent class ၏ definition ကို ရရှိသည်။ အပေါ်က ဥပမာတွင် Parent ဖြစ်သည်။
- parameter အသစ် ထပ်ထည့်ရန် parent class မှ __init__() method သည် child class မှ Person.__init__() method ကို လှမ်းခေါ်သည်။ ထည့်လိုသည့် parameter အသစ်များ ထည့်နိုင်သည်။
- email ဆိုသည့် parameter အသစ်ထည့်သောကြောင့် `self.email = email line` ထည့်ပေးရသည်။
- child class သည် လုပ်မည့်အလုပ်(method)သည် parent class မတူသည့်အခါ super() ကို အသုံးသည်။

**Super() Example-2**

```
class Person:              #class Person(object): ကဲ့သို့လည်း ရေးနိုင်သည်။
    def __init__(self, name, sex):
        self.name = name
        self.sex = sex

    def say_hello(self):
        return 'Hello Python. I am ' + self.name

    def show_me(self):
        return 'my name is %s , sex is %s' % (self.name, self.sex)

class Student(Person):
    # Student class ကို define လုပ်သည့်အခါ score attribute ထည့်ပေးသည်။
    def __init__(self, name, sex, score):
        #super(Student, self).__init__(name, sex) ဟုလည်း ရေးနိုင်သည်။
        super().__init__(name, sex)
        self.score = score

    def student(self):
        ""add subclass method"""
        return 'I am a student, my name is %s' % self.name

    def show_me(self):
        """Override parent class method"""
        return 'my name is %s , sex is %s , my final score is %d' %
               (self.name, self.sex, self.score)

student_1 = Student("Lin Htet", 'Male', 20)
print(student_1.name)
```

```
Lin Htet
```

student_1.__dict__ ဖြင့် student_1 ၏ ဒေတာများကို ကြည့်သည်။

```
print(student_1.__dict__)
```

```
{'name': 'Lin Htet', 'sex': 'Male', 'score': 20}
```

Parent class မှ inherite လုပ်ထားသည့် subclass method ကို ကြည့်သည်။ subclass method သည် `say_hello()` ဖြစ်သည်။

```
print(student_1.say_hello())
```

```
Hello Python. I am Lin Htet
```

Add လုပ်ထားသည့် subclass method ကို ကြည့်သည်။ (Add method)

```
print(student_1.student())
```

```
I am a student, my name is Lin Htet.
```

Parent class methods ကို override လုပ်ထားသည့် subclasses ကို ကြည့်သည်။ (override method)

```
print(student_1.show_me())
```

```
my name is Lin Htet , sex is Male , my final score is 20
```

**16.3 Public and Private**

အချို့သော object-oriented language များတွင် ပြင်ပမှသူများ access လုပ်မရနိုင်သည့် private object များရှိသည်။ ထိုအခါ ပရိဂရမ်မာများက private object များ၏ attribute များကို access လုပ်ရန်အတွက် write getter methods နှင့် setter method များရေးကြရသည်။ getter method(get method) သည် private attribute များကို ဖတ်(read လုပ်)ရန် အတွက်ဖြစ်သည်။ setter method (set method)သည် private attribute များကို ရေး(write လုပ်)ရန် အတွက်ဖြစ်သည်။

Python တွင် getter methods နှင့် setter method များ မလိုအပ်ပါ။ Python တွင် attribute များ အားလုံးနှင့် method များအားလုံးသည် public ဖြစ်သည်။ အကယ်၍ သင့်က public attribute နှင့် method ကြောင့် စိတ်အနှောက်အယှက်ဖြစ်လျှင် private အဖြစ်ထားပြီး getters နှင့် setter များ ရေးပြီး အသုံးပြုနိုင်သည်။ Pythonic နည်းဖြစ်သည့် property များ အသုံးပြုပါ။

-End-