

Ch18-Inheritance

September 10, 2025

1 Inheritance

- <http://openbookproject.net/thinkcs/python/english3e/inheritance.html>
- https://www.python-course.eu/python3_inheritance.php
- powerful feature that facilitates code reuse mimicking real-world phenomena
- ability to define a new class (child) that is modified version of an existing class (parent)
- can add new methods and properties to a class without modifying the existing class
- some imitation(s) if inheritance:
 - can make programs difficult to read
 - when method is invoked, it is sometimes not clear where to find its definition esp. in a large project relevant code may be scattered among several modules
- see better example (a hand of cards) in the text
- syntax:

```
class childClassName(parentClass1, baseClass2, ...):  
    #code (attributes and methods)  
    pass
```

1.1 Single Inheritance

- supported by almost all OOP languages

```
[1]: # by default all python class implicitly inherit from object base class  
class A(object):  
    def __init__(self):  
        self.a = "A"  
  
    def printMe(self):  
        print("A's printMe called!")  
        print('a = {}'.format(self.a))  
  
    def sayHi(self):  
        print('{} says HI!'.format(self.a))
```

```
[2]: obja = A()
obja.printMe()
obja.sayHi()
```

A's printMe called!

a = A

A says HI!

```
[3]: # single inheritance
class B(A):
    def __init__(self):
        # must explicitly invoke base classes constructors
        # to inherit properties/attributes
        #A.__init__(self) # try commenting this out
        self.b = 'B'

    def update(self):
        print("Attributes before modification: {} and {}".format(self.a, self.b))
        self.a = 'AAA' #can modify inherited attributes
        print("Attributes after modification: {} and {}".format(self.a, self.b))

    # overrides inherited printMe
    def printMe(self):
        print("B's printMe called")
        print('a = {}'.format(self.a))
```

```
[4]: objb = B()
# shows that A's properties are inherited by B
objb.update()
```

```
-----
AttributeError                                Traceback (most recent call last)
/var/folders/4f/1pkkv7h960j42p0ppgk9n4ywjr6t_b/T/ipykernel_95581/1312489048.py
↳in <module>
      1 objb = B()
      2 # shows that A's properties are inherited by B
----> 3 objb.update()

/var/folders/4f/1pkkv7h960j42p0ppgk9n4ywjr6t_b/T/ipykernel_95581/1596476040.py
↳in update(self)
      8
      9     def update(self):
----> 10         print("Attributes before modification: {} and {}".format(self.a,
↳self.b))
      11         self.a = 'AAA' #can modify inherited attributes
      12         print("Attributes after modification: {} and {}".format(self.a,
↳self.b))
```

```
AttributeError: 'B' object has no attribute 'a'
```

```
[5]: # object a's properties are independent from object b's properties
print("obja's property a = {}".format(obja.a))
print("objb's property a = {}".format(objb.a))
```

```
obja's property a = A
```

```
-----
AttributeError                                Traceback (most recent call last)
/var/folders/4f/1pkkv7h960j42p0ppgk9n4ywj6t_b/T/ipykernel_95581/1308486093.py
↳in <module>
      1 # object a's properties are independent from object b's properties
      2 print("obja's property a = {}".format(obja.a))
----> 3 print("objb's property a = {}".format(objb.a))

AttributeError: 'B' object has no attribute 'a'
```

```
[6]: # B inherits A's sayHi()
# what is the output of the following?
objb.sayHi()
```

```
-----
AttributeError                                Traceback (most recent call last)
/var/folders/4f/1pkkv7h960j42p0ppgk9n4ywj6t_b/T/ipykernel_95581/1749312439.py
↳in <module>
      1 # B inherits A's sayHi()
      2 # what is the output of the following?
----> 3 objb.sayHi()

/var/folders/4f/1pkkv7h960j42p0ppgk9n4ywj6t_b/T/ipykernel_95581/949562657.py i:
↳sayHi(self)
      9
     10     def sayHi(self):
----> 11         print('{} says HI!'.format(self.a))

AttributeError: 'B' object has no attribute 'a'
```

1.2 Overriding

- child class can redefine method that are inherited from parent class with the same name
- e.g., printMe() method in class B overrides A's printMe
- A's printme can still be called
 - syntax
ClassName.method(object)

```
[7]: objb.printMe()
```

B's printMe called

```
-----  
AttributeError                                Traceback (most recent call last)  
/var/folders/4f/1pkkv7h960j42p0ppgk9n4ywjr6t_b/T/ipykernel_95581/581139490.py i:  
  ↳<module>  
----> 1 objb.printMe()  
  
/var/folders/4f/1pkkv7h960j42p0ppgk9n4ywjr6t_b/T/ipykernel_95581/1596476040.py  
  ↳in printMe(self)  
    15     def printMe(self):  
    16         print("B's printMe called")  
----> 17         print('a = {}'.format(self.a))  
  
AttributeError: 'B' object has no attribute 'a'
```

```
[8]: A.printMe(obja)
```

A's printMe called!
a = A

```
[9]: A.printMe(objb)
```

A's printMe called!

```
-----  
AttributeError                                Traceback (most recent call last)  
/var/folders/4f/1pkkv7h960j42p0ppgk9n4ywjr6t_b/T/ipykernel_95581/3977030366.py  
  ↳in <module>  
----> 1 A.printMe(objb)  
  
/var/folders/4f/1pkkv7h960j42p0ppgk9n4ywjr6t_b/T/ipykernel_95581/949562657.py i:  
  ↳printMe(self)  
    6     def printMe(self):  
    7         print("A's printMe called!")  
----> 8         print('a = {}'.format(self.a))  
    9  
   10     def sayHi(self):  
  
AttributeError: 'B' object has no attribute 'a'
```

```
[10]: # C inherits from B which inherits from A  
class C(B):  
    def __init__(self):
```

```

    B.__init__(self)
    self.c = 'C'

    def printMe(self):
        print("C's printMe called:")
        print("Attributes are {}, {}, {}".format(self.c, self.b, self.a))

```

```

[11]: c1 = C()
      c1.printMe()

```

C's printMe called:

```

-----
AttributeError                                Traceback (most recent call last)
/var/folders/4f/1pkkv7h960j42p0ppgk9n4ywjr6t_b/T/ipykernel_95581/241926923.py i
↳<module>
      1 c1 = C()
----> 2 c1.printMe()

/var/folders/4f/1pkkv7h960j42p0ppgk9n4ywjr6t_b/T/ipykernel_95581/1912106273.py
↳in printMe(self)
      7     def printMe(self):
      8         print("C's printMe called:")
----> 9         print("Attributes are {}, {}, {}".format(self.c, self.b, self.a)
      10

AttributeError: 'C' object has no attribute 'a'

```

```

[12]: # sayHi() inherited from A
      c1.sayHi()

```

```

-----
AttributeError                                Traceback (most recent call last)
/var/folders/4f/1pkkv7h960j42p0ppgk9n4ywjr6t_b/T/ipykernel_95581/1158629671.py
↳in <module>
      1 # sayHi() inherited from A
----> 2 c1.sayHi()

/var/folders/4f/1pkkv7h960j42p0ppgk9n4ywjr6t_b/T/ipykernel_95581/949562657.py i
↳sayHi(self)
      9
     10     def sayHi(self):
----> 11         print('{} says HI!'.format(self.a))

AttributeError: 'C' object has no attribute 'a'

```

```
[13]: c1.update()
```

```
-----  
AttributeError                                Traceback (most recent call last)  
/var/folders/4f/1pkkv7h960j42p0ppgk9n4ywj6t_b/T/ipykernel_95581/1031084644.py  
  ↳ in <module>  
----> 1 c1.update()  
  
/var/folders/4f/1pkkv7h960j42p0ppgk9n4ywj6t_b/T/ipykernel_95581/1596476040.py  
  ↳ in update(self)  
      8  
      9     def update(self):  
----> 10         print("Attributes before modification: {} and {}".format(self.a,  
  ↳ self.b))  
      11         self.a = 'AAA' #can modify inherited attributes  
      12         print("Attributes after modification: {} and {}".format(self.a,  
  ↳ self.b))  
  
AttributeError: 'C' object has no attribute 'a'
```

1.3 Multiple Inheritance

- Python allows a class to derive/inherit from multiple base classes
 - similar to C++
- Java doesn't allow it (it's messy!)

```
[14]: # not required to explicitly inherit from object class  
class D:  
    def __init__(self):  
        self.a = 'AAAAA'  
        self.d = 'D'  
  
    def scream(self):  
        print("D's scream() called:")  
  
# class E inherits from class C and D  
class E(C, D):  
    def __init__(self):  
        # the order in which the base constructors are called matters!  
        # same attributes of prior constructors are overridden by later  
  ↳ constructors  
        # e.g., try switching D and C's constructor calls  
        D.__init__(self)  
        C.__init__(self)  
        self.e = 'E'
```

```

def printMe(self):
    print("E's printMe called:")
    print("Attributes are {}, {}, {}, {}, {}".format(self.e, self.d, self.
↵c, self.b, self.a))

```

```

[15]: e1 = E()
      e1.printMe()

```

```

E's printMe called:
Attributes are E, D, C, B, AAAAA

```

```

[16]: e1.scream()

```

```

D's scream() called:

```

```

[17]: e1.sayHi()

```

```

AAAAA says HI!

```

1.4 abc module - Abstract Base Classes and abstract methods

- allows to define ABCs with abstract methods @abstractmethod decorators
- an abstract class is a class that contains at least one **abstract method**
- an **abstract method** is a method that is declared, but not implemented
- these are base classes that must be extended/derived from and can't be instantiated
 - derived/children classes implement the details of the methods

```

[1]: from abc import ABC, abstractmethod

class Shape(ABC):
    def __init__(self):
        pass

    @abstractmethod
    def area(self):
        pass

    def sayHi(self):
        print('hello from Shape ABC')

[2]: # can't instantiate an abstract class
      a = Shape()

```

```

-----
TypeError                                Traceback (most recent call last)
/var/folders/4f/1pkkv7h960j42p0ppgk9n4ywj6t_b/T/ipykernel_49786/3852130175.py
↵in <module>
      1 # can't instantiate an abstract class
----> 2 a = Shape()

```

```
TypeError: Can't instantiate abstract class Shape with abstract method area
```

```
[3]: a.area()
```

```
-----  
NameError                                Traceback (most recent call last)  
/var/folders/4f/1pkkv7h960j42p0ppgk9n4ywjr6t_b/T/ipykernel_49786/3969111684.py  
  ↳ in <module>  
----> 1 a.area()  
  
NameError: name 'a' is not defined
```

```
[4]: class Rectangle(Shape):  
      def __init__(self, length=5, wid=5):  
          Shape.__init__(self)  
          self.length = length  
          self.width = wid  
  
      # uncomment area function to implement it in this derived class  
      """  
      def area(self):  
          return self.length * self.width  
      """
```

```
[5]: r = Rectangle()
```

```
-----  
TypeError                                Traceback (most recent call last)  
/var/folders/4f/1pkkv7h960j42p0ppgk9n4ywjr6t_b/T/ipykernel_49786/1069889248.py  
  ↳ in <module>  
----> 1 r = Rectangle()  
  
TypeError: Can't instantiate abstract class Rectangle with abstract method area
```

```
[6]: r.area()
```

```
-----  
NameError                                Traceback (most recent call last)  
/var/folders/4f/1pkkv7h960j42p0ppgk9n4ywjr6t_b/T/ipykernel_49786/1011413101.py  
  ↳ in <module>  
----> 1 r.area()  
  
NameError: name 'r' is not defined
```



```
[7]: r.sayHi()
```

```
-----  
NameError                                Traceback (most recent call last)  
/var/folders/4f/1pkkv7h960j42p0ppgk9n4ywj6t_b/T/ipykernel_49786/520891810.py i:  
  ↳<module>  
----> 1 r.sayHi()  
  
NameError: name 'r' is not defined
```

```
[ ]:
```