

# Ch18-Inheritance

October 30, 2020

## 1 Inheritance

- <http://openbookproject.net/thinkcs/python/english3e/inheritance.html>
- [https://www.python-course.eu/python3\\_inheritance.php](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

```
[116]: # 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))
```

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

A's printMe called!  
a = A  
A says HI!

```
[118]: # 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))
```

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

Attributes before modification: A and B  
Attributes after modification: AAA and B

```
[121]: # 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  
objb's property a = AAA

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

AAA says HI!

## 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)

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

```
B's printMe called  
a = AAA
```

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

```
A's printMe called!  
a = A
```

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

```
A's printMe called!  
a = AAA
```

```
[130]: # 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))
```

```
[131]: c1 = C()  
c1.printMe()
```

```
C's printMe called:  
Attributes are C, B, A
```

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

```
A says HI!
```

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

```
Attributes before modification: A and B  
Attributes after modification: AAA and B
```

### 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!)

```
[153]: # 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))
```

```
[154]: e1 = E()
e1.printMe()
```

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

```
[155]: e1.scream()
```

```
D's scream() called:
```

```
[156]: e1.sayHi()
```

```
A says HI!
```

### 1.4 abc module - Abstract Base Classes

- allows to define ABCs with abstract methods @abstractmethod decorators

[ ]: