# Classes

講者：Isaac

## Outline

- What's Class
  - Encapsulation
  - Inheritance
  - Multiple inheritance
  - Polymorphism

## Class

## OOP

- Object-Oriented Programming
- factor code to minimize redundancy,
- write programs by *customizing* existing code instead of changing it in-place.

## Class

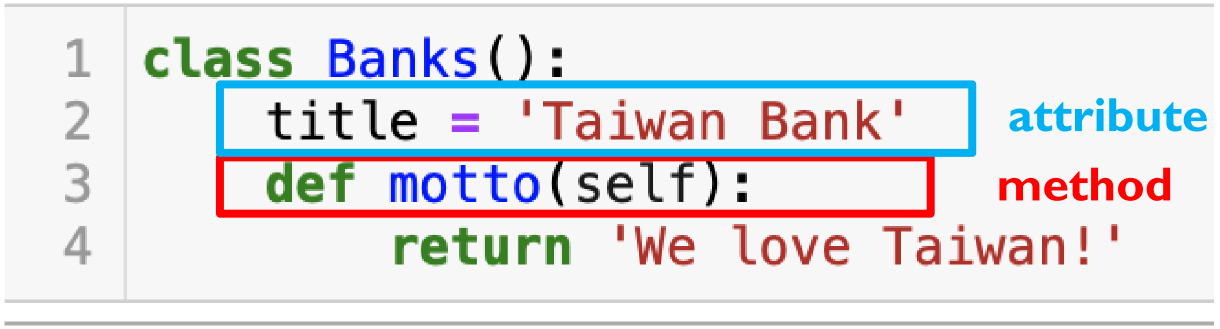
- Classes are python's main object-oriented programming(OOP) tool.
- Classes Syntax:

```
class Class_name():
    statement_1
    …
    statement_n
```

**Uppercase on first character**

## Class

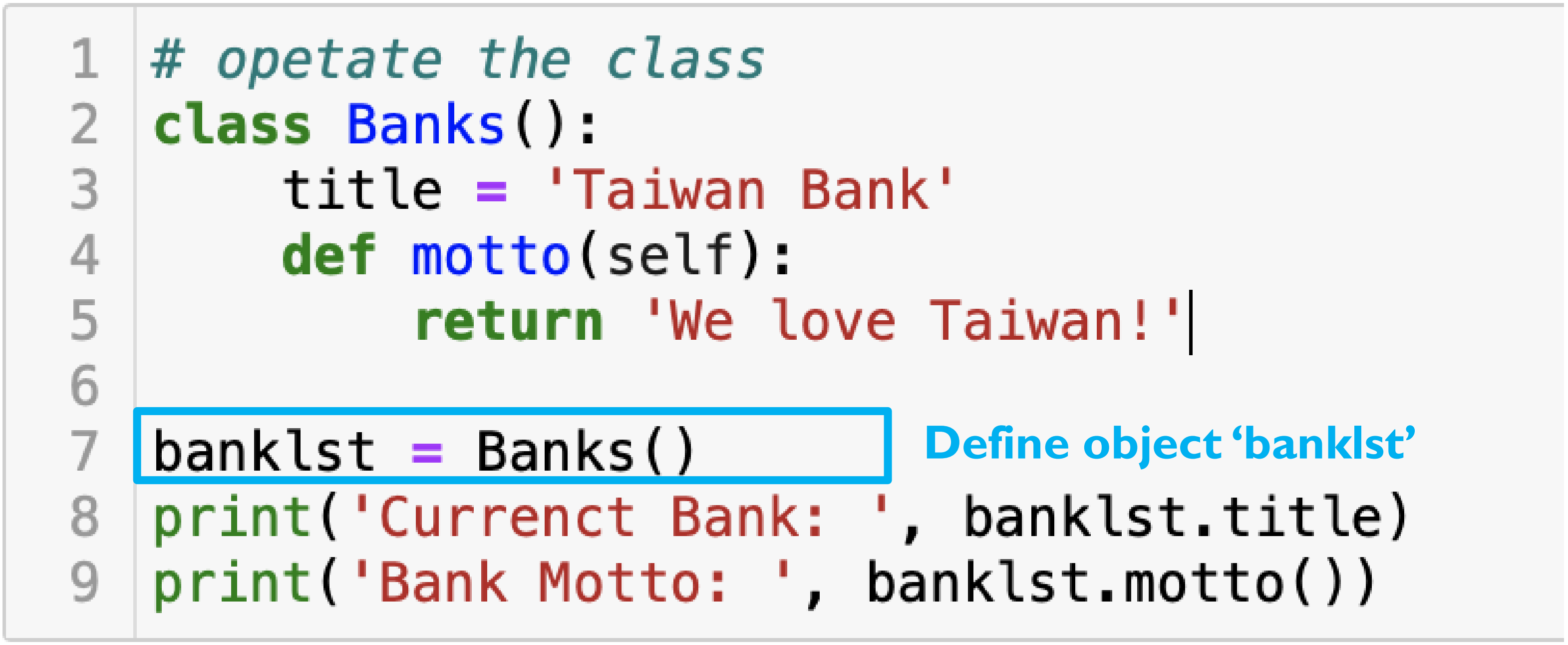
- Example
  - A simple example for defining class.



```python
class Banks():
    title = 'Taiwan Bank'
    def motto(self):
        return 'We love Taiwan!'
```

## Class

- Example
  - Class calls



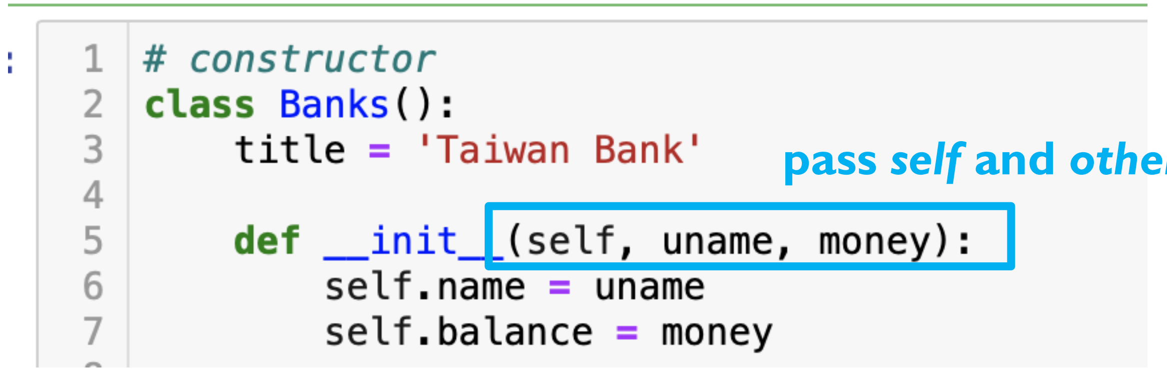
```
# opetate the class
class Banks():
    title = 'Taiwan Bank'
    def motto(self):
        return 'We love Taiwan!'

banklst = Banks()
print('Currenct Bank: ', banklst.title)
print('Bank Motto: ', banklst.motto())
```

Define object 'banklst'

```
Currenct Bank:  Taiwan Bank
Bank Motto:  We love Taiwan!
```

## Class

- Operator overloading method
  - Use constructor "__init__" to initial values in the class.
  - Syntax Example

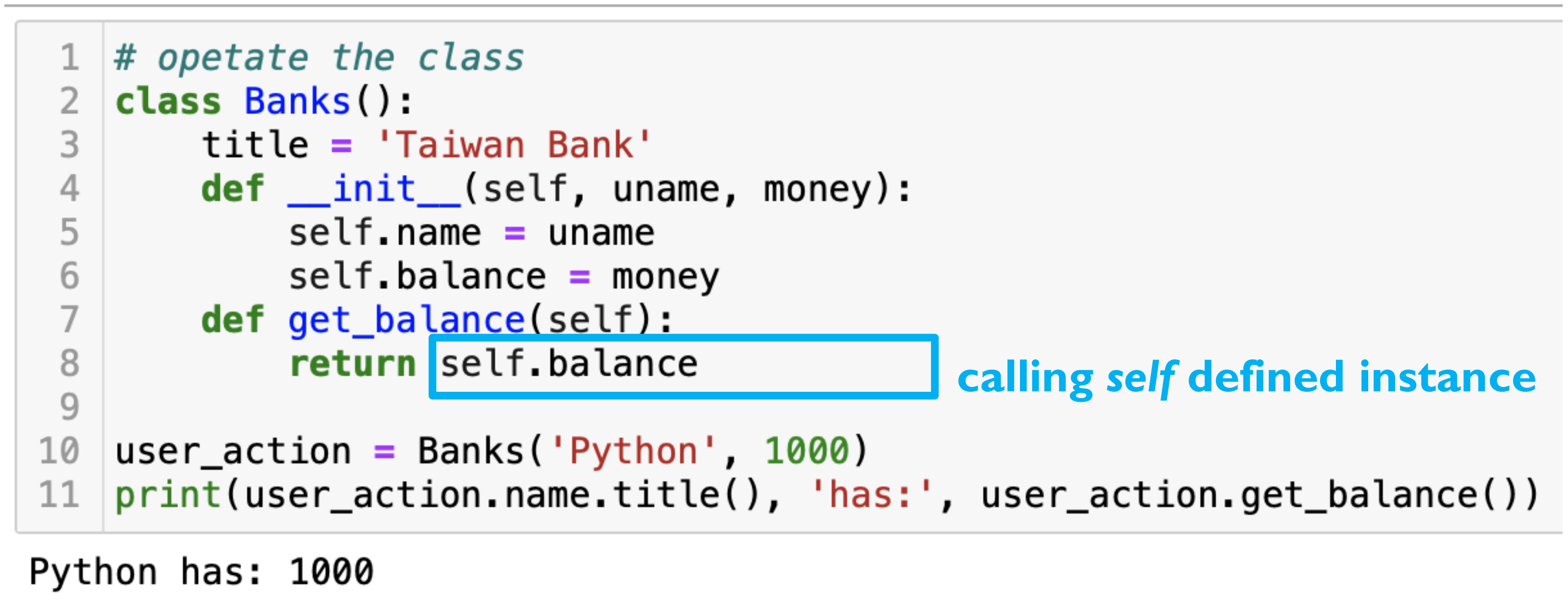


```python
# constructor
class Banks():
    title = 'Taiwan Bank'

    def __init__(self, uname, money):
        self.name = uname
        self.balance = money
```

**pass *self* and *other parameters***

# Class

- Example



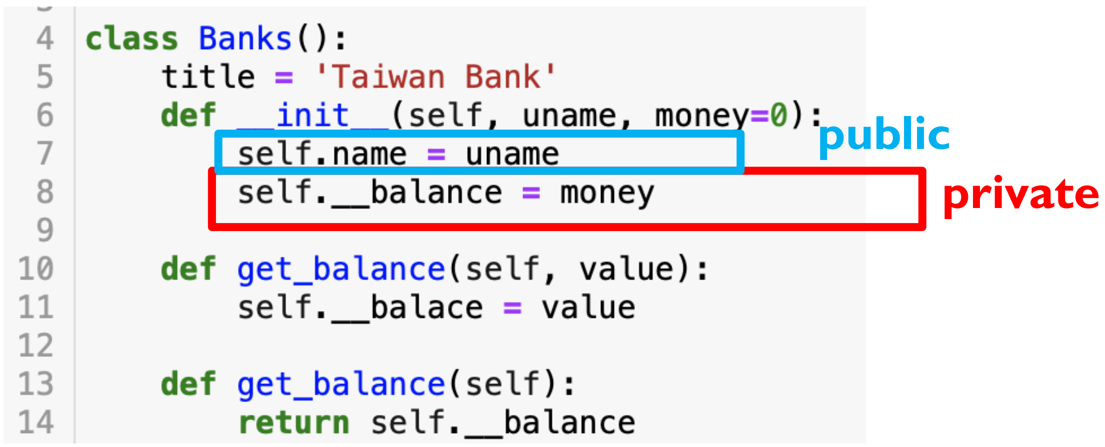
```python
# opetate the class
class Banks():
    title = 'Taiwan Bank'
    def __init__(self, uname, money):
        self.name = uname
        self.balance = money
    def get_balance(self):
        return self.balance

user_action = Banks('Python', 1000)
print(user_action.name.title(), 'has:', user_action.get_balance())
```

```
Python has: 1000
```

calling *self* defined instance

# Class

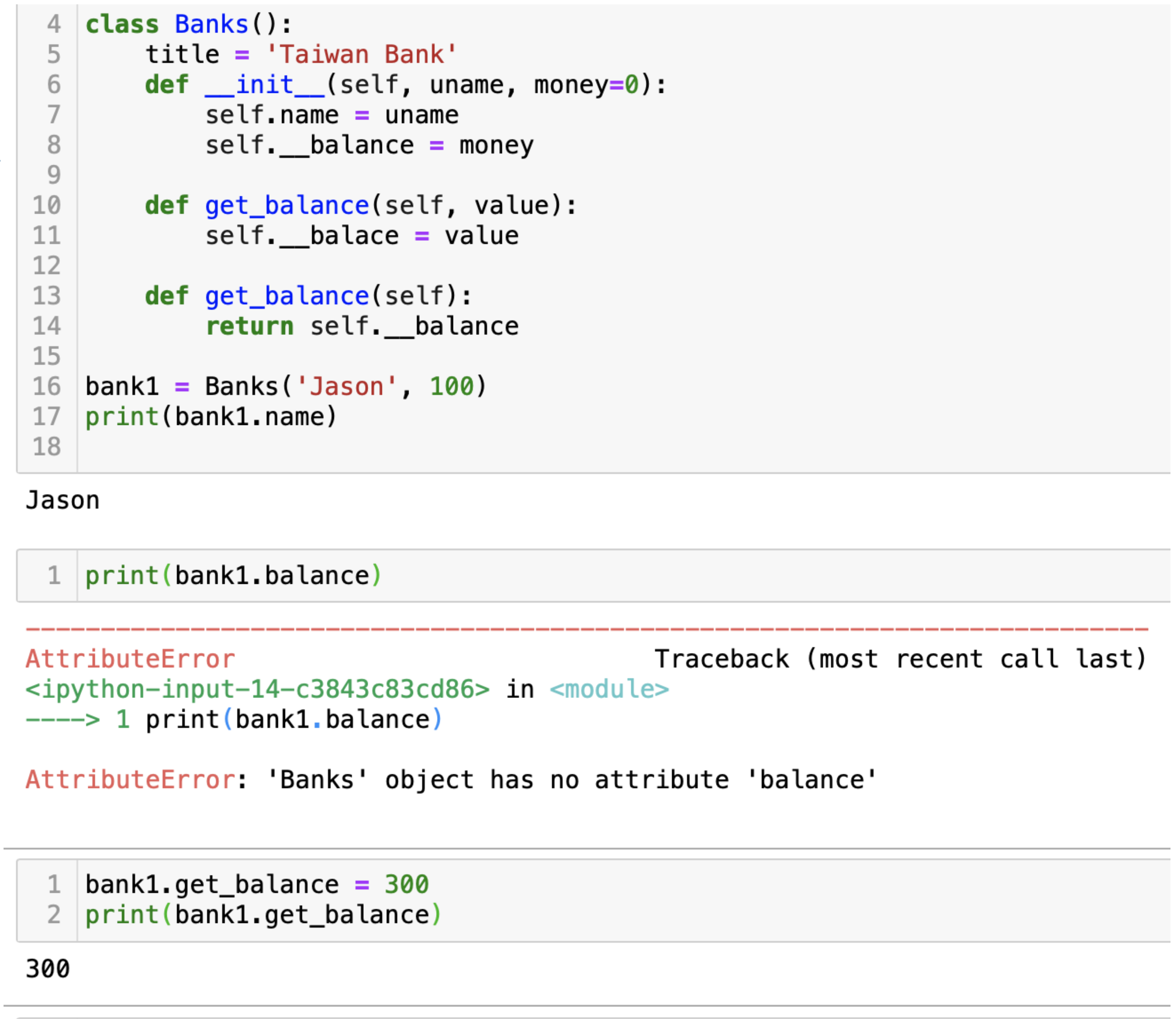
- Encapsulation
  - Avoid attributes inside classes being changed.
  - Make attribute private by using "__"(double underline).
  - Example:
    - Build a class with public and private attributes.



```python
class Banks():
    title = 'Taiwan Bank'
    def __init__(self, uname, money=0):
        self.name = uname
        self.__balance = money

    def get_balance(self, value):
        self.__balace = value

    def get_balance(self):
        return self.__balance
```

# Class

- Example



```python
class Banks():
    title = 'Taiwan Bank'
    def __init__(self, uname, money=0):
        self.name = uname
        self.__balance = money

    def get_balance(self, value):
        self.__balace = value

    def get_balance(self):
        return self.__balance

bank1 = Banks('Jason', 100)
print(bank1.name)
```

```
Jason
```

```python
print(bank1.balance)
```

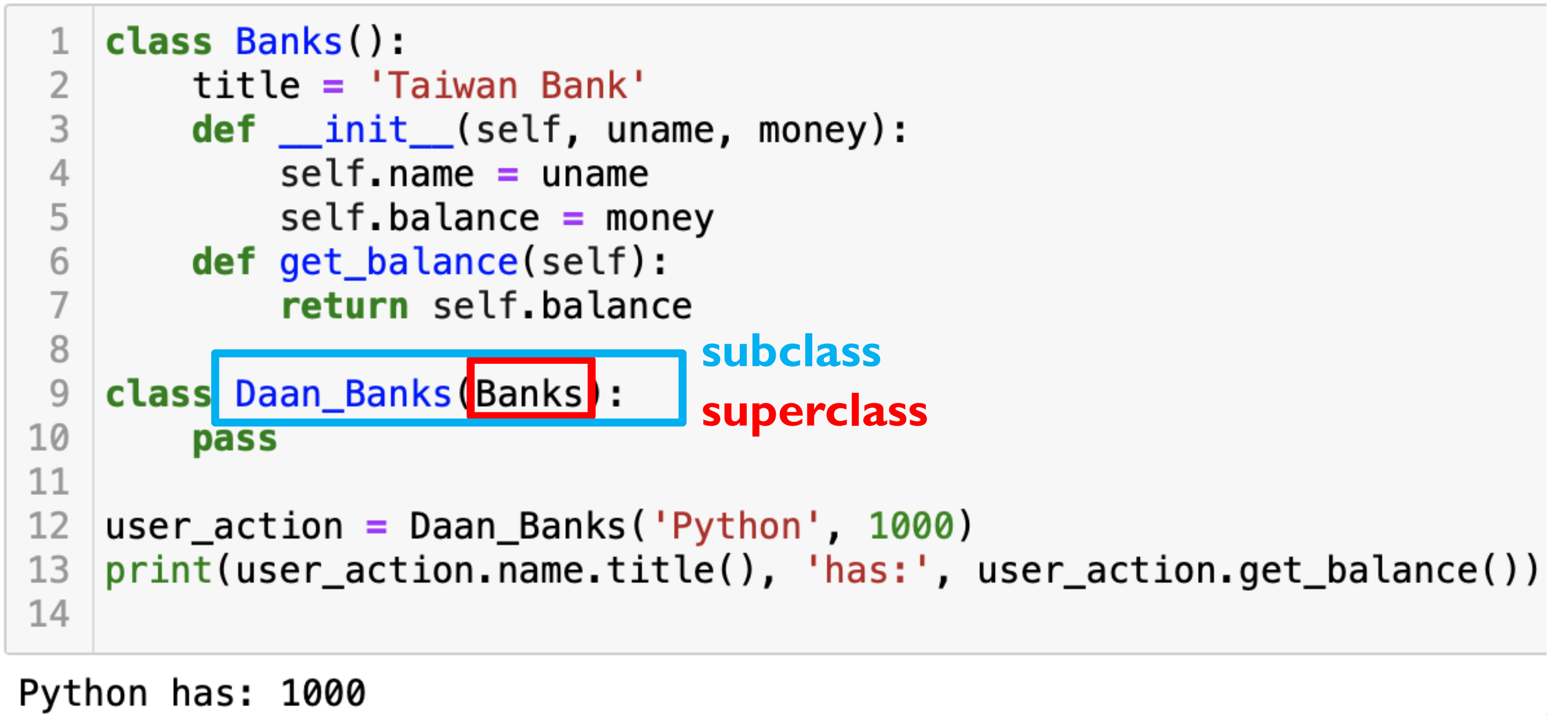
```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-14-c3843c83cd86> in <module>
----> 1 print(bank1.balance)

AttributeError: 'Banks' object has no attribute 'balance'
```

```python
bank1.get_balance = 300
print(bank1.get_balance)
```

```
300
```

## Class

- Inheritance
  - Attribute can be inherited by current class, the new class called subclass, the inherited class called superclass.
  - Example:

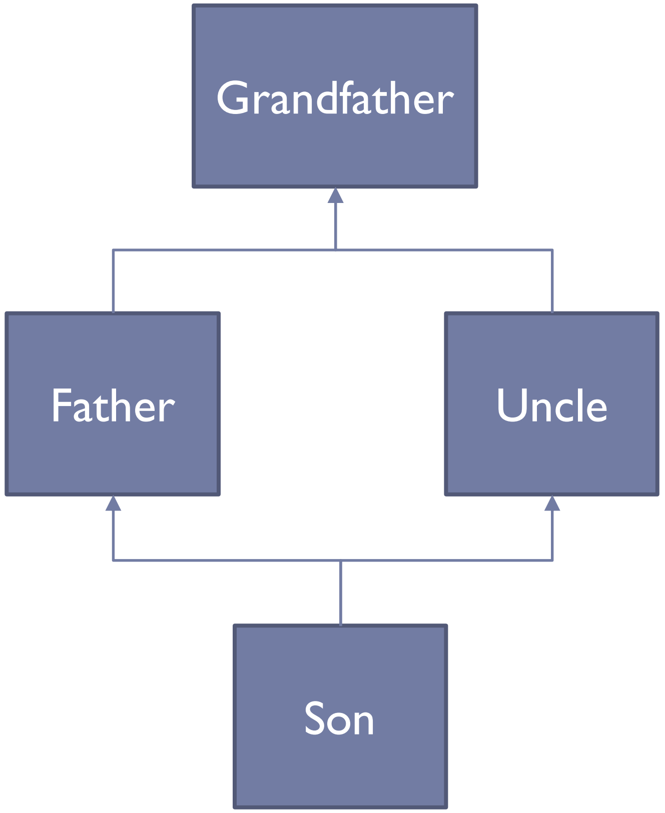


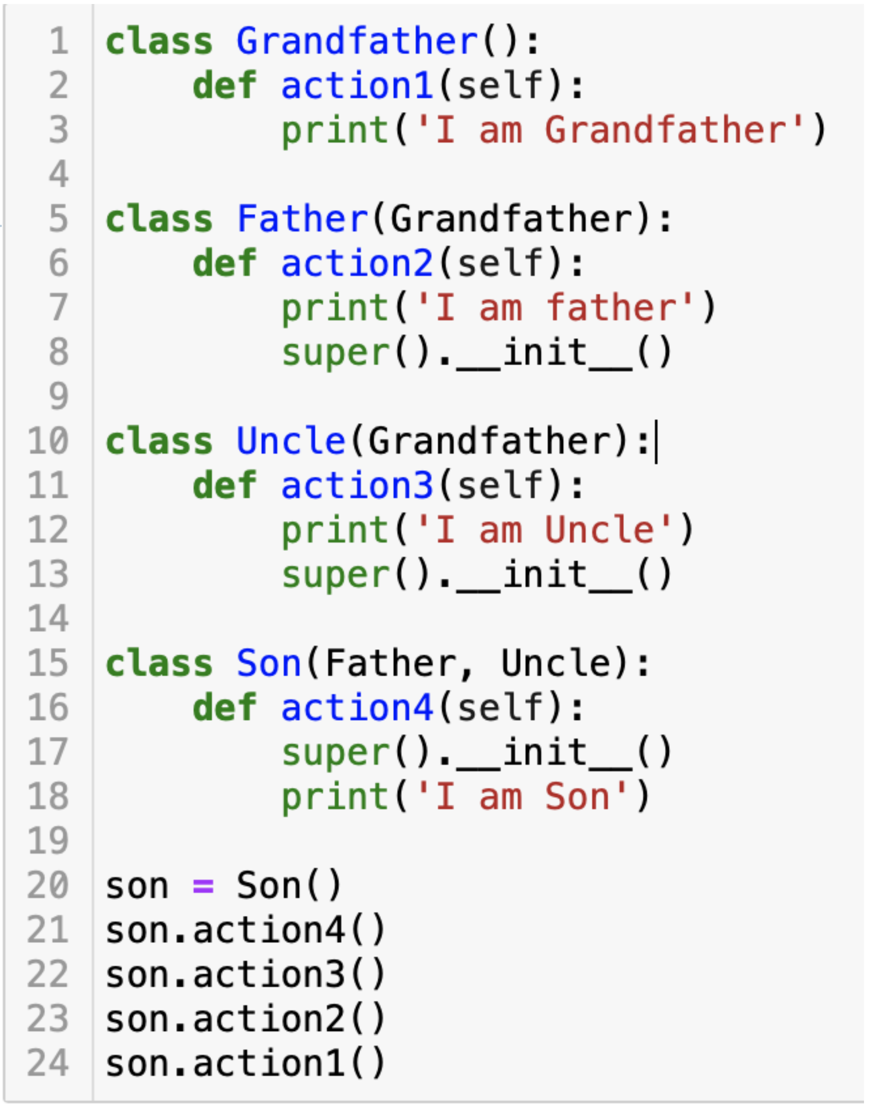
```python
class Banks():
    title = 'Taiwan Bank'
    def __init__(self, uname, money):
        self.name = uname
        self.balance = money
    def get_balance(self):
        return self.balance

class Daan_Banks(Banks):
    pass

user_action = Daan_Banks('Python', 1000)
print(user_action.name.title(), 'has:', user_action.get_balance())

```

```
Python has: 1000
```

## Class

- Multiple-inheritance



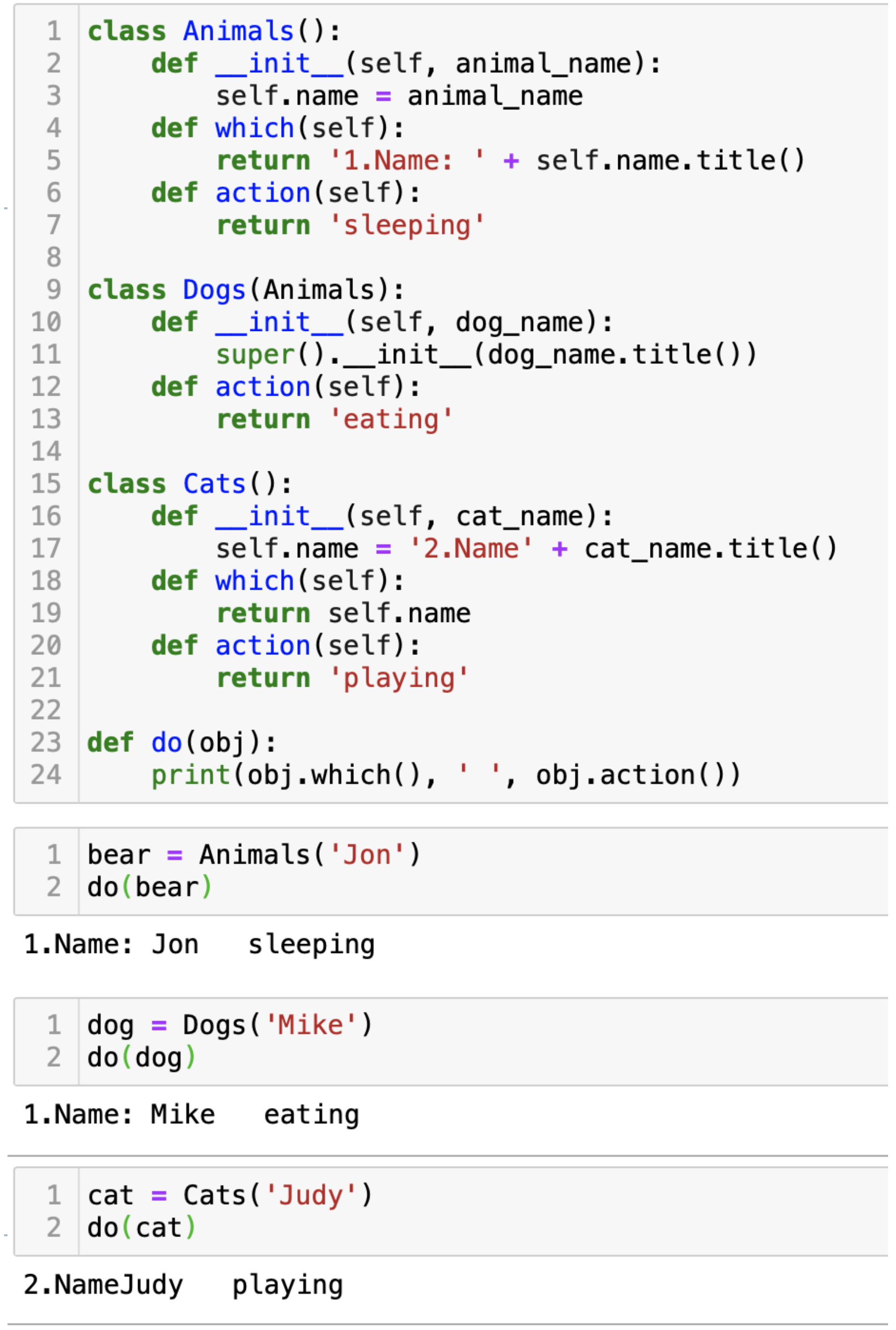


```
class Grandfather():
    def action1(self):
        print('I am Grandfather')

class Father(Grandfather):
    def action2(self):
        print('I am father')
        super().__init__()

class Uncle(Grandfather):
    def action3(self):
        print('I am Uncle')
        super().__init__()

class Son(Father, Uncle):
    def action4(self):
        super().__init__()
        print('I am Son')

son = Son()
son.action4()
son.action3()
son.action2()
son.action1()
```

```
I am Son
I am Uncle
I am father
I am Grandfather
```

# Class

- Polymorphism
  - Allow child class to define methods with the same name in parent class.



```python
class Animals():
    def __init__(self, animal_name):
        self.name = animal_name
    def which(self):
        return '1.Name: ' + self.name.title()
    def action(self):
        return 'sleeping'

class Dogs(Animals):
    def __init__(self, dog_name):
        super().__init__(dog_name.title())
    def action(self):
        return 'eating'

class Cats():
    def __init__(self, cat_name):
        self.name = '2.Name' + cat_name.title()
    def which(self):
        return self.name
    def action(self):
        return 'playing'

def do(obj):
    print(obj.which(), ' ', obj.action())
```

```python
bear = Animals('Jon')
do(bear)
```

```
1.Name: Jon   sleeping
```

```python
dog = Dogs('Mike')
do(dog)
```

```
1.Name: Mike   eating
```

```python
cat = Cats('Judy')
do(cat)
```

```
2.NameJudy   playing
```