Introduction to Programming in Python

Classes

Objectives

- To be able to read and write Python classes.
- To understand the concept of encapsulation and how it contributes to building modular and maintainable programs.
- To be able to write programs involving simple class definitions.

Class definition

- Class definitions have the form

```
class <class-name>:
      def __init__(self):
          <initialization>
      def <method1>:
           return <variant>
```

类的定义

```
class Person:
    def __init__(self, name, num):
        self.name = name
        self.num = str(num)
    def showme(self):
        print('I am %s!' % self.name)
    def info(self):
        return self.name + self.num
```

An Example of Class

```
class Person:
    def __init__(self, name, num):
        self.name = name
        self.num = str(num)
    def showme(self):
        print('I am %s!' % self.name)
    def info(self):
        return self.name + self.num
```

定义类的关键字class

def function():

An Example of Class

```
class Person:
    def __init__(self, name, num):
        self.name = name
        self.num = str(num)
```

_ _init_ _

类的初始化方法/构造函数

An Example of Class

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

self自参数

self代表类的实例：

class人类→self指某人、张三或李四

An Example of Class

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

self点操作符——访问属性

Attributes 属性——类的变量

An Example of Class

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

张三自己的学号2020

学号2020

```
class person:
    def __init__(self, name, num):
        self.name = name
        self.num = str(num)
pp = person('Peppa', 20202020)
print('I am ' + pp.name)
print('My number is '+ pp.num)
```

pp是person类的对象（实例）

由构造pp对象的过程：实例化

An Example of Class

```
bg = person('Peppa', 2019999)
print('I am ' + bg.name)
print('My number is '+ bg.num)
```

```
I am Peppa
My number is 2019999
```

对象bg点操作符——访问其属性

Attributes 属性——对象的变量

An Example of Class

Method方法→类内函数

```
class person:
    def __init__(self, name, num):
        self.name = name
        self.num = str(num)
    def showme(self):
        print('I am %s!' % self.name)
    def info(self):
        return self.name + self.num
```

Example of Class

```
stu1 = person('Peppa', 201999999)
stu2 = person('George', 20220000)
stu1.showme()
stu2.showme()
```

Method方法→类的函数

对象stu1/stu2点操作符→调用方法

Example of Class

```
stu1 = person('Peppa', 201999999)
stu2 = person('George', 20220000)
stu1.showme()
stu2.showme()
```

```
I am Peppa!
I am George!
```

```
class person:
    def __init__(self, name, num):
        self.name = name
    def showme(self):
        print('I am %s!' % self.name)
```

```
stu1 = person('Peppa', 201999999)
stu1.showme()
```

Example: Multi-Sided Dice

```
#类的方法
def setValue(self, value):
    self.value = value
```

区分函数与类的方法?

```
#函数
def setValue(value):
    new_value = value
     return new_value
```

new_value在函数外面能访问吗?

Example: Multi-Sided Dice

- 实例变量可以记住对象self的状态，并且这些信息可以作为对象的一部分在程序中传递。
- 这与局部函数变量不同，局部函数变量的值在函数终止时消失。

Questions?

華南師範大學
SOUTH CHINA NORMAL UNIVERSITY

UNIVERSITY OF ABERDEEN

Private Properties of Class

- **_ _private_attrs**: begin with two underlines stating that the property is private and cannot be accessed directly outside the class. Use **self.__private_attrs** in methods within a class.
  - `_ _weights,`
  - `_ _password,`
  - `_ _healthy,`
  - `...`

类的私有属性

Private Methods of Class

- Private Methods, e. g.,
  - validating a password,
  - internal processing,
  - Personal willingness,
  - ...

类的私有方法

Private Methods of Class

- **__private_method**:

Begin with two underlines stating that the method is private and cannot be accessed directly outside the class.

Use **self.__private_method** in methods within a class.

类的私有方法：仅供类的内部调用

不使用私有属性：泄露、攻击

```
class person:
    def __init__(self, name, age, weight):
        self.name = name
        self.age = age
        self.weight = weight

andy = person(‘Andy’, 18, 100)
andy.age = 81
andy.weight  = 200
```

私有属性如何访问?

```
class person:
    def __init__(self, name, age):
        self.name = name
        self.__age = age
andy = person('Andy', 18)
print(andy.__age)
```

```
AttributeError: 'person' object
has no attribute '__age'
```

私有属性如何访问？getter

```
class person:
    def __init__(self, name, age, weight):
        self.name = name
        self.__age = age

  def get_age(self):
        return self.__age
andy = person(‘Andy’, 18)
print( andy.get_age() )
```

私有属性如何访问？getter

```python
class person:
    def __init__(self, name, age):
        self.name = name
        self.__age = age

    def get_age(self):
        return self.__age

andy = person(‘Andy’, 18)
print( andy.get_age() )
```

```python
class person:
    def __init__(self, name, age):
        self.name = name
        self.__age = age

    @property
    def get_age(self):
        return self.__age
andy = person(‘Andy’, 18)
print( andy.get_age )
```

私有属性如何访问？setter

```
class person:
.........
    @property
    def get_age(self):
        return self.__age

    @get_age.setter
    def set_age(self, age2):
        self.__age = age2
```

```
andy = person('Andy',

print(f'{andy.name}\'s
age is {andy.get_age}
#getter

andy.set_age = 81 #se

print(f'{andy.name}\'s
age is {andy.get_age}
```

Inheritance of Class 类的继承

- **class** DerivedClassName(BaseClassName):

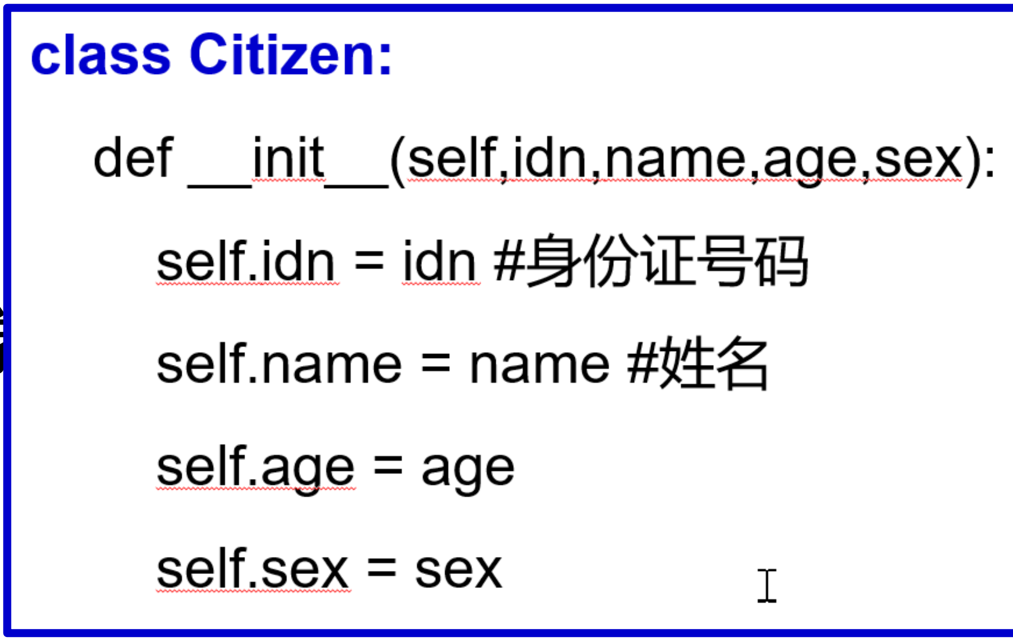
      <Class body>

- Python can create new classes based on one or more classes (**parent**) that can use some of the properties and methods. This process called **inheritance**. Use the **super()** method to call the constructor of the superclass.

Example: Citizen and person

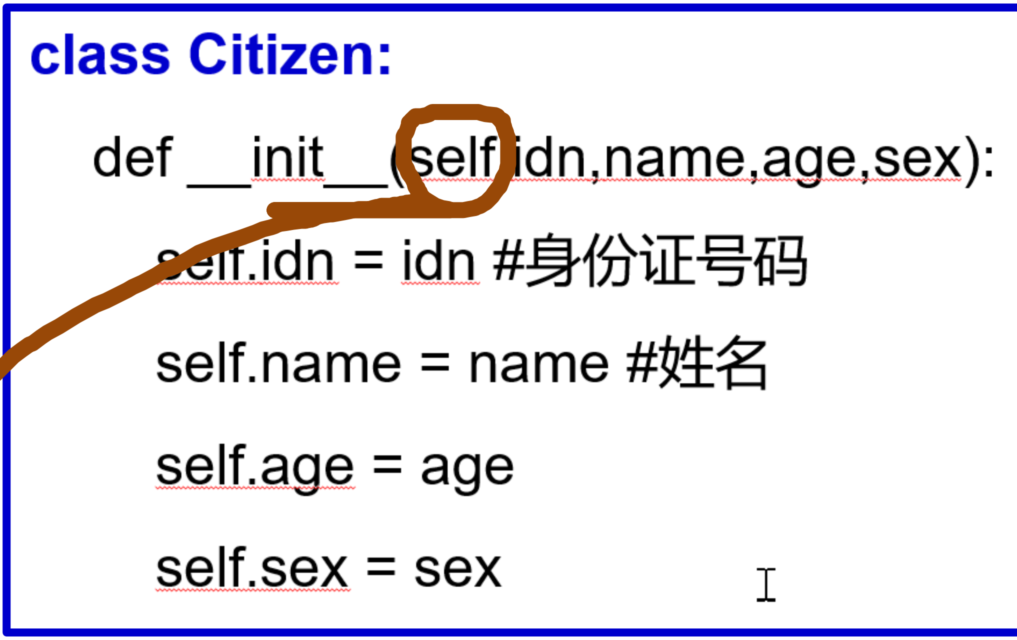
```
class Citizen:
    def __init__(self,idn,name,age,sex):
        self.idn = idn #身份证号码
        self.name = name #姓名
        self.age = age
        self.sex = sex
```

```
class Student:
    def __init__(self,idn,name,age,sex,stdno,grade,score):
        self.idn = idn #身份证号码
        self.name = name #姓名
        self.age = age
        self.sex = sex
        self.stdno = stdno #学
        self.grade = grade
       self.score = score
```



```
class Citizen:
    def __init__(self,idn,name,age,sex):
        self.idn = idn #身份证号码
        self.name = name #姓名
        self.age = age
        self.sex = sex
```

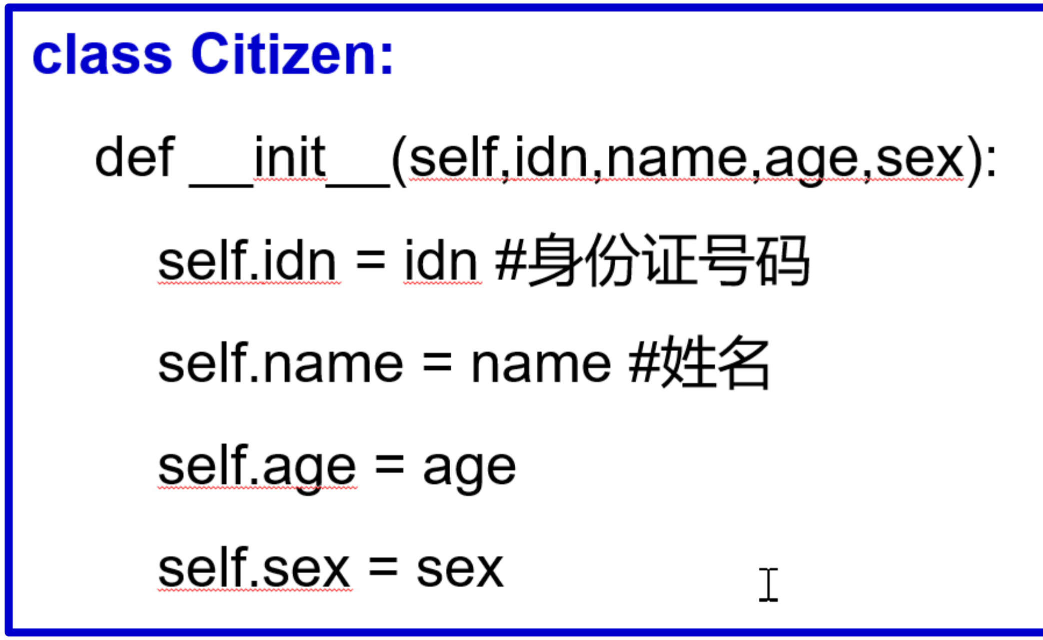
```
class Student():
    def __init__(self,stdno,grade,score):
        self.stdno = stdno #学生证号
        self.grade = grade
        self.score = score
```



```
class Citizen:
    def __init__(self,idn,name,age,sex):
        self.idn = idn #身份证号码
        self.name = name #姓名
        self.age = age
        self.sex = sex
```

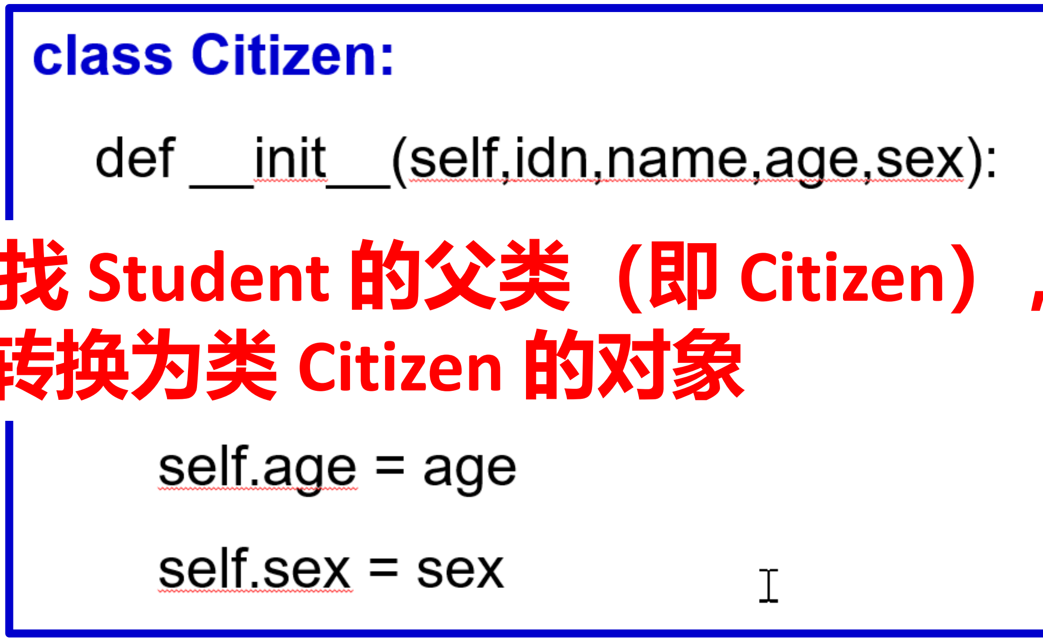
```
# idn,name,age,sex?
stu_object = Student()
Citizen.__init__(stu_object, idn,name,age,sex)
```

```
class Student(Citizen):
    def __init__(self,idn,name,age,sex,stdno,grade,score):
        Citizen.__init__(self, idn,name,age,sex)
        self.stdno = stdno #学生证号
        self.grade = grade
        self.score = score
```



```
class Citizen:
    def __init__(self,idn,name,age,sex):
        self.idn = idn #身份证号码
        self.name = name #姓名
        self.age = age
        self.sex = sex
```

```
class Student(Citizen):
    def __init__(self,idn,name,age,sex,stdno,grade,score):
        super(Student, self).__init__(idn,name,age,sex)
        self.stdno = stdno #学生证号
        self.grade = grade
        self.score = score
```



```
class Citizen:
    def __init__(self,idn,name,age,sex):
```

```
        self.age = age
        self.sex = sex
```

# super(Student,self) 首先找 Student 的父类（即 Citizen），然后把类 Student 的对象转换为类 Citizen 的对象

```
class Student(Citizen):
    def __init__(self,idn,name,age,sex,stdno,grade,score):
        super().__init__(idn,name,age,sex)
        self.stdno = stdno #学生证号
        self.grade = grade
        self.score = score
```

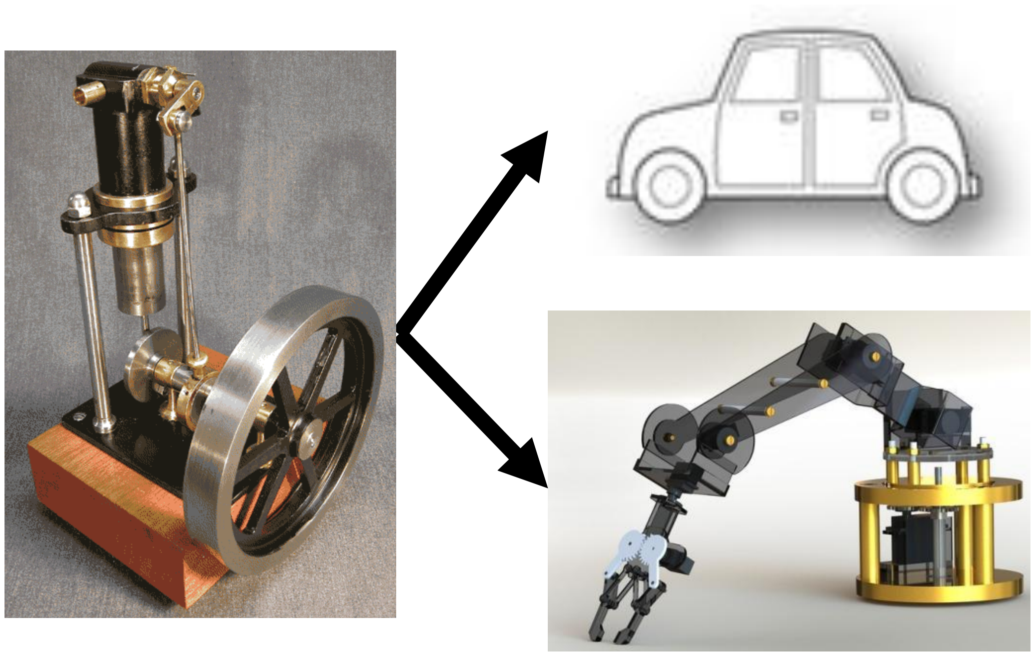
```
super(Student, self).__init__(idn,name,age,sex)
```

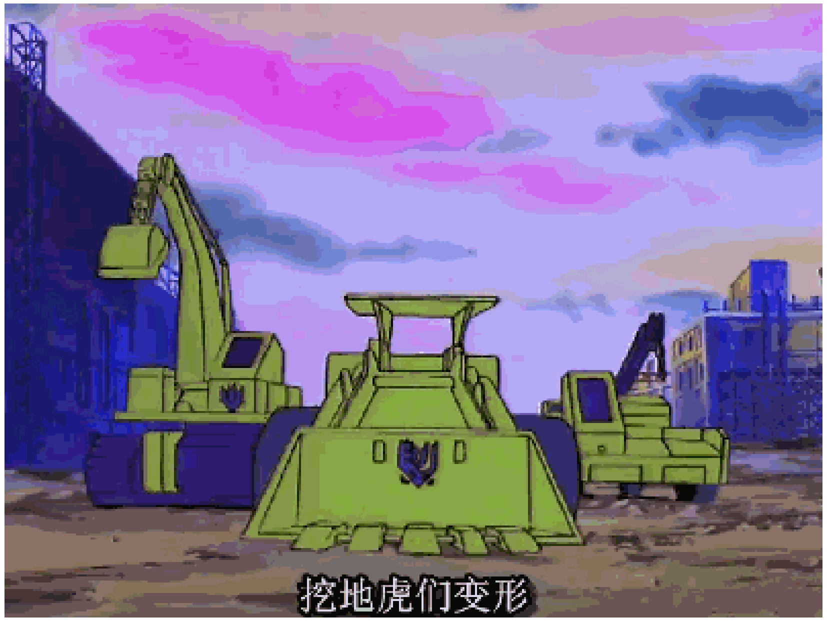
简化

Inheritance of Class

- Multiple inheritance

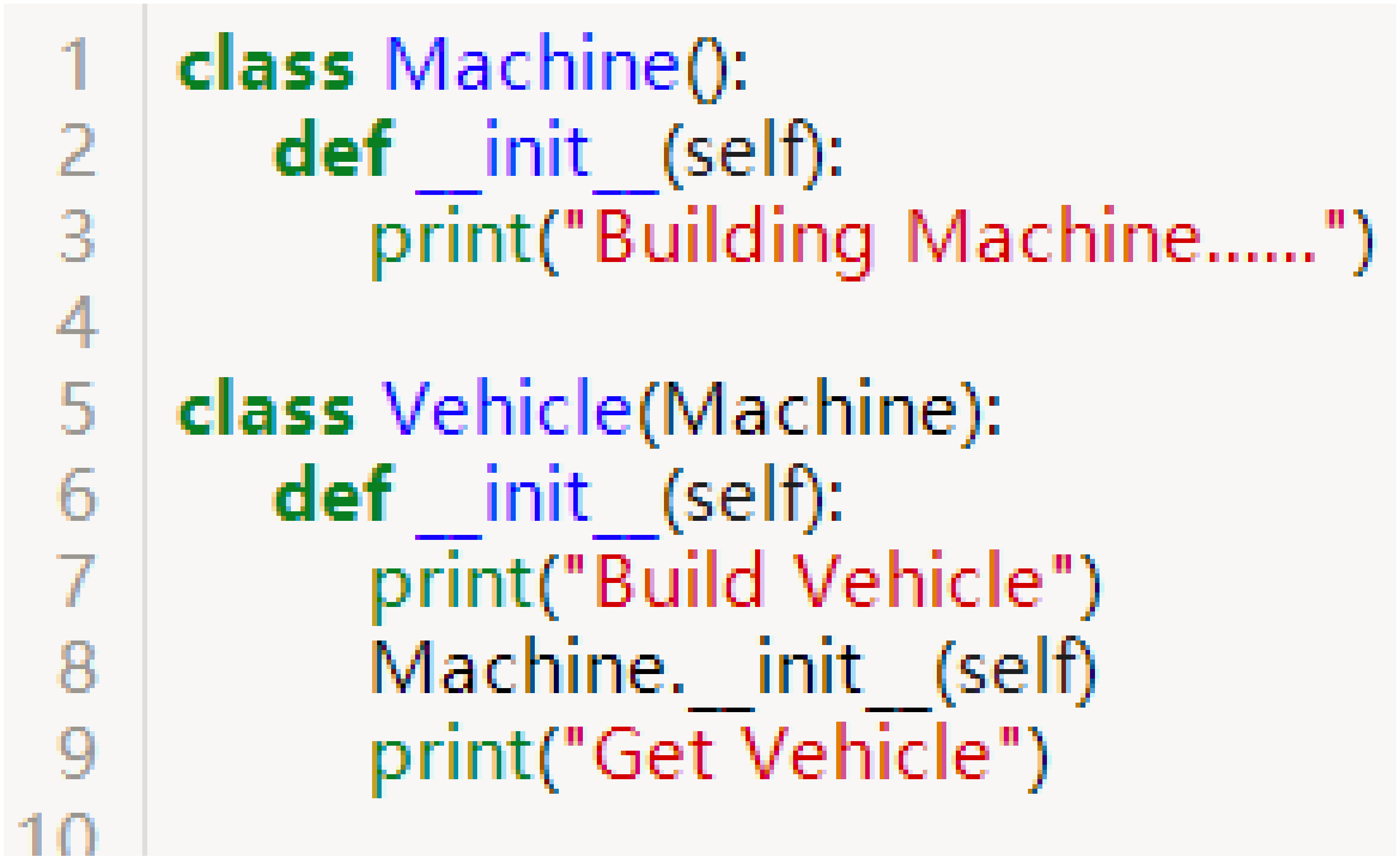
多重继承





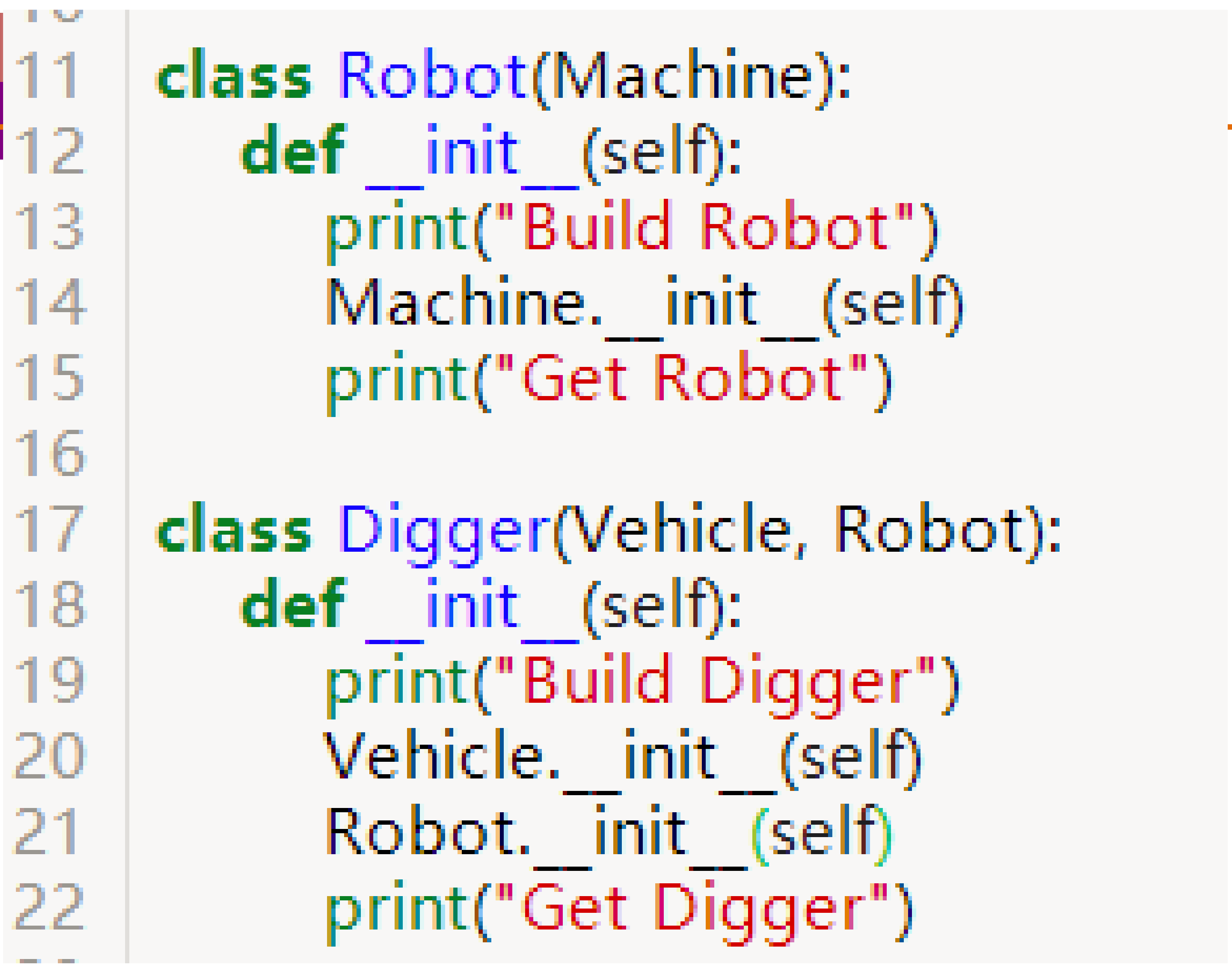
Inheritance of Class

- Multiple inheritance



```python
class Machine():
    def __init__(self):
        print("Building Machine......")

class Vehicle(Machine):
    def __init__(self):
        print("Build Vehicle")
        Machine.__init__(self)
        print("Get Vehicle")

```

■ Mu



```
class Robot(Machine):
    def __init__(self):
        print("Build Robot")
        Machine.__init__(self)
        print("Get Robot")

class Digger(Vehicle, Robot):
    def __init__(self):
        print("Build Digger")
        Vehicle.__init__(self)
        Robot.__init__(self)
        print("Get Digger")
```

```python
lx_dig = Digger()
print(Digger.__mro__)
```

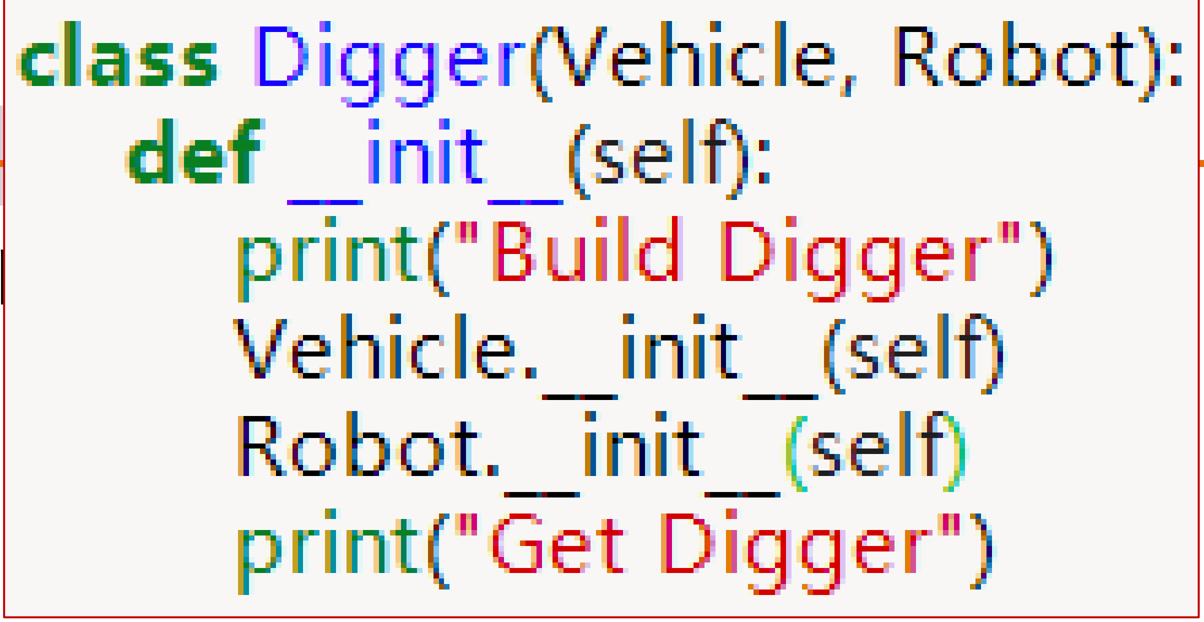
```
Build Digger
Build Vehicle
Building Mach
Get Vehicle
Build Robot
Building Machine......
Get Robot
Get Digger
```

```python
class Digger(Vehicle, Robot):
    def __init__(self):
        print("Build Digger")
        Vehicle.__init__(self)
        Robot.__init__(self)
        print("Get Digger")
```

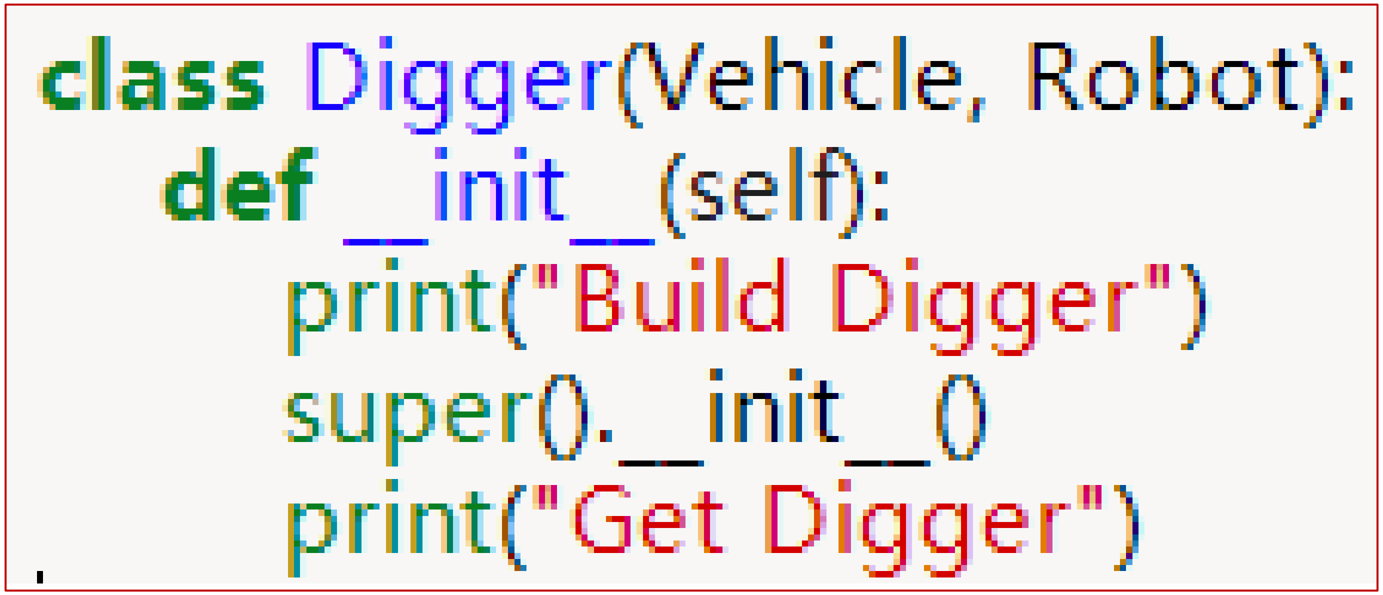
华南师范大学 SOUTH CHINA NORMAL UNIVERSITY

UNIVERSITY OF ABERDEEN

- Multi



```python
class Digger(Vehicle, Robot):
    def __init__(self):
        print("Build Digger")
        Vehicle.__init__(self)
        Robot.__init__(self)
        print("Get Digger")
```



```python
class Digger(Vehicle, Robot):
    def __init__(self):
        print("Build Digger")
        super().__init__()
        print("Get Digger")
```

面向对象编程中常见概念深入解析

继承：继承自拖拉机，实现了扫地的接口。

封装：无需知道如何运作，开动即可。

多态：平时扫地，天热当风扇。

重用：没有额外动力，充分利用了发动机能量。

多线程：多个扫把同时工作。

低耦合：扫把可以换成拖把而无需改动。

组件编程：每个配件都是可单独利用的工具。

适配器模式：无需造发动机，继承自拖拉机

只取动力方法。

代码托管：无需管理垃圾，直接扫到路边即可。

Polymorphism of Class

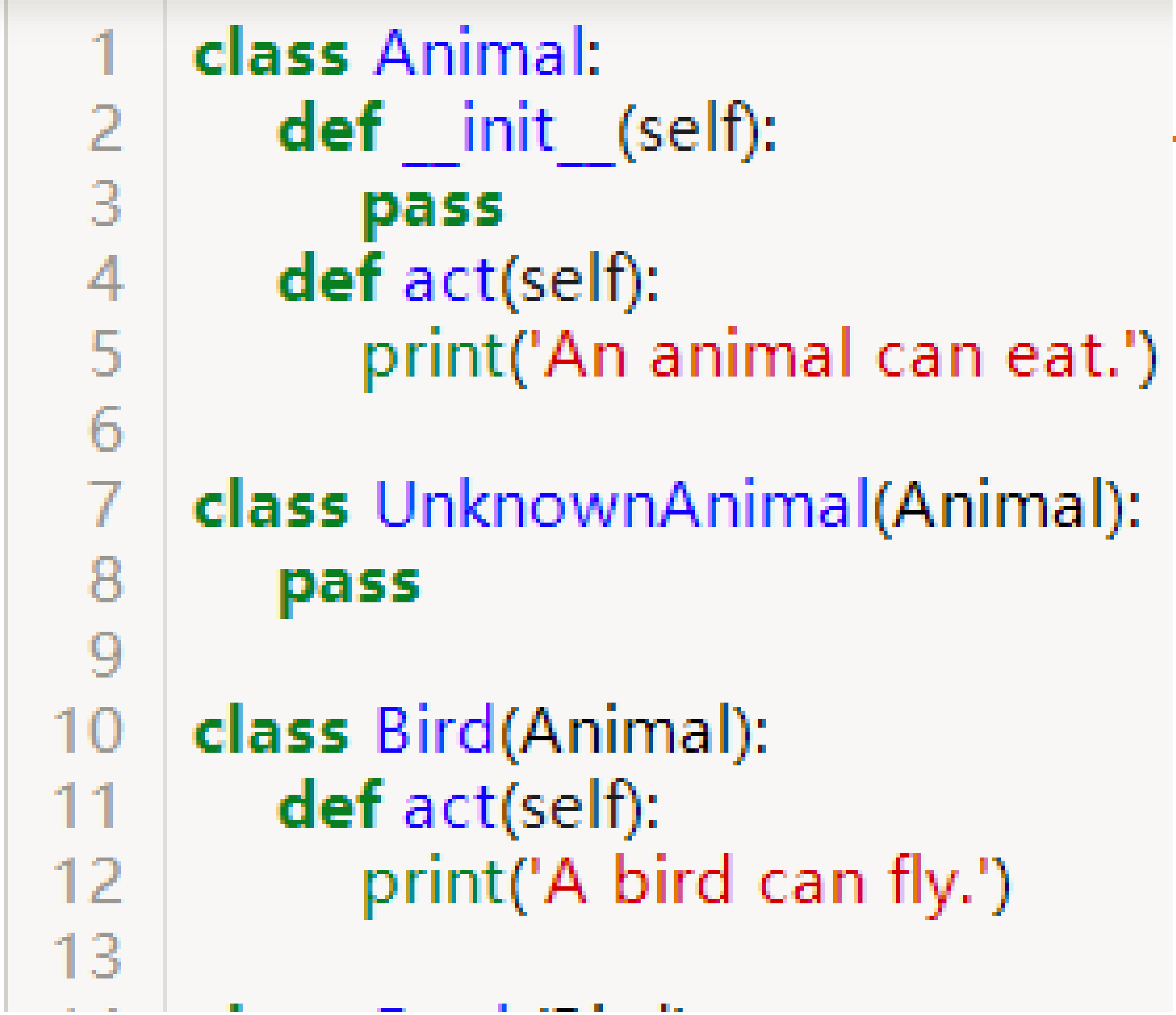
- A concept of using common operation in different ways for different data input.
- An important concept when you deal with child and parent class.
- Polymorphism is applied through **method overriding & operator overloading**.

类的多态性体现：方法重写、运算符重载

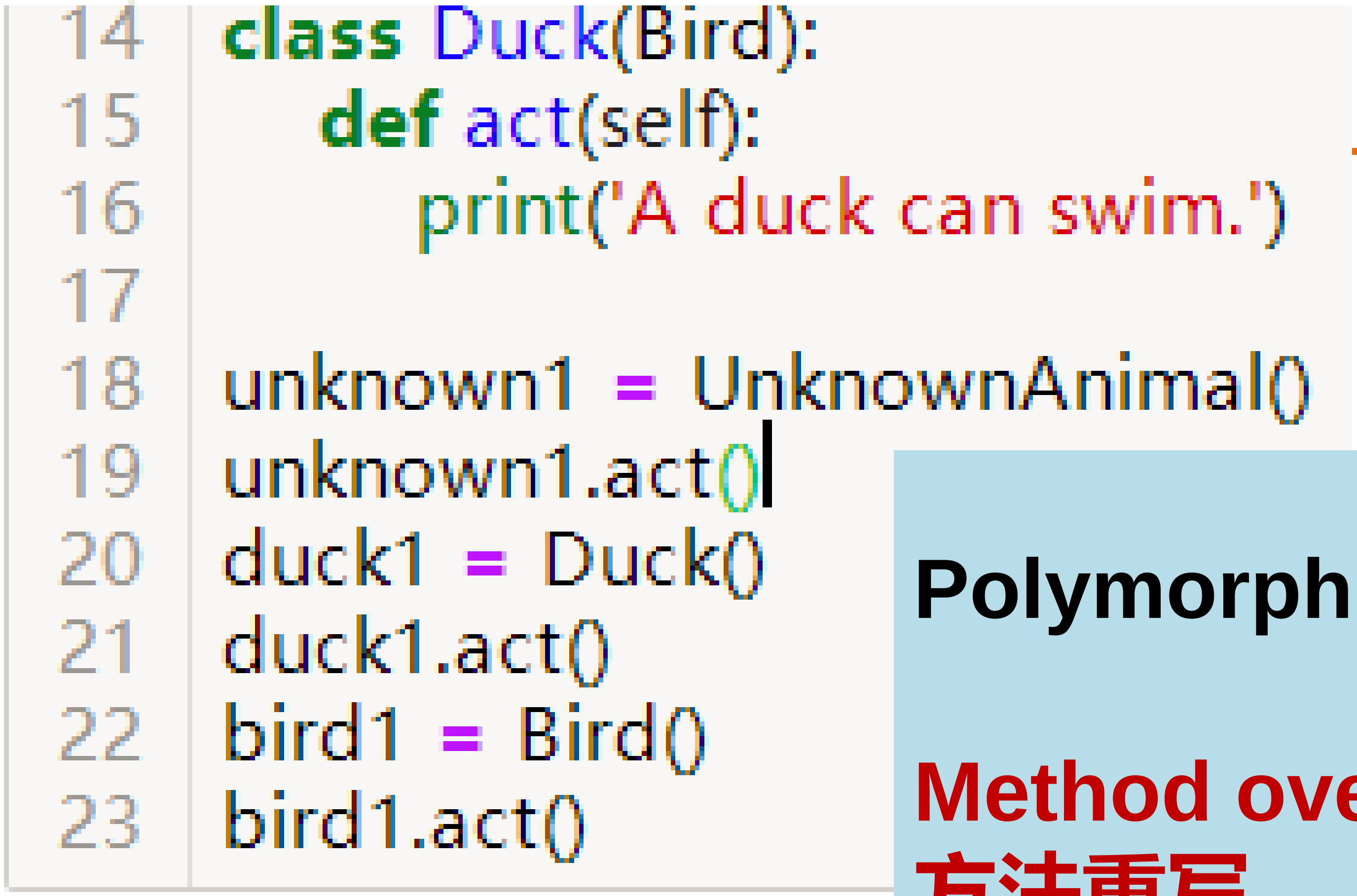
Polymorphism of Class:

Method Overriding

Method overriding allows us to have a method in the child class with the same name as in the parent class but the definition of the child class method is different from parent class method.



```python
class Animal:
    def __init__(self):
        pass
    def act(self):
        print('An animal can eat.')

class UnknownAnimal(Animal):
    pass

class Bird(Animal):
    def act(self):
        print('A bird can fly.')

```

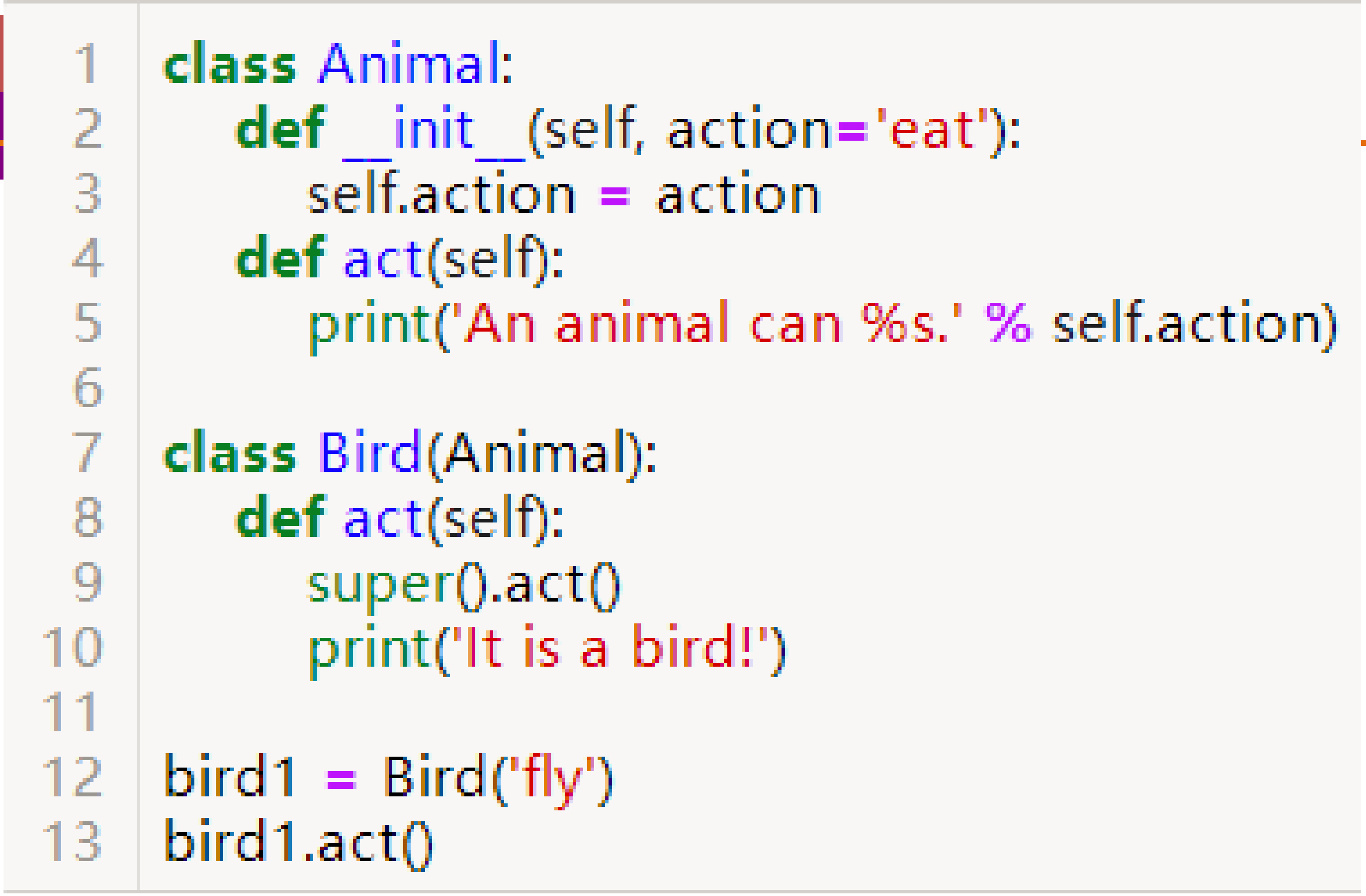


```python
class Duck(Bird):
    def act(self):
        print('A duck can swim.')

unknown1 = UnknownAnimal()
unknown1.act()
duck1 = Duck()
duck1.act()
bird1 = Bird()
bird1.act()
```

```
An animal can eat.
A duck can swim.
A bird can fly.
```

Polymorphism多态性

Method overriding 方法重写



```python
class Animal:
    def __init__(self, action='eat'):
        self.action = action
    def act(self):
        print('An animal can %s.' % self.action)

class Bird(Animal):
    def act(self):
        super().act()
        print('It is a bird!')

bird1 = Bird('fly')
bird1.act()
```

```
An animal can fly.
It is a bird!
```

Arbitrary Argument in Class

可变参数在类中的应用

```
def info(*args, **kwargs):
    for name in args:
        print('Name:', name)
    for name in kwargs:
        print(name, kwargs[name])
info('Andy')
info('Andy', 'Bob', 'Candy')
info(Andy=1, Bob=2, Candy=3)
```

Arbitrary Argument in Class

可变参数在类中的应用

```
class Car:
    def __init__(self, brand, color):
        self.brand = brand
        self.color = color

                car_obj1 = ECar1('Tesla', 'Red', '500')
class ECar1:
    def __init__(self, brand, color, power):
        self.brand = brand
        self.color = color
        self.power = power
```

Arbitrary Argument in Class

可变参数在类中的应用

```
class ECar2(Car):
    def __init__(self, brand, color, power):
        super().__init__(brand, color)
         #需要一个个传输参数
        self.power = power
```

```
car_obj2 = ECar2('Tesla', 'Red', 500)
```

可变参数在类中的应用

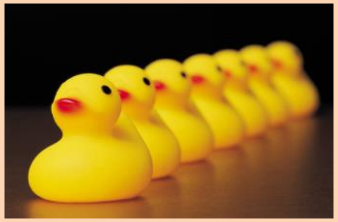
```
class Car:
    def __init__(self, brand, color, *args, **k):
        self.brand = brand
        self.color = color
class ECar3(Car):
    def __init__(self, power, *args, **kwargs):
        super().__init__(*args, **kwargs)
        #可变参数→传输参数
        self.power = power
```

```
car_obj3 = ECar3(‘Tesla’, ‘Red’, 500)
car_obj4 = ECar3(‘Tesla’, ‘Red’, 500, year=3)
```

Polymorphism of Class

- **Duck Typing** 多态性，鸭子类型
  - Form of polymorphism implemented by Python.
  - Duck typing allows us to use **<u>any object that provides the required behaviour</u>** without forcing it to be a subclass.

The term "duck typing" comes from an adage attributed to poet James Whitcomb Riley, stating that "when I see a bird that walks like a duck and swims like a duck and quacks like a duck, I call that bird a duck."



Polymorphism of Class:

- Python operators work for built-in classes. But same operator behaves differently with different types.
- **For example, the + operator will, perform arithmetic addition on two numbers, merge two lists and concatenate two strings.**
- This feature in Python, that allows same operator to have different meaning according to the context is called operator overloading.

Polymorphism of Class:

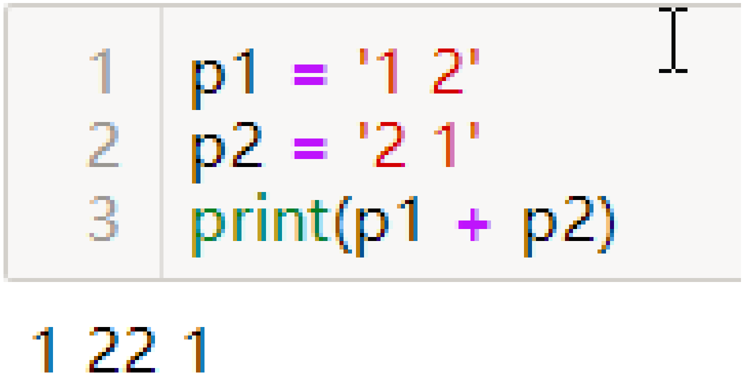
- 123 + 321?
- '321’ + '123' ?
- [[1,2]
  [2, 1]] + 1 ?

Polymorphism多态性

Operator overloading 运算符重载

Polymorphism of Class:

- Suppose you have created a Point class to represent two-dimensional points, what happens when you use the plus operator to add them? Most likely Pyth

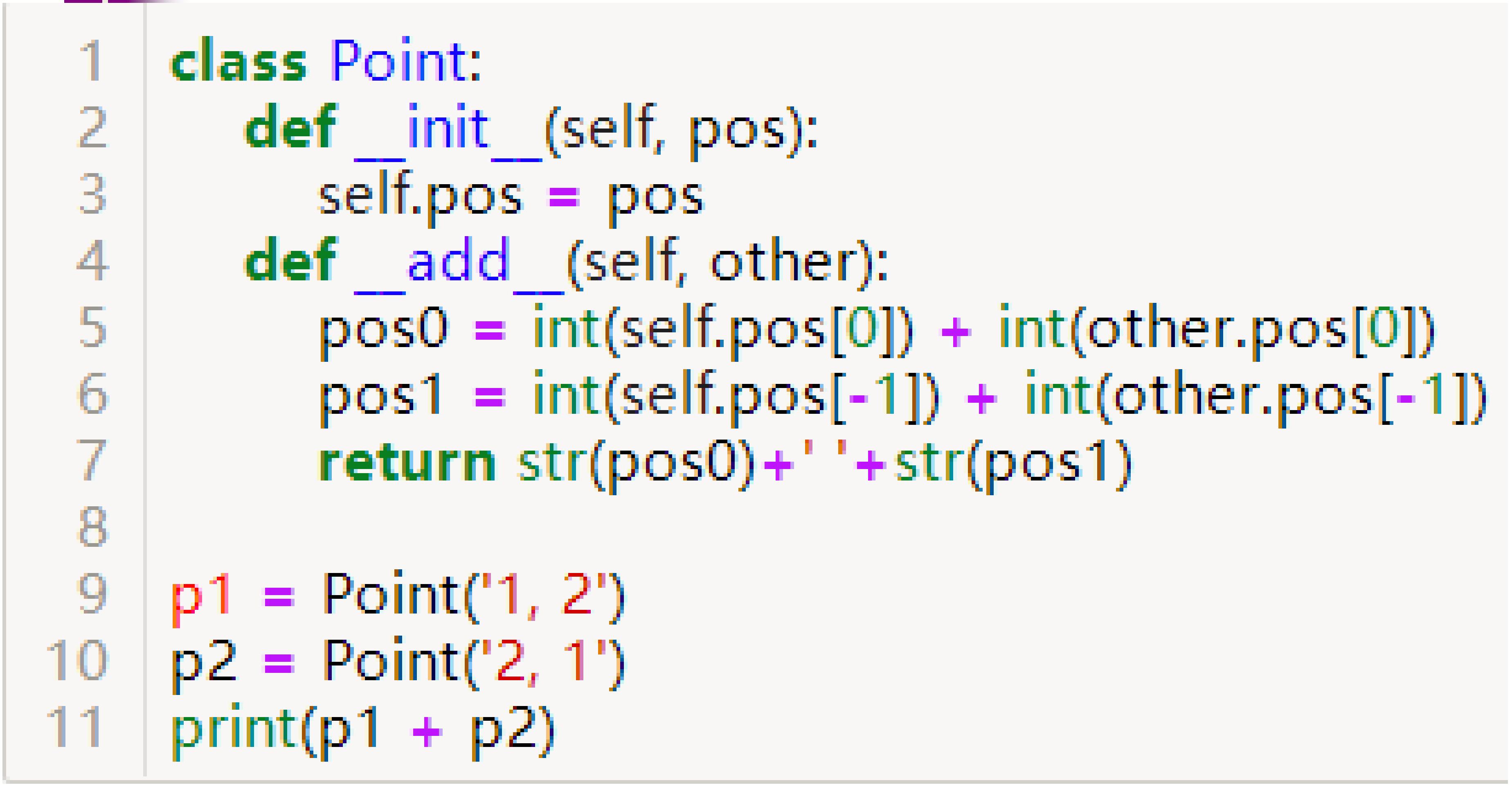


```
p1 = '1 2'
p2 = '2 1'
print(p1 + p2)
```

```
1 22 1
```

- You could, however, define the __*add*__ method in your class to perform vector addition and then the plus operator would behave as per expectation –

Polymorphism of Class:



```python
class Point:
    def __init__(self, pos):
        self.pos = pos
    def __add__(self, other):
        pos0 = int(self.pos[0]) + int(other.pos[0])
        pos1 = int(self.pos[-1]) + int(other.pos[-1])
        return str(pos0)+' '+str(pos1)

p1 = Point('1, 2')
p2 = Point('2, 1')
print(p1 + p2)
```

```
3 3
```

```python
import matplotlib.pyplot as plt
from PIL import Image
lenna = Image.open('./lenna.jpg')

class int:
    def __init__(self, num):
        self.num=num
        
    def __add__(self, other):
        
        out = self.num + other.num
        print('c=',out)
        plt.imshow(lenna)
        plt.show()
        return out
```

```python
a = int(3.1314)
b = int(2.1)
c = a+b
```

```
c = 5.3314
```

Polymorphism of Class:

- What actually happens is that, when you do p1 + p2, Python will call p1.__add__(p2) which in turn is Point.__add__(p1,p2).
- Similarly, we can overload other operators as well. The special function that we need to implement is tabulated below.

Polymorphism of Class:

Operator	Expression	Internally
Addition	p1 + p2	p1.__add__(p2)
Subtraction	p1 - p2	p1.__sub__(p2)
Multiplication	p1 * p2	p1.__mul__(p2)
Power	p1 ** p2	p1.__pow__(p2)
Division	p1 / p2	p1.__truediv__(p2)
Floor Division	p1 // p2	p1.__floordiv__(p2)
Remainder (modulo)	p1 % p2	p1.__mod__(p2)

Polymorphism of Class:

Operator	Expression	Internally
Less than	p1 < p2	p1.__lt__(p2)
Less than or equal to	p1 <= p2	p1.__le__(p2)
Equal to	p1 == p2	p1.__eq__(p2)
Not equal to	p1 != p2	p1.__ne__(p2)
Greater than	p1 > p2	p1.__gt__(p2)
Greater than or equal to	p1 >= p2	p1.__ge__(p2)
Less than	p1 < p2	p1.__lt__(p2)

Python Class

- **Encapsulation**: Hiding the private details of a class from other objects.
- **Inheritance**: A process of using details from a new class without modifying existing class.
- **Polymorphism**: A concept of using common operation in different ways for different data input.

Questions?

Modules and packages

- We have learn how to create classes and instantiate objects.
- For small programs, we can just put all our classes into one file.
- As the projects grow, it can become difficult to find the one class among many classes.

Modules and packages

- What is Modules and packages?

➢Modules and packages can help us to organize the programs.

```
import math
import random
```

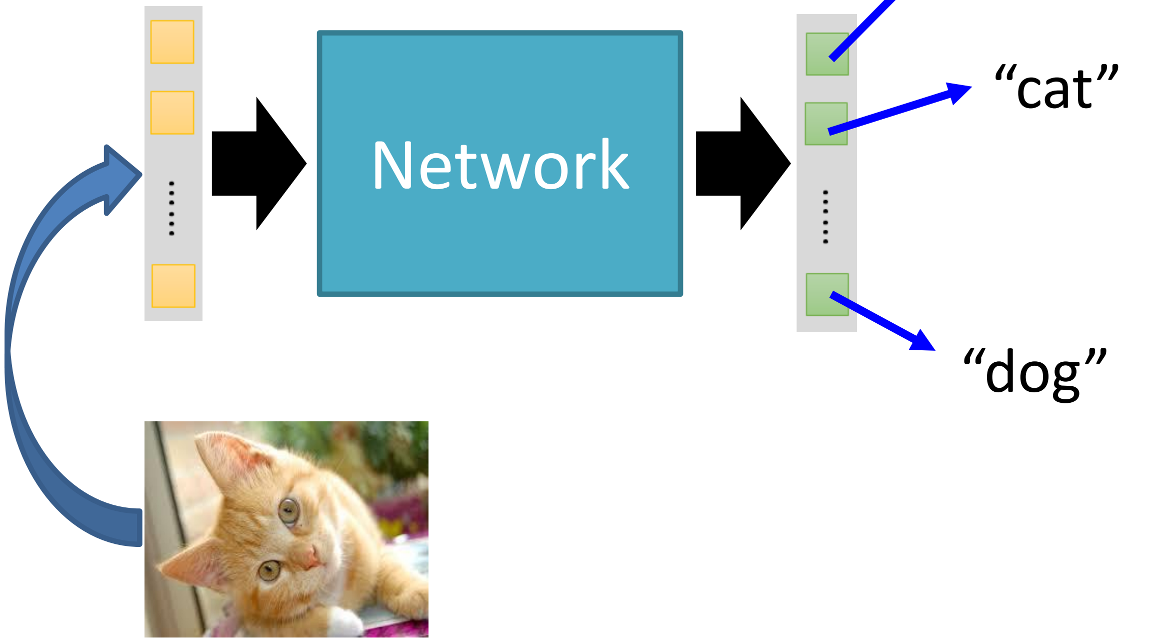
Modules and packages

```
from playsound import playsound
file = './music.mp3'
playsound(file)
```

Modules and packages

```
from PIL import Image

img = Image.open('./1.png')
img.show()
```

DATA
Ai
IoT
Artificial Intelligence

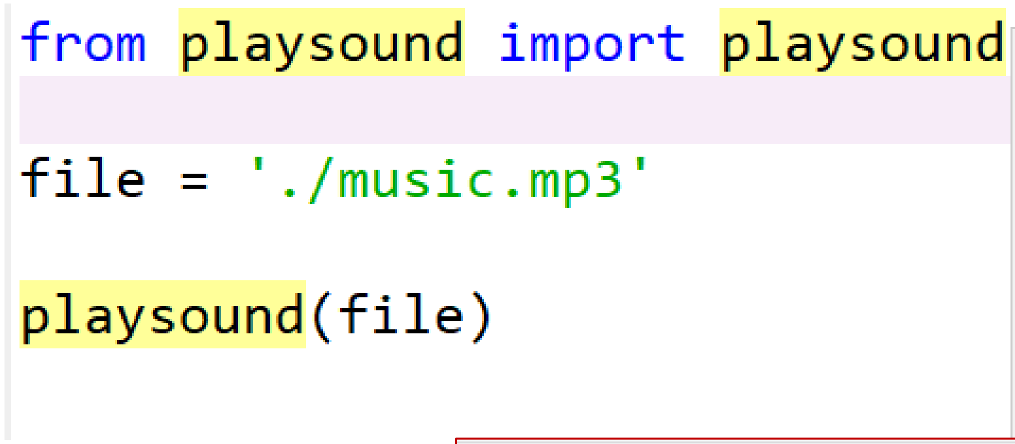

Network
"monkey"
"cat"
"dog"

"monkey"
"cat"
"dog"

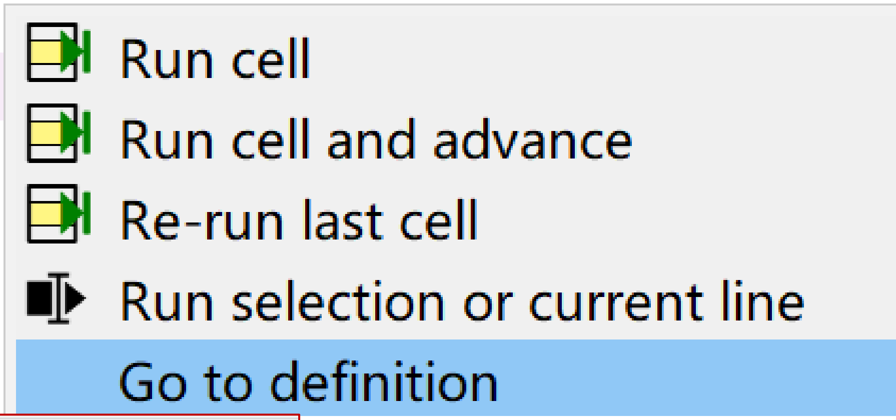
Modules and packages

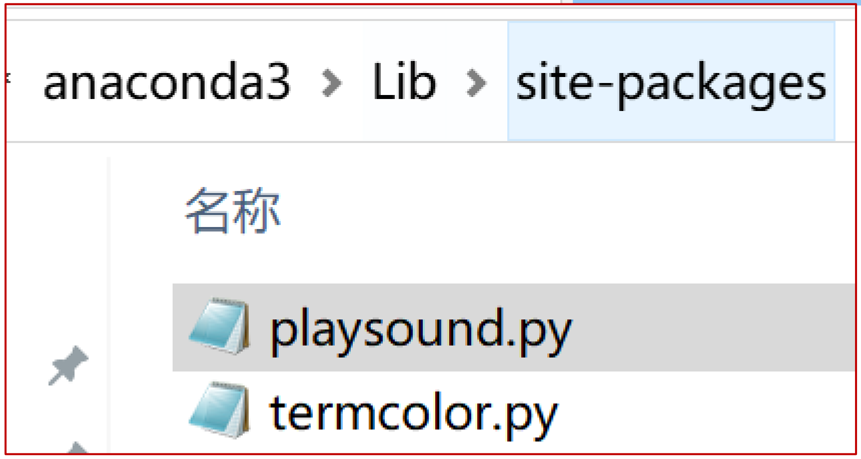
- What is Modules and packages?

They are simply Python files!



```
from playsound import playsound

file = './music.mp3'

playsound(file)
```





anaconda3 › Lib › site-packages

名称

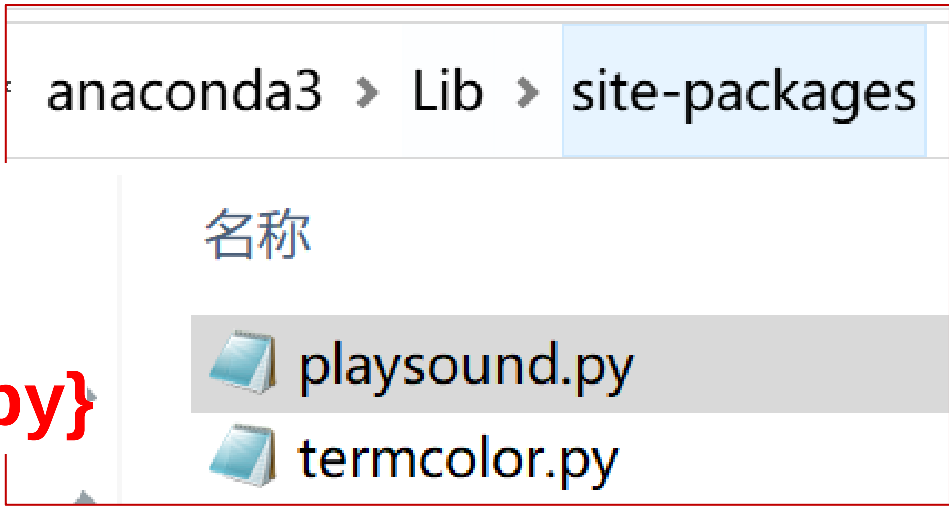
playsound.py

termcolor.py

Modules and packages

- What is Modules and packages?

They are simply Python files!



anaconda3 › Lib › site-packages

名称

playsound.py

termcolor.py

Module → *.py

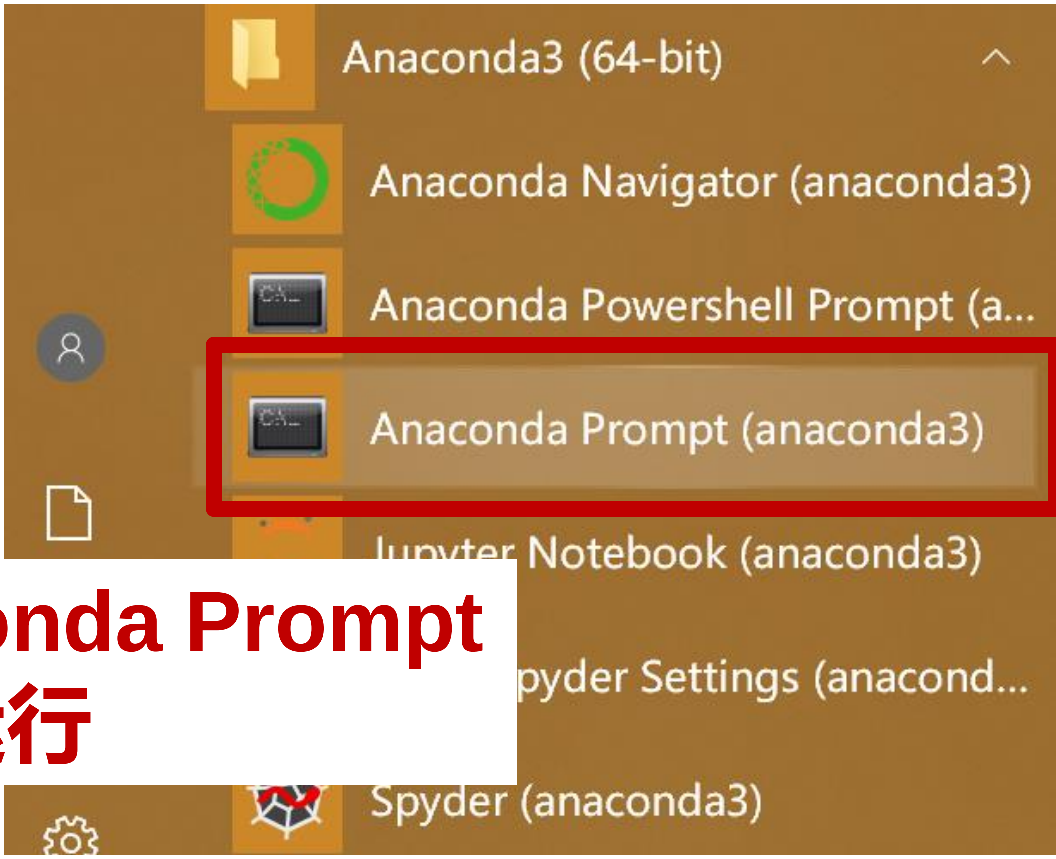
Package → Directory{*.py}

Modules and packages

Installing Modules and packages

1. pip install xxx

2. conda install xxx


Anaconda3 (64-bit)
Anaconda Navigator (anaconda3)
Anaconda Powershell Prompt (a...
Anaconda Prompt (anaconda3)
Jupyter Notebook (anaconda3)
pyder Settings (anacond...
Spyder (anaconda3)

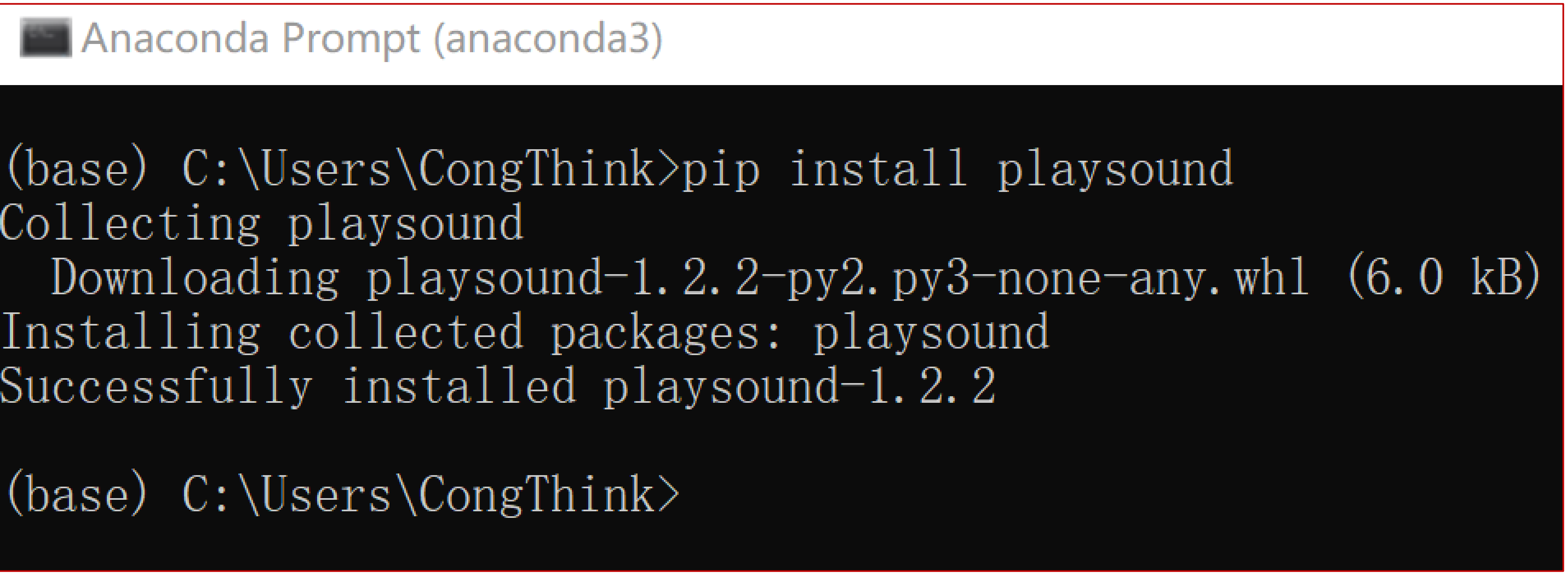
开始菜单找到Anaconda Prompt
右键以管理员身份运行

Modules and packages

- 开始菜单找到Anaconda Prompt

1. pip install playsound

- 必须联网


Anaconda Prompt (anaconda3)

```
(base) C:\Users\CongThink>pip install playsound
Collecting playsound
  Downloading playsound-1.2.2-py2.py3-none-any.whl (6.0 kB)
Installing collected packages: playsound
Successfully installed playsound-1.2.2

(base) C:\Users\CongThink>
```

Modules and packages

- **开始菜单找到Anaconda Prompt**

1. pip install pillow

- **PIL库名称为pillow**

```
from PIL import Image

img = Image.open('./1.png')
img.show()
```

Modules and packages

Pip安装如果很慢，加入清华大学镜像源

To accelerate installation:

```
pip install pip -U
pip config set global.index-url https://pypi.tuna.tsinghua.edu.cn/simp
```

from https://mirrors.tuna.tsinghua.edu.cn/help/pypi/

Modules and packages

Importing Modules and packages

1. **import** PIL
2. **from** PIL **import** Image, ImageMode
3. **from** PIL **import** Image **as** im1

   from XX import Image **as im2**
4. **from** PIL **import** *****

Modules and packages

What is exactly the imported thing?

```python
from PIL import Image
print(type(Image))
```

```
<class 'module'>
```

模块、类、函数、方法

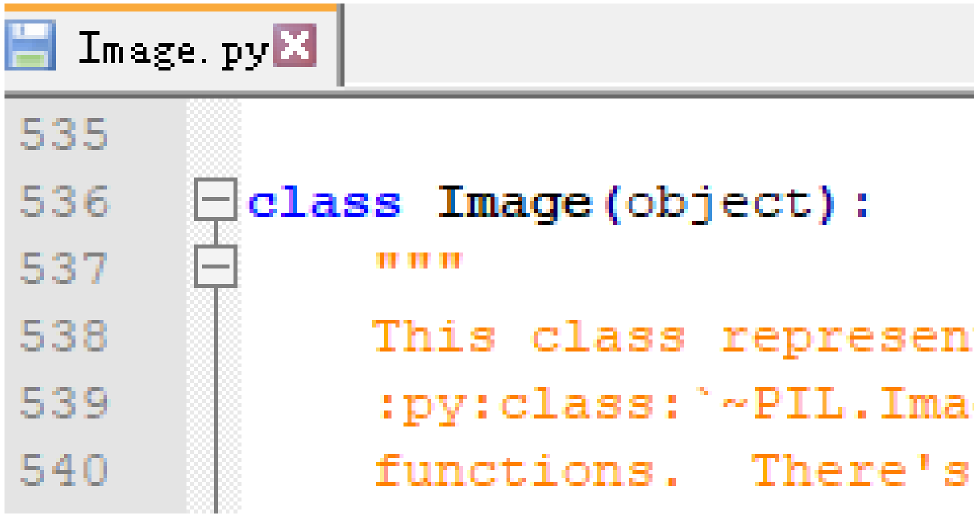


Image.py

```
class Image(object):
    """
    This class represen
    :py:class:`~PIL.Ima
    functions.  There's
```

```python
import PIL
print(type(PIL))
```

```
<class 'module'>
```

```python
from math import sqrt
print(type(sqrt))
```

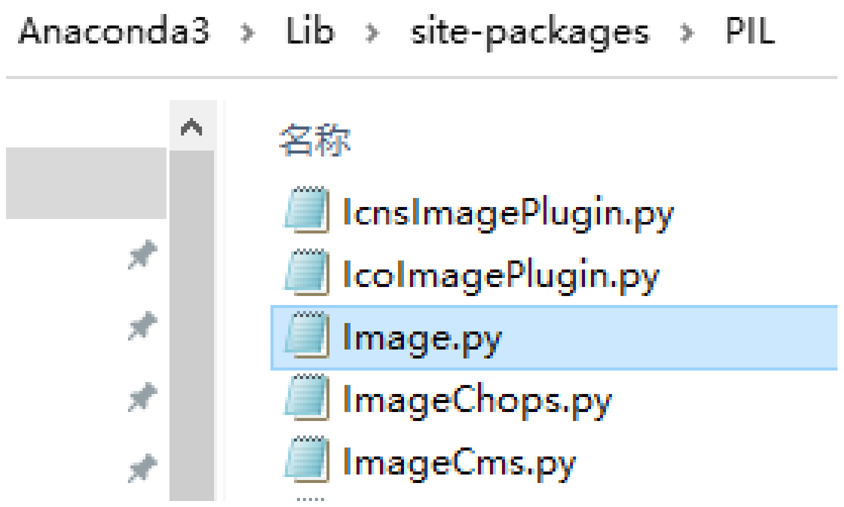
```
<class 'builtin_function_or_method'>
```

Modules and packages

① Global: Python site-packages

import xxx **OR** from xx import xxx

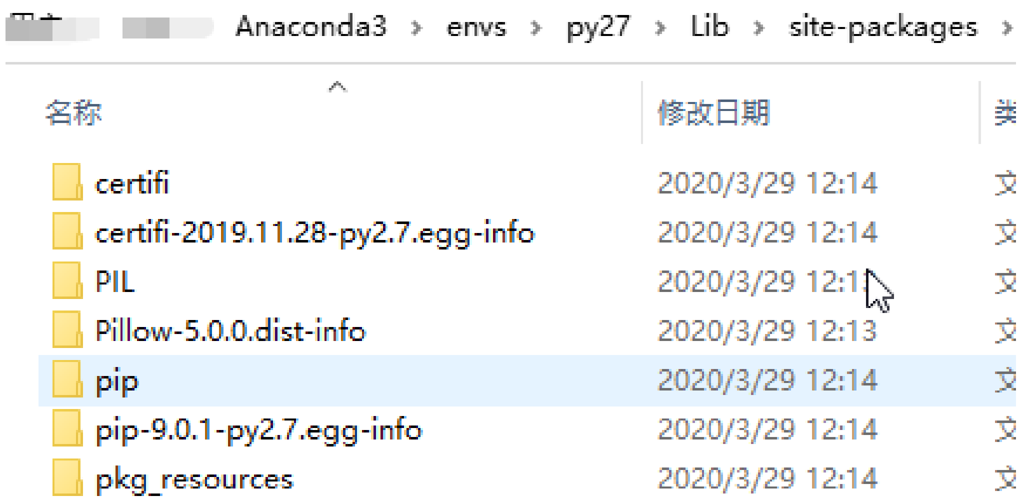
import math



Anaconda3 › Lib › site-packages › PIL

名称

IcnsImagePlugin.py
IcoImagePlugin.py
Image.py
ImageChops.py
ImageCms.py

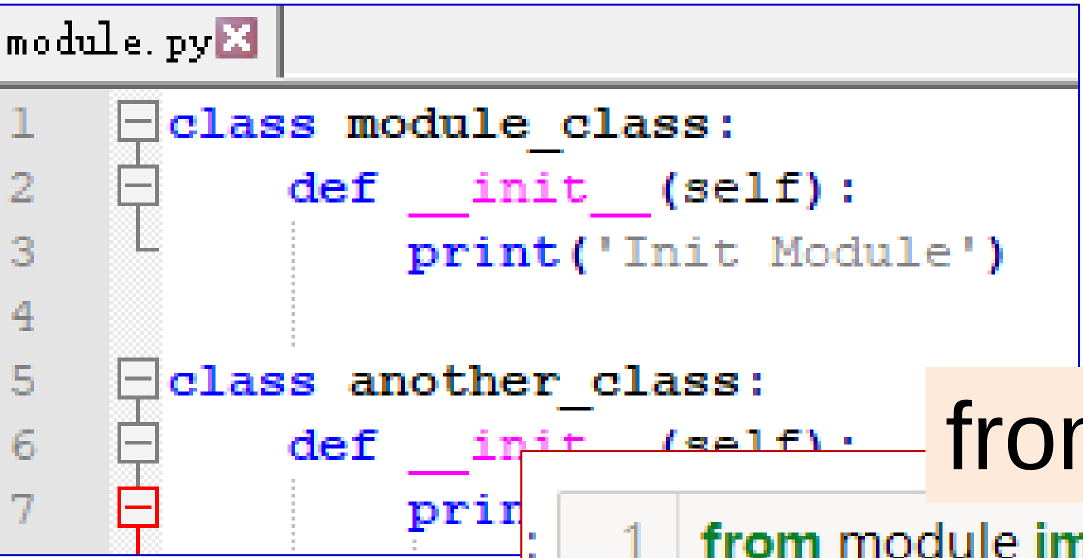


Anaconda3 › envs › py27 › Lib › site-packages ›

名称	修改日期
certifi	2020/3/29 12:14
certifi-2019.11.28-py2.7.egg-info	2020/3/29 12:14
PIL	2020/3/29 12:1
Pillow-5.0.0.dist-info	2020/3/29 12:13
pip	2020/3/29 12:14
pip-9.0.1-py2.7.egg-info	2020/3/29 12:14
pkg_resources	2020/3/29 12:14

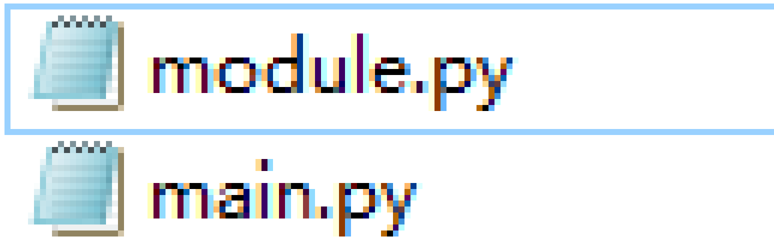
Modules and packages

② Local: Current Directory

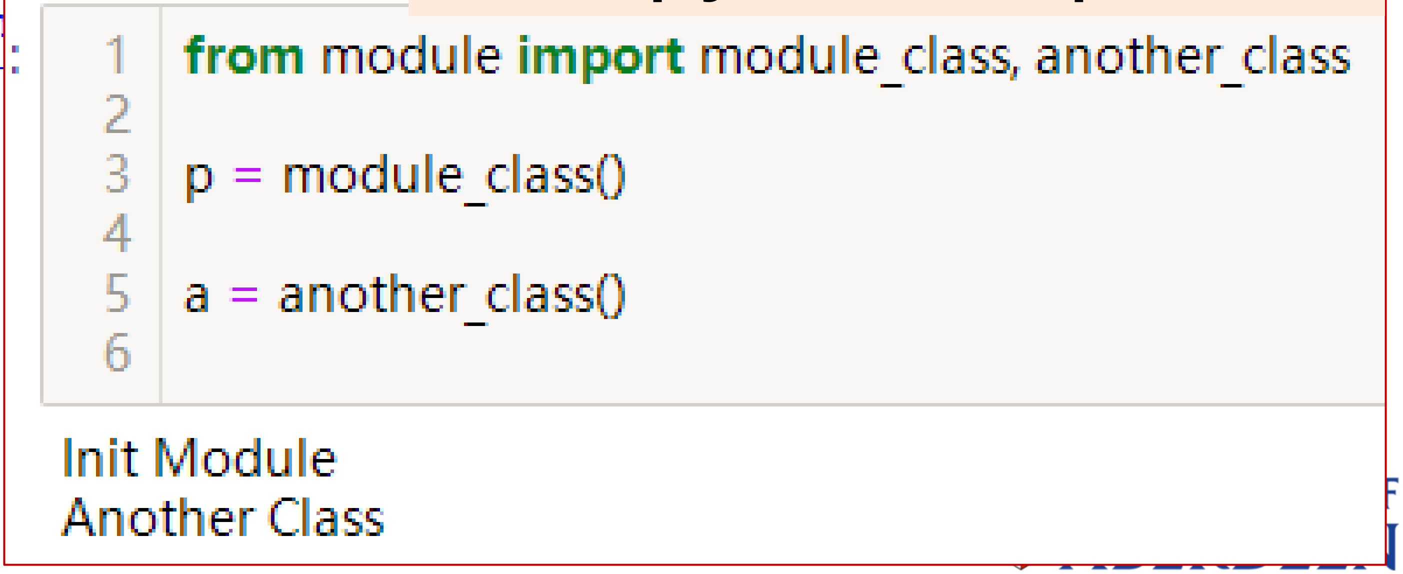


module.py

```
class module_class:
    def __init__(self):
        print('Init Module')

class another_class:
    def __init__(self):
        prin
```



from py文件 import xx

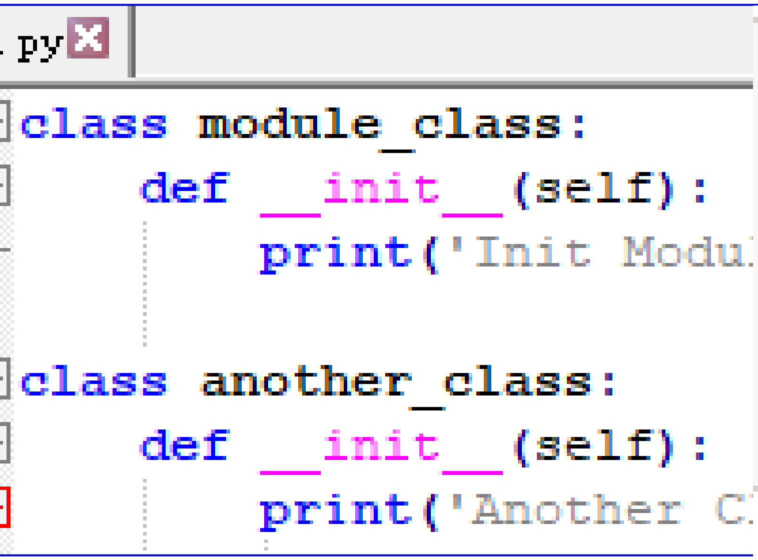


```
from module import module_class, another_class

p = module_class()

a = another_class()

```

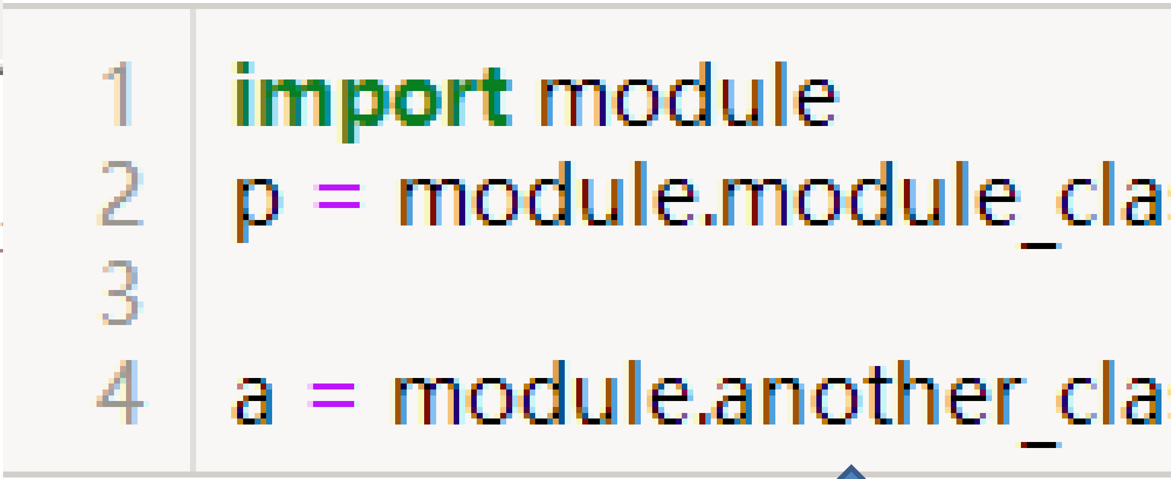
```
Init Module
Another Class
```

Modules and packages

② Local: Current Directory



```
class module_class:
    def __init__(self):
        print('Init Modu

class another_class:
    def __init__(self):
        print('Another C
```



```
import module
p = module.module_cla

a = module.another_cla
```

```
Init Module
Another Class
```

import py文件模块，再点操作符载入类

Modules and packages

② Local: Current Directory

import module → module.variable/func/class

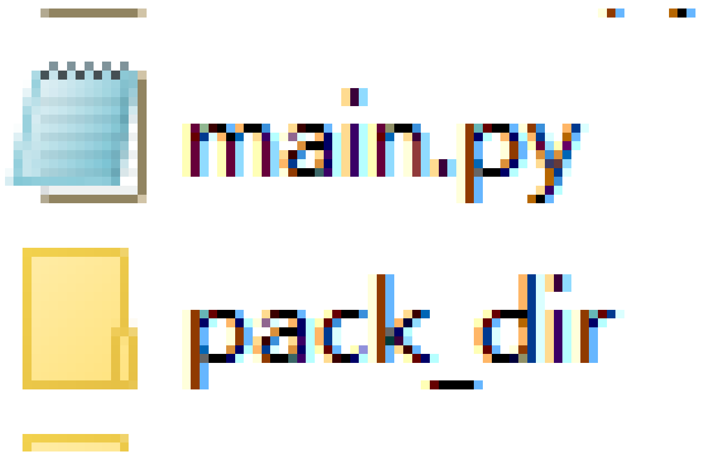
from module import variable/func/class

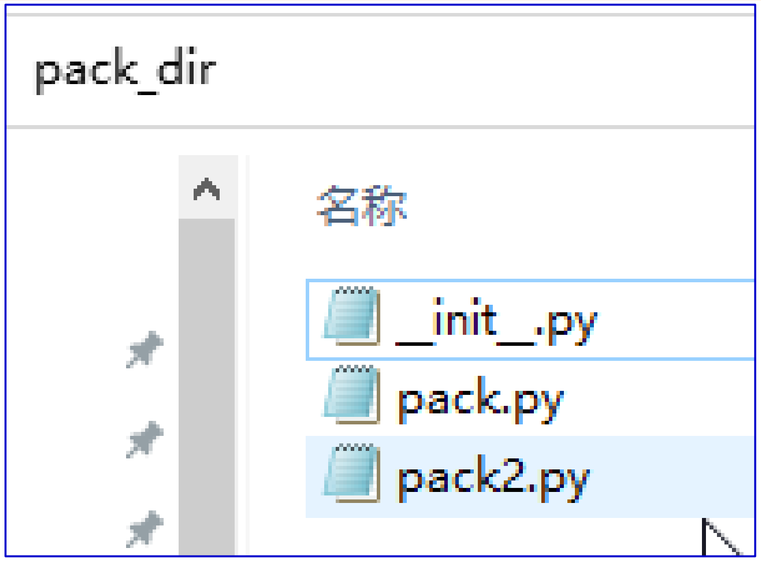
从当前目录下，载入模块的<u>变量，函数或类</u>

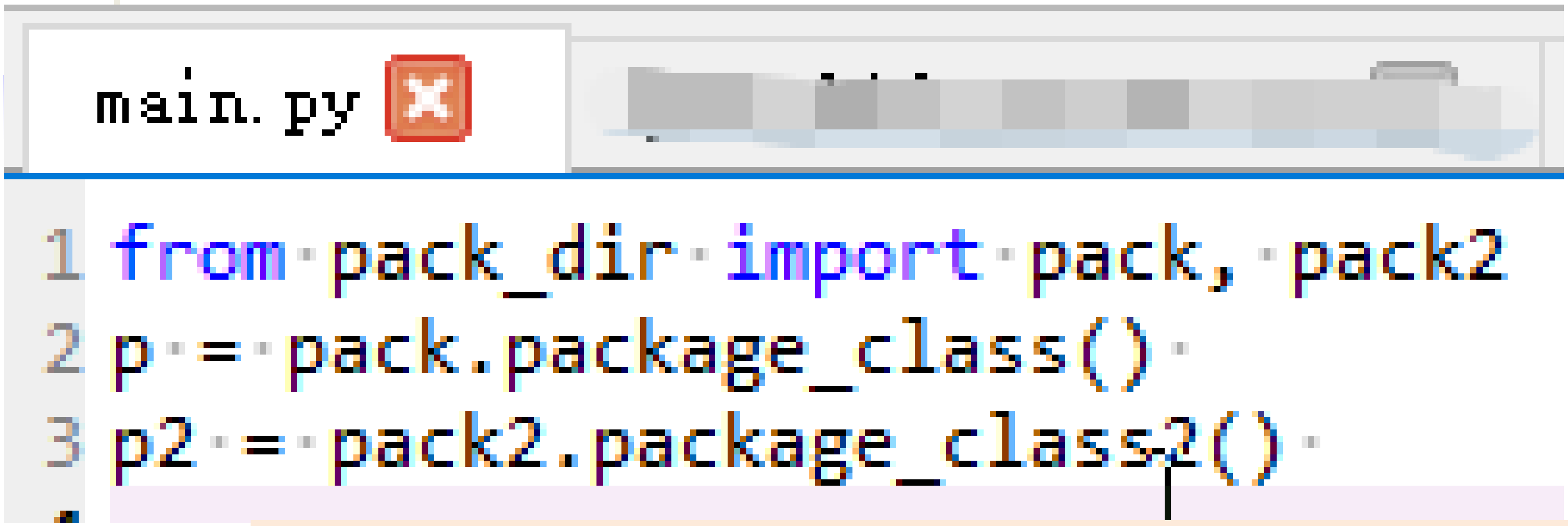
Modules and packages

③ Local: Subdirectory

从子目录下，载入模块的变量，函数或类

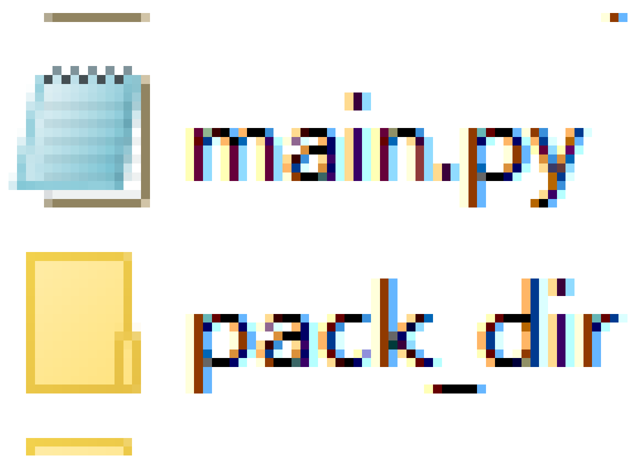


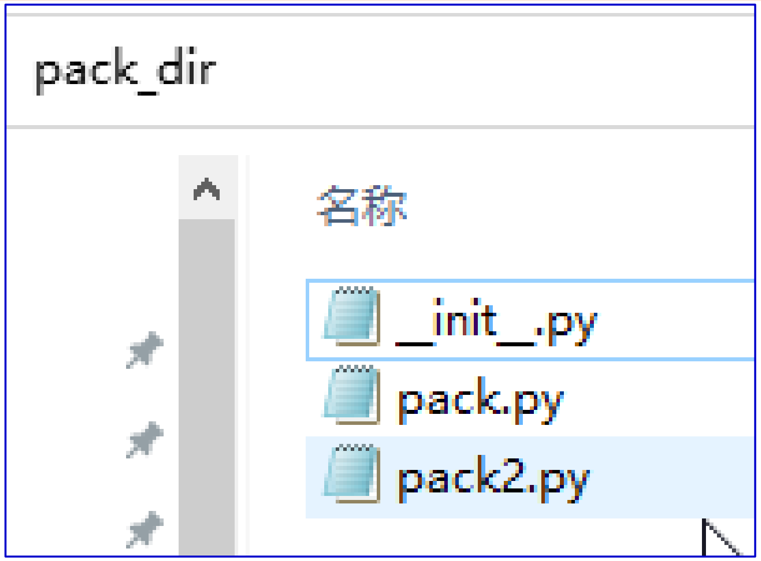


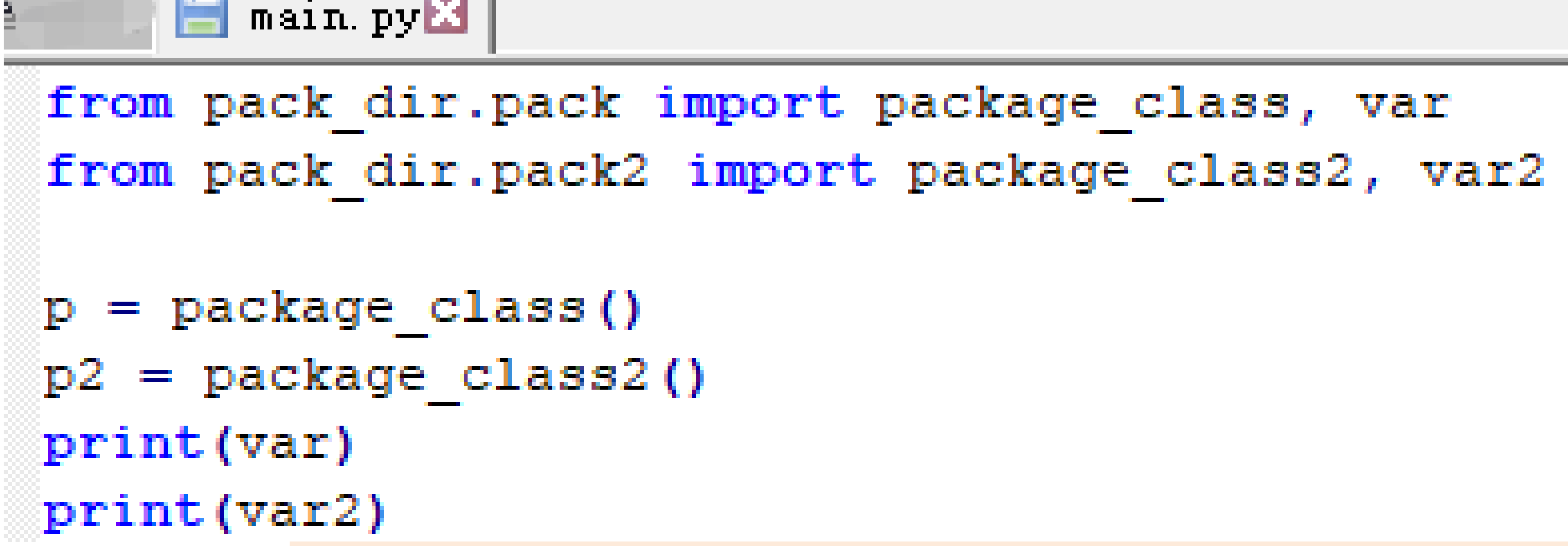


```python
from pack_dir import pack, pack2
p = pack.package_class()
p2 = pack2.package_class2()
```

从子目录下，载入模块的变量，函数或类

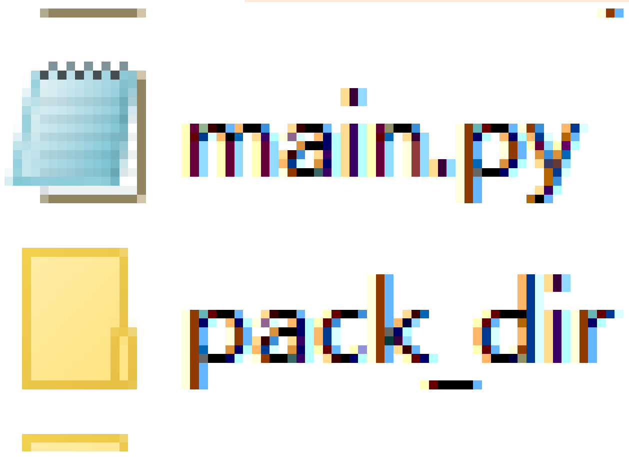


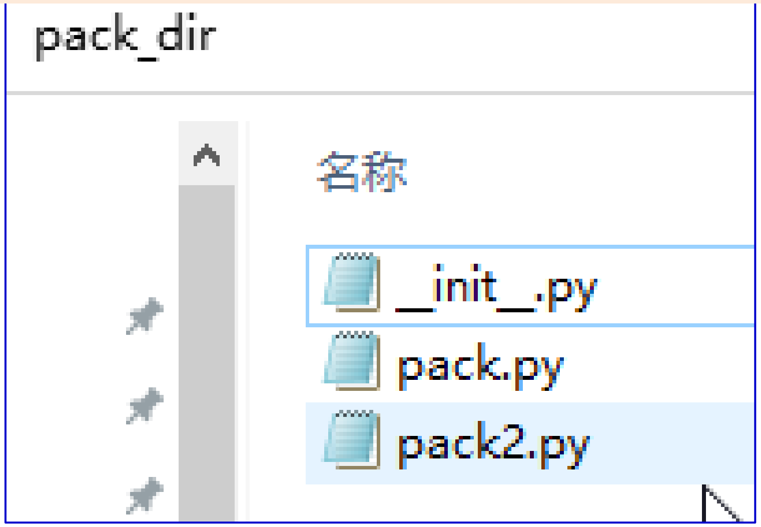



main.py

```
from pack_dir.pack import package_class, var
from pack_dir.pack2 import package_class2, var2

p = package_class()
p2 = package_class2()
print(var)
print(var2)
```

从子目录下，载入模块的变量，函数或类





Questions?

Python Class

面向对象编程中常见概念深入解析

继承：继承自拖拉机，实现了扫地的接口。
封装：无需知道如何运作，开动即可。
多态：平时扫地，天热当风扇。
重用：没有额外动力，充分利用了发动机能量。
多线程：多个扫把同时工作。

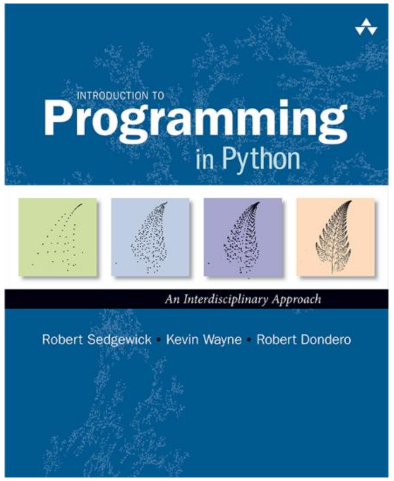
低耦合：扫把可以换成拖把而无需改动。
组件编程：每个配件都是可单独利用的工具。
适配器模式：无需造发动机，继承自拖拉机只取动力方法。
代码托管：无需管理垃圾，直接扫到路边即可。

Objectives

- Read and write Python classes.
  1. Definition 定义类
  2. Objects/Instances 对象、实例
  3. Attributes 属性（私有）
  4. Methods 方法（私有）
  5. Inheritance 继承
- Install and use modules and packages

  模块安装与使用

Introduction to Programming in Python



Thank you!