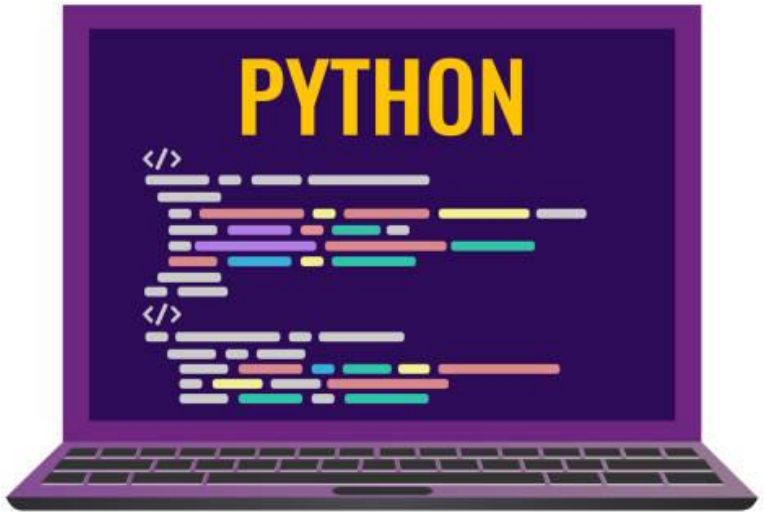
Python图像处理

叶志鹏

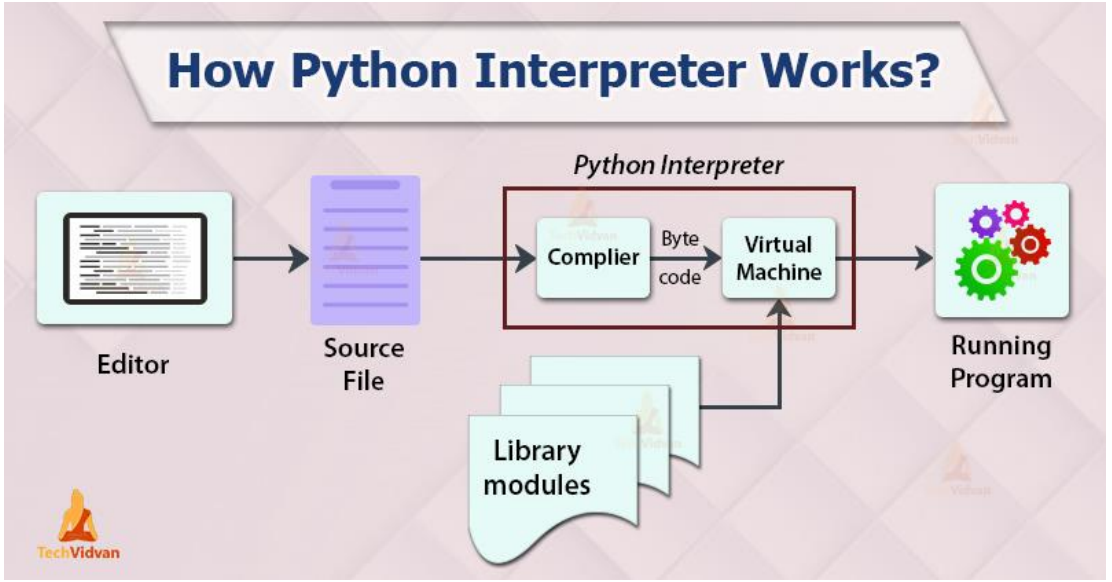
大纲

- Python 环境下载与配置
- 变量与赋值表达式
- 输入输出
- 列表，字符串与可迭代对象
- 循环与逻辑判断
- 函数（复用代码块）
- 元组，字典，集合
- Python的浅拷贝与深拷贝
- 读写文件
- 面向对象编程
- 模块与包
- 常用图像处理的包与模块



Python 解释器

- Python 是一门解释型语言
- 编译型语言：有单独编译过程，讲程序翻译成机器码
- 解释型语言：没有单独编译过程，每次运行一条指令，才进行翻译。
- 所以，我们得有Python解释器，才能运行程序，就像C语言需要 gcc 编译器。



Python 解释器的下载

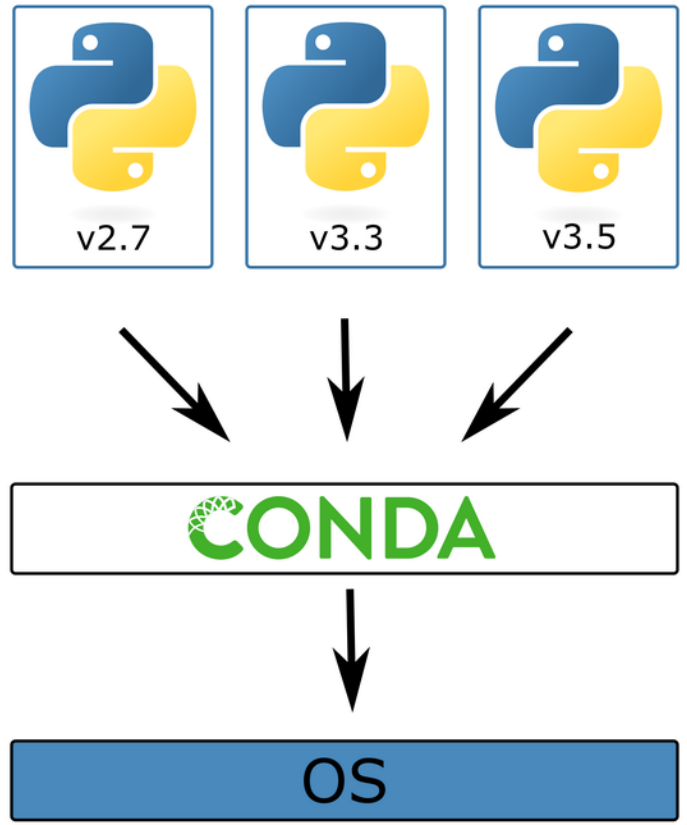
1. 官网下载安装

   https://www.python.org/downloads/

2. Anaconda 下载安装

   https://www.anaconda.com/

对比：Anaconda有更好的环境隔离，保障用户程序的依赖管理，Python官网下载比较简单。

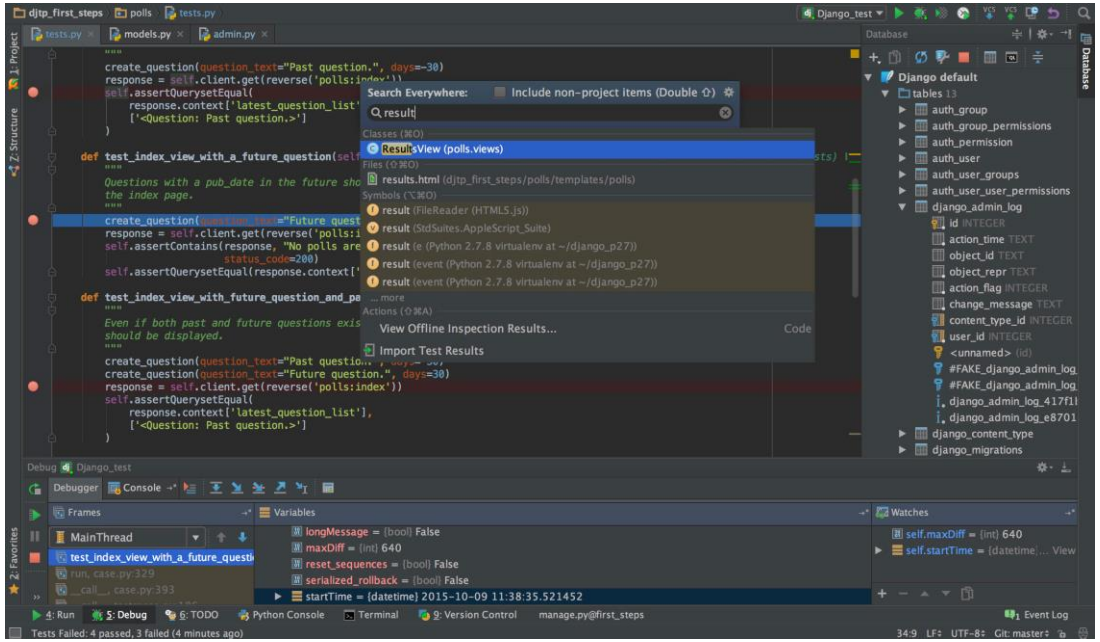


Python编辑器/集成开发环境

1. 记事本 ✓
2. Vim ✓✓
3. Visual Studio code ✓✓✓
4. PyCharm ✓✓✓✓

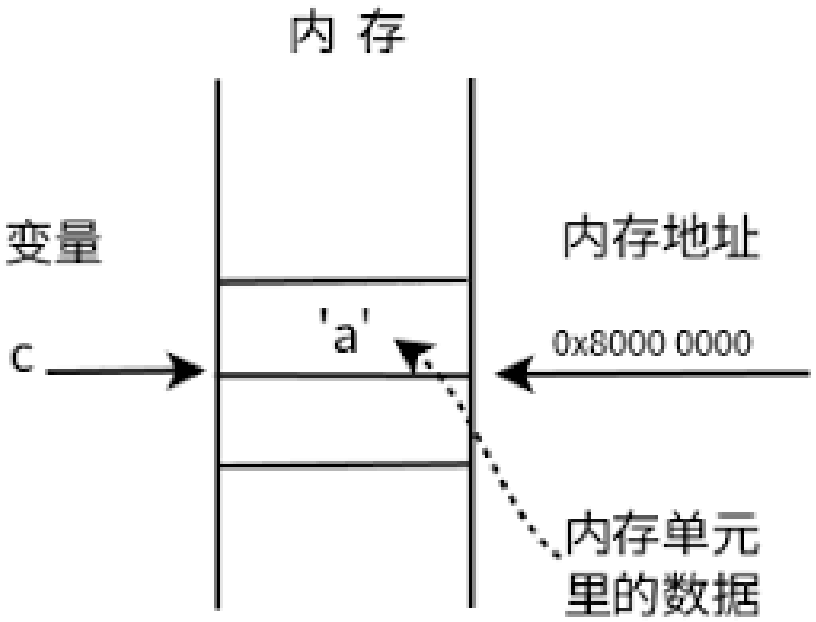
具体配置：请 Bing/Google/Baidu

（计算机专业学生的基本素养）



Python 变量与赋值表达式

- 变量定义
  - Python的变量会在内存中开辟一个空间，用来存放数据。
- 变量与赋值表达式
  - 变量名 = 变量值
  - 例如 a = 10
- 变量名书写要求：
  - 变量名可以由字母、下划线和数字组成
  - 不能以数字开头、不能与关键字重名，关键字就是在python内部已经使用的名称
  - 另外，请取有意义的变量名



Python 输入输出

- 输入

  a = input()

- 输出

  print(‘Hello world’)

- 变量

  a = 10

  print(a)

- 格式化输入

  a = input(‘please input your number:’)

- 格式化输出

  a = ‘Tom’

  b = ‘James’

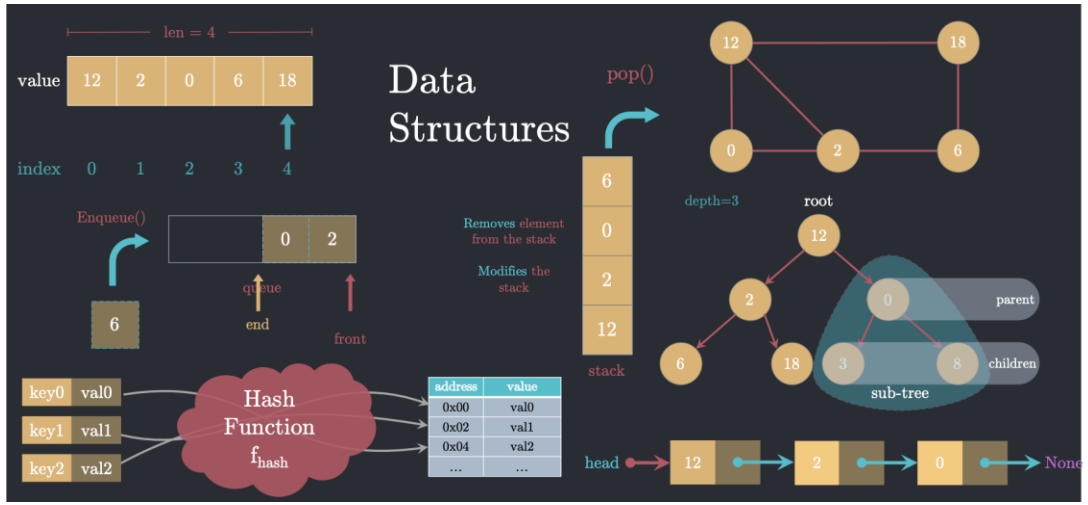
  print(a, b)

  print(f’{a} is {b}\’s friend’)

  print(f”{a} is {b}\’s friend”)

Python 数据类型

- 基础数据类型:
  - 整数 1 2 10
  - 浮点数 1.2321 3.1415926
    1.123131
  - 布尔值 True False
  - 字符串 “Hello world” ‘James’ ‘文\n 本’
  - 空值 None
- 高级数据结构
  - List 列表 [‘a’,’b’,’d’,’f’,’a’]
  - Dictionary 字典 {‘a’:1,’b’:10,’d’:200}
  - Set 集合 {1,2,3,4,5}



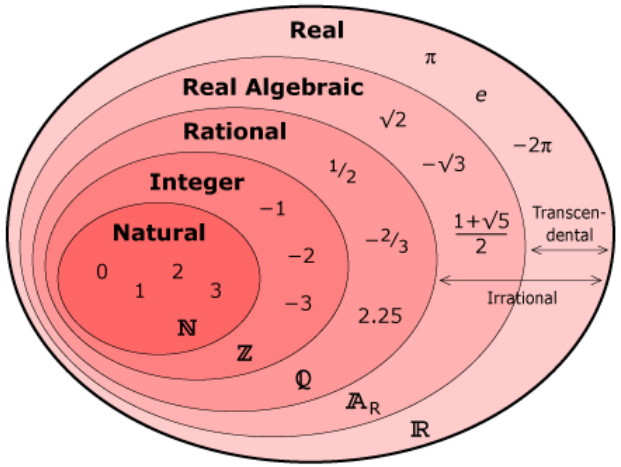
Python 整数与浮点数

- 整数

```
a = 1
b = 20
c = a*b
print(c)
```

- 浮点数

```
a = 1.12342112454356
b = 231231
print(a*b)
```



Python 算术运算符

- Python 自带了一些基础的数学运算，如加减乘除
- 一切复杂运算都可以通过加减乘除模拟
- 比如数值微分，数值积分法
  - $f'(x) = \lim_{x \to 0} \frac{f(x+\Delta x) - f(x)}{\Delta x}$

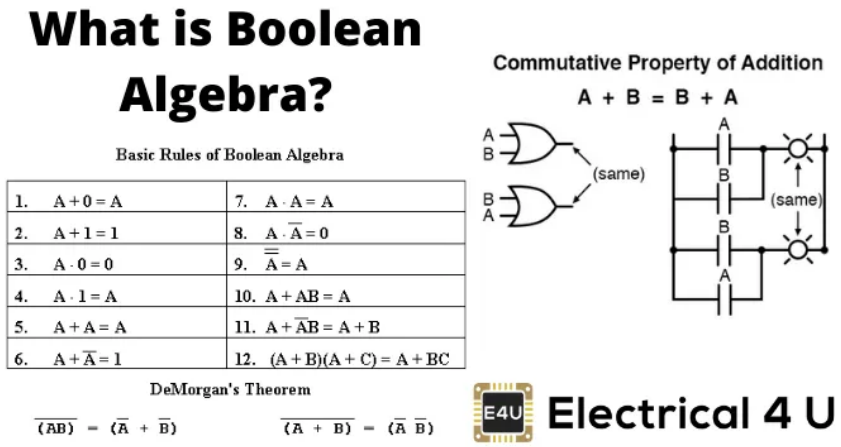
Python 比较运算符

- Python 比较运算符会返回布尔值 True / False
- 区分 == 比较运算符 与 = 赋值运算符
- 常用在 if else, while 等结构中

比较运算符	含义
>	大于运算符
>=	大于等于运算符
<	小于运算符
<=	小于等于运算符
==	等于运算符
!=或<>	不等于运算符

Python 布尔值

- Python 布尔值 True / False
- 支持布尔运算符
  - and, or, not
- 常与比较运算符连用



Python 列表，字符串与可迭代对象

- Python 列表的定义

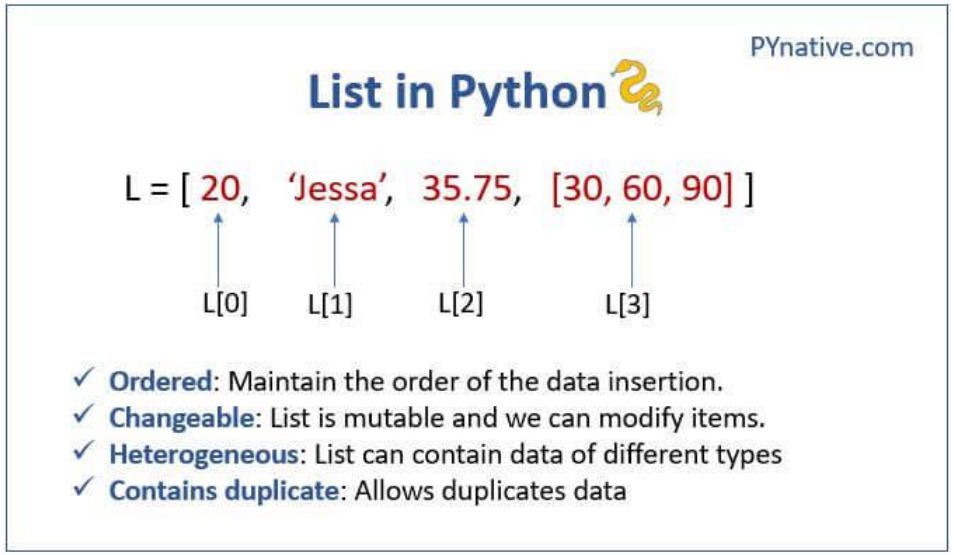
  a = [1,3,1,5,6,1], a[[1,2],[3,4]]

- 访问 List 元素

  a[0], a[len(a)-1], a[-1]

- 切片，获取某一段

  a[1:3], a[:3], a[3:],a[::-1],a[1:5:2]



Python 列表，字符串与可迭代对象

- 列表增加元素

  a.append(’test’), a.extend([‘a’,’b’]),

  a.insert(2, ‘b’)

- 列表删除元素

  a.remove(‘test’), val = a.pop(3), a.pop(),

  del a[1] (del 可操作多维 list)

- 列表修改元素

  a[0] = ‘p’, a[1:5] = [‘a’, ‘b’, ‘c’, ‘d’]

- 列表查找元素

  ‘test’ in a, a.count(‘a’), a.index(‘b’)

- 列表是可迭代对象

```
for val in a:
    print(val)
For I in. range(10):
    print(i)
```

- List comprehension

```
a = [i for i in range(10)]
```

Python 列表，字符串与可迭代对象

- Python 字符串的定义 (无法修改)

  str_ = ‘Hello world’, str_ = “Hello world”

- 字符串中的转译字符

  \n, \t, \\, \’, \”

- 原格式定义

  “”” ”””

- 获取字符串的长度

  len(str_)

- 字符串获取单个字符，或者字串跟 list 一样

  str_[index], str_[start:end:step]

- 字符串的拼接

  ’asdsfwer’+’fwetwreq’

  ’a’*10+’b’*5+’a’*10

- 字符串查找元素

  ‘test’ in a, a.count(‘a’), a.index(‘b’) (会报错)

- 字符串的比较运算符

  >, >=, <, <= , ==, !=

- 字符串的方法

  .center(50) 居中

  .find() 查找字串位置 (找不到，返回-1)

  .replace() 替换

  .strip() 删除字符串的开头与末尾空格

- 字符串的迭代:

```
for val in str_:
    print(val)
```

循环与逻辑判断

- 循环结构 (注意冒号与缩紧) for 循环

```
for i in 可迭代对象:
        print(i)
for i in range(start, end, step):
        print(i)
```

- While 循环

```
a = 0
While  a<= 10:
        print(a)
        a = a+1
```

- 嵌套循环

```
for i in range(10):
        for j in range(10):
                print(i, j)

i = 0
While i <= 100:
        j = 0
        While j <= 100:
                print(i, j)
```

- 跳出当次循环 continue
- 跳出当前循环体 break

循环与逻辑判断

- 逻辑判断

```
if 布尔值(或布尔表达式):
        print()
elif 布尔值(或布尔表达式):
        print()
elif 布尔值(或布尔表达式):
        print()
else:
        print()
```

- 比如

```
age = 20
if age > 0 and age < 10:
        print('He is child')
elif age < 25:
        print('He is teenager')
elif age < 50:
        print('He is adult')
else:
        print('He is old')

age = 20
if age > 0 and age < 10:
        print('He is child')
if age >= 10 and age < 25:
        print('He is teenager')
if age >= 25 and age <= 50:
        print('He is adult')
If age > 50:
        print('He is old')
```

函数（复用代码块）

- 语法：

```
def fun_name(param1, param2, param3):
        …….
        return some_value
```

- 调用函数

```
y = fun_name(1, 2, 3)
```

- 函数的参数
  - 可以是任意对象类型
  - 定义的时候是形参，调用的时候是实参
  - 参数的默认值

```
def fun_name(param1=1, param2=3, param3=6):
        …….
        return some_value
y = fun_name(1)
```

- 局部变量与全局变量

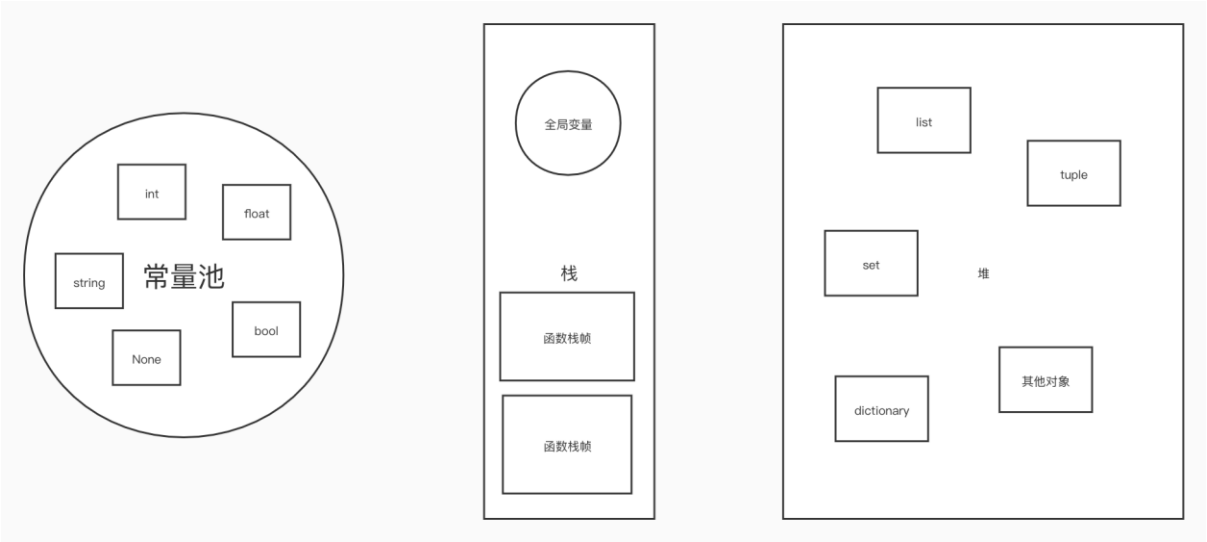
```
//全局变量
a = 20
def add_one():
        //局部变量
        a = 20
add_one()
print(a)
```

- 可变数据类型与不可变数据类型

```
a = 20
def add_one(a):
        a=30
add_one(a)
print(a)

a = [1,2,3]
def modify(a):
        a[0] = 3
modify(a)
print(a)
```

可变数据类型与不可变数据类型

- 不可变数据类型
  - 整数
  - 浮点数
  - 布尔值
  - 字符串
  - None
  - tuple 元组
- 可变数据类型
  - 列表
  - 字典
  - 集合



元组，字典，集合

- 元组 tuple 定义

  a = (1, 2, 45) 或者直接 a = 1, 2, 45

- 元组和列表数据结构类似，但是无法修改，删除或增加

  a[0], a[1]

- 元组的拆包

  a, b = (3, 5)

- 元组的常用场景
- 交换 2 个数的值

  x, y = y, x

- 封装成元组，使函数具有返回多个值的功能

```
def test():
    return 1,2,3
x, y, z = test()
```

元组，字典，集合

- 字典 dictionary 定义

  Key : value 形式的数据结构

- 语法：

  student = {‘name’: ‘Zhipeng’, ‘age’: 30,

  ‘friends’ : [‘james’, ‘Euler’]}

  print(student[‘friends’])

- 通过键获取值

  student[‘name’] (找不到会报错),

  student.get(‘name’)

- 增加 key, value

  student[’school’] =. ‘Ulink’

- 修改字典的值

  student[‘school’] = ‘dongnan’

- 合并并修改字典

  student1 = {'name': 'Zhipeng', 'age': 30}
  student2 = {'age': 20}
  student1.update(student2)
  print(student1)

- 字典删除键值对

  del student1[‘age’], student1.pop(‘name’)

元组，字典，集合

- 字典删除键值对

  del student1[‘age’], student1.pop(‘name’)

- 字典可以根据 Key 遍历

```
for key in student1:
    print(key)
```

- 字典遍历值

```
for key in student1:
    print(key)
```

元组，字典，集合

- 集合的定义: (不包含重复元素)

  student_id = {1, 2, 3} student_id = {}, student_id = set()

- 集合与 List 互转

  student_id = [1, 2, 1]

  student_set = set(student_id)

  stu_list = list(student_set)

- 增加元素

  student.add(‘a’)

- 合并集合

  a = {1,2} b = {2, 3} a.update(b)

- 删除集合的元素

  a.remove(1), a.discard(1), a.pop()

- 判断元素在不在集合内

  2 in a

- 集合大小

  len(a)

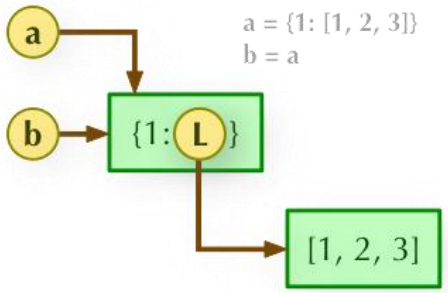
- 集合的数学运算

  a & b , a | b

Python的浅拷贝与深拷贝

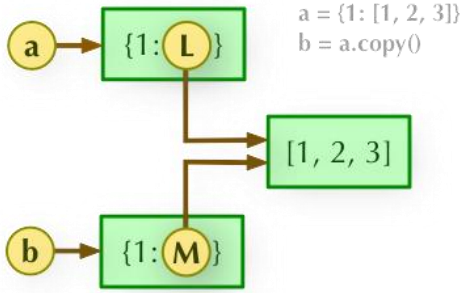
- 直接赋值：

```
a = [1,2,3]
b = a
b[0] = 2
print(a)
```



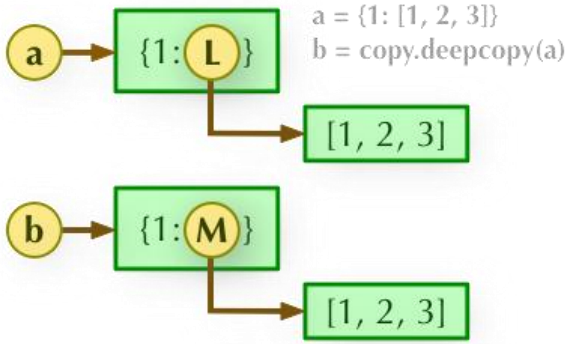
- 浅拷贝：

```
a = {1:[1,2,3]}
b = a.copy()
b[1][0] = [2]
print(a)
print(b)
```



- 深拷贝：

```
import copy
a = {1:[1,2,3]}
b = copy.deepcopy(a)
b[1][0] = [2]
print(a)
print(b)
```



文件读写

- 一次性读文件

```
with open('breast-cancer-wisconsin.data','r') as file:
    content = file.read()
    print(content)
```

- 按照行分批读文本

```
with open('breast-cancer-wisconsin.data','r') as file:
    for line in file:
        print(line)
```

- 覆盖模式写文件

```
a = [0]*10000
with open('test.txt','w') as file:
    file.write(str((a)))
a = [0]*10000
with open('test.txt','w') as file:
    for val in a:
        file.write(str(val)+'\n')
```

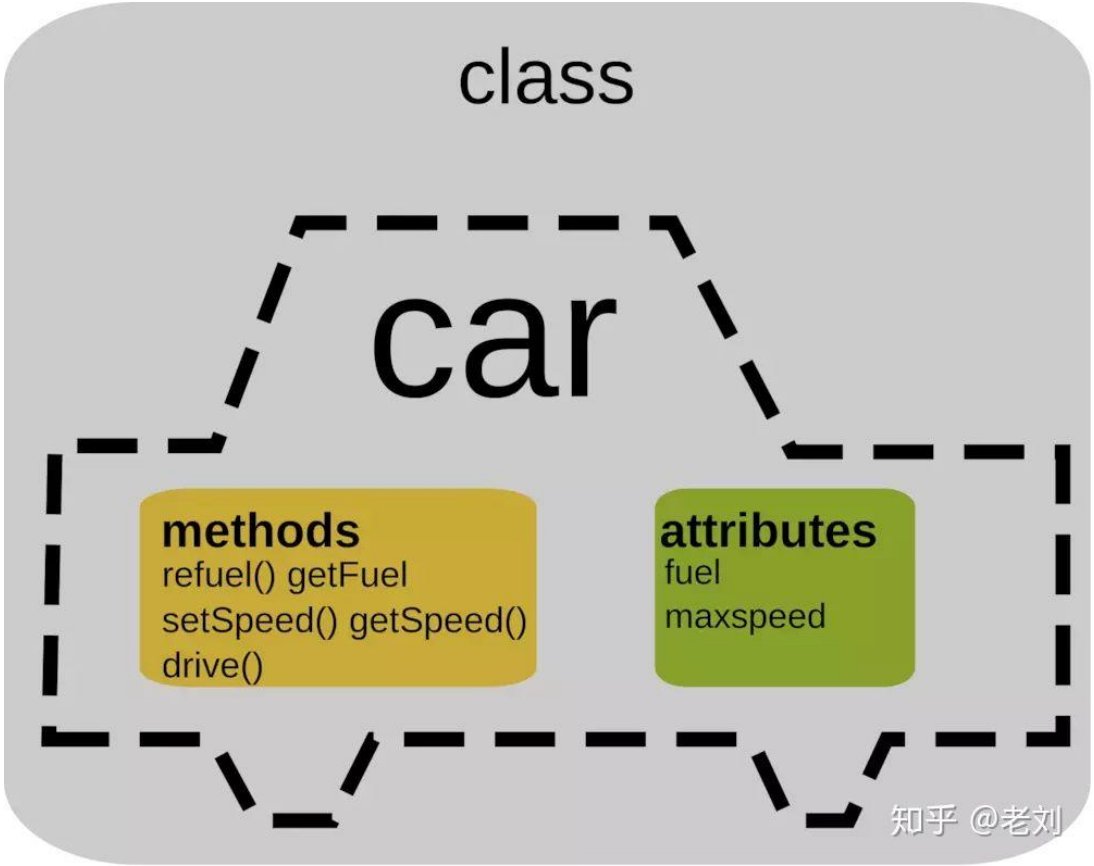
- 追加模式写文件

```
with open('test.txt','a') as file:
    for val in a:
        file.write(str(val)+'\n’)
```

- ‘w+’, ‘a+’ 没有文件名就创建, 有的话照常

面向对象编程（OOP）

- 传统的面向过程编程是将问题拆分成步骤，分步实现
- 面向对象编程将问题转化成对象的信息交互
- 面向对象编程包含封装，继承，多态三大性质



面向对象编程 （OOP）

- 问题： 洗衣机里面放有脏衣服，怎么洗干净？
- 面向过程编程
    1. 放入衣服
    2. 加洗衣液
    3. 开洗衣机
    4. 拿出衣服

- 面向对象编程

```
Class person:
        def 放衣服(self):
                ...
        def 加洗衣液(self):
                ...
        def 拿衣服(self):
                ...
Class washer:
        def start(self):
                加水
                搅拌
                ...
        def 加水(self):
                ...
        def 搅拌(self):
Person.放衣服()
Person.加洗衣液()
Washer.start()
Person.拿衣服()

                ...
```

面向对象编程（OOP）- 类与对象

- 类与对象
  - 类是对象的模版
  - 对象是类的内存实例
- class Student:

  pass

  student1 = Student()

  student2 = Student()

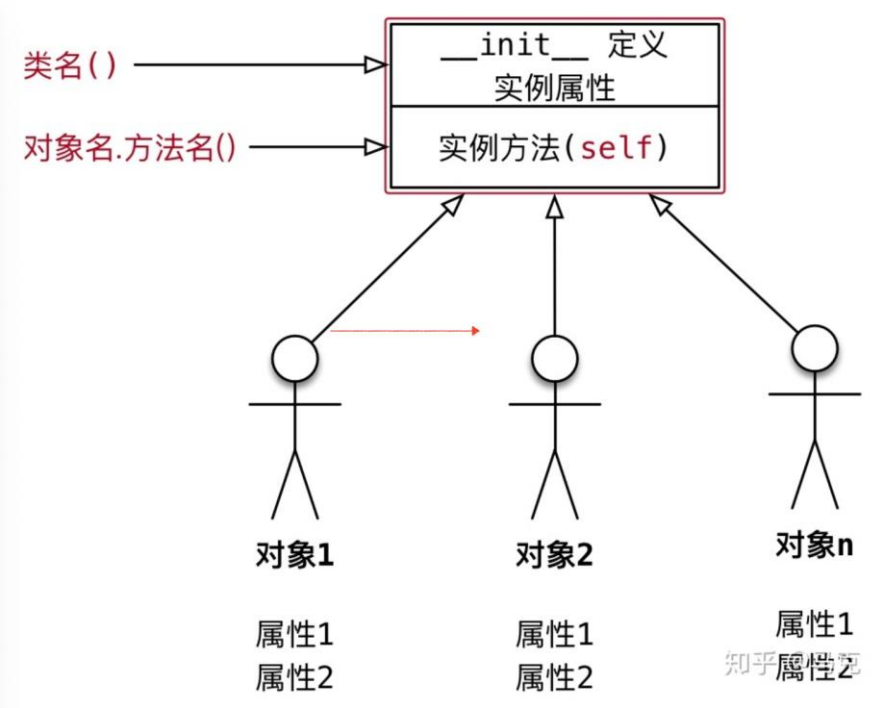
  print(student1, student2)

<__main__.Student object at 0x7fb2fde4e828> <__main__.Student object at 0x7fb2fde7d358>

面向对象编程 （OOP） - 属性与方法

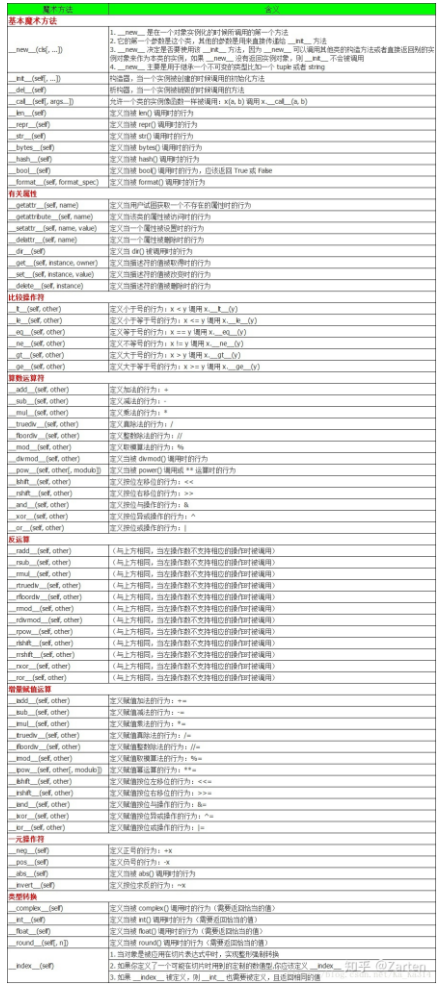
- 属性与__init__(self) 方法

```
class student:
    //属性
    def __init__(self, name, age):
        self.name = name
        self.age = age
    //方法
    def print_score(self):
        print(f'{self.name} {self.age}')
```



面向对象编程（OOP）- 特殊方法

- 特殊方法是支持对象进行运算的工具
- 如__len__(), __str__(), __add__(), __mul__(), __getitem__()



魔术方法	含义
基本魔术方法	
__new__(cls[, ...])	1. __new__ 是在一个对象实例化时所调用的第一个方法 2. 它的第一个参数是这个类，其他的参数是用来直接传递给 __init__ 方法 3. __new__ 决定是否要使用该 __init__ 方法，因为 __new__ 可以调用其他类的构造方法或者直接返回别的实例对象来作为本类的实例，如果 __new__ 没有返回实例对象，则 __init__ 不会被调用 4. __new__ 主要是用于继承一个不可变的类型比如一个 tuple 或者 string
__init__(self, ...)	构造器，当一个实例被创建的时候调用的初始化方法
__del__(self)	析构器，当一个实例被销毁的时候调用的方法
__call__(self, args...)	允许一个类的实例像函数一样被调用：x(a, b) 调用 x.__call__(a, b)
__len__(self)	定义当被 len() 调用时的行为
__repr__(self)	定义当被 repr() 调用时的行为
__str__(self)	定义当被 str() 调用时的行为
__bytes__(self)	定义当被 bytes() 调用时的行为
__hash__(self)	定义当被 hash() 调用时的行为
__bool__(self)	定义当被 bool() 调用时的行为，应该返回 True 或 False
__format__(self, format_spec)	定义当被 format() 调用时的行为
有关属性	
__getattr__(self, name)	定义当用户试图获取一个不存在的属性时的行为
__getattribute__(self, name)	定义当该类的属性被访问时的行为
__setattr__(self, name, value)	定义当一个属性被设置时的行为
__delattr__(self, name)	定义当一个属性被删除时的行为
__dir__(self)	定义当 dir() 被调用时的行为
__get__(self, instance, owner)	定义当描述符的值被取得时的行为
__set__(self, instance, value)	定义当描述符的值被改变时的行为
__delete__(self, instance)	定义当描述符的值被删除时的行为
比较操作符	
__lt__(self, other)	定义小于号的行为：x < y 调用 x.__lt__(y)
__le__(self, other)	定义小于等于号的行为：x <= y 调用 x.__le__(y)
__eq__(self, other)	定义等于号的行为：x == y 调用 x.__eq__(y)
__ne__(self, other)	定义不等号的行为：x != y 调用 x.__ne__(y)
__gt__(self, other)	定义大于号的行为：x > y 调用 x.__gt__(y)
__ge__(self, other)	定义大于等于号的行为：x >= y 调用 x.__ge__(y)
算数运算符	
__add__(self, other)	定义加法的行为：+
__sub__(self, other)	定义减法的行为：-
__mul__(self, other)	定义乘法的行为：*
__truediv__(self, other)	定义真除法的行为：/
__floordiv__(self, other)	定义整数除法的行为：//
__mod__(self, other)	定义取模算法的行为：%
__divmod__(self, other)	定义当被 divmod() 调用时的行为
__pow__(self, other[, module])	定义当被 power() 调用或 ** 运算时的行为
__lshift__(self, other)	定义按位左移位的行为：<<
__rshift__(self, other)	定义按位右移位的行为：>>
__and__(self, other)	定义按位与操作的行为：&
__xor__(self, other)	定义按位异或操作的行为：^
__or__(self, other)	定义按位或操作的行为：|
反运算	
__radd__(self, other)	（与上方相同，当左操作数不支持相应的操作时被调用）
__rsub__(self, other)	（与上方相同，当左操作数不支持相应的操作时被调用）
__rmul__(self, other)	（与上方相同，当左操作数不支持相应的操作时被调用）
__rtruediv__(self, other)	（与上方相同，当左操作数不支持相应的操作时被调用）
__rfloordiv__(self, other)	（与上方相同，当左操作数不支持相应的操作时被调用）
__rmod__(self, other)	（与上方相同，当左操作数不支持相应的操作时被调用）
__rdivmod__(self, other)	（与上方相同，当左操作数不支持相应的操作时被调用）
__rpow__(self, other)	（与上方相同，当左操作数不支持相应的操作时被调用）
__rlshift__(self, other)	（与上方相同，当左操作数不支持相应的操作时被调用）
__rrshift__(self, other)	（与上方相同，当左操作数不支持相应的操作时被调用）
__rxor__(self, other)	（与上方相同，当左操作数不支持相应的操作时被调用）
__ror__(self, other)	（与上方相同，当左操作数不支持相应的操作时被调用）
增量赋值运算	
__iadd__(self, other)	定义赋值加法的行为：+=
__isub__(self, other)	定义赋值减法的行为：-=
__imul__(self, other)	定义赋值乘法的行为：*=
__itruediv__(self, other)	定义赋值真除法的行为：/=
__ifloordiv__(self, other)	定义赋值整数除法的行为：//=
__imod__(self, other)	定义赋值取模算法的行为：%=
__ipow__(self, other[, module])	定义赋值幂运算的行为：**=
__ilshift__(self, other)	定义赋值按位左移位的行为：<<=
__irshift__(self, other)	定义赋值按位右移位的行为：>>=
__iand__(self, other)	定义赋值按位与操作的行为：&=
__ixor__(self, other)	定义赋值按位异或操作的行为：^=
__ior__(self, other)	定义赋值按位或操作的行为：|=
一元操作符	
__neg__(self)	定义正号的行为：+x
__pos__(self)	定义负号的行为：-x
__abs__(self)	定义当被 abs() 调用时的行为
__invert__(self)	定义按位求反的行为：~x
类型转换	
__complex__(self)	定义当被 complex() 调用时的行为（需要返回恰当的值）
__int__(self)	定义当被 int() 调用时的行为（需要返回恰当的值）
__float__(self)	定义当被 float() 调用时的行为（需要返回恰当的值）
__round__(self[, n])	定义当被 round() 调用时的行为（需要返回恰当的值）
__index__(self)	1. 当对象是被应用在切片表达式中时，实现整形强制转换 2. 如果你定义了一个可能在切片时用到的定制数值型，你应该定义 __index__ 3. 如果 __index__ 被定义，则 __int__ 也需要被定义，且返回相同的值

面向对象编程 (OOP)- 封装，继承，多态

- 封装就是把对象的属性或方法屏蔽（在属性与方法前加上__），多态性指子类们继承同一个父类的同一个方法，可以有不同的表现形式 [课程不要求掌握]
- 继承：子类会继承父类的属性和方法，但是允许子类覆盖父类的属性与方法

```
class  Person:
    def __init__(self):
            self.name = 'test'
    def speak(self):
            print('I am a person')
class student(Person):
    def speark(self):
            print('I am a student')
```

面向对象编程 （OOP） - 万物皆对象

- Python 中所有的数据类型都是对象，包括函数也是对象
- 所有的对象都继承与父类，最后归结到上帝类 Object

```
print(type(123))
print(type(123.1))
print(type('abc'))
print(type([1,2,3]))
print(type({1,2,3,5,1}))
print(type({'a':1,'b':2,'c':3}))
```

```
<class 'int'>
<class 'float'>
<class 'str'>
<class 'list'>
<class 'set'>
<class 'dict'>
```

```
def test(a):
    print('hello!')
print(type(test))
```

```
<class 'function'>
```

```
print(type(type))
```

```
<class 'type'>
```

模块与包

- 为了代码的维护，项目化，创建了模块与包的概念
- 模块：
  - 一个 .py 文件就是一个模块
- 包：
  - 装有模块的文件夹

模块与包

- Python 内置的模块
  - 如 random， math，sys, os, re, pickle, argparse 模块等。（查阅官网文档）
- Python 模块的使用
  - import random

    print(random.random())
  - import random as rd (别名)

    print(rd.random())

模块与包

- Python 自定义模块

fib.py

```
def fiber1(n):
    if n <= 2:
        return 1
    return fiber1(n-1)+fiber1(n-2)
def fiber2(n):
    a = [1]*n
    for i in range(2, n):
        a[i] = a[i-1]+a[i-2]
    return a[n-1]
```

main.py

```
import fib as fb

print(fb.fiber1(10))
print(fb.fiber2(9))


from fib import *
print(fiber1(10))
print(fiber2(9))
```

模块与包

- Python 第三方包的使用
  - 首先确保本地环境有相关包

    pip install Numpy

    conda install Numpy
  - 和使用模块差不多的用法

```
import sklearn
model = sklearn.linear_model.LogisticRegression()
model.fit([[0, 0], [1, 1], [2, 2]], [0, 1, 2])
print(model.coef_)
```

```
from sklearn import linear_model
model = linear_model.LogisticRegression()
```

```
from sklearn import linear_model as lm
model = lm.LogisticRegression()
```

```
from sklearn.linear_model import *
model = LogisticRegression()
```

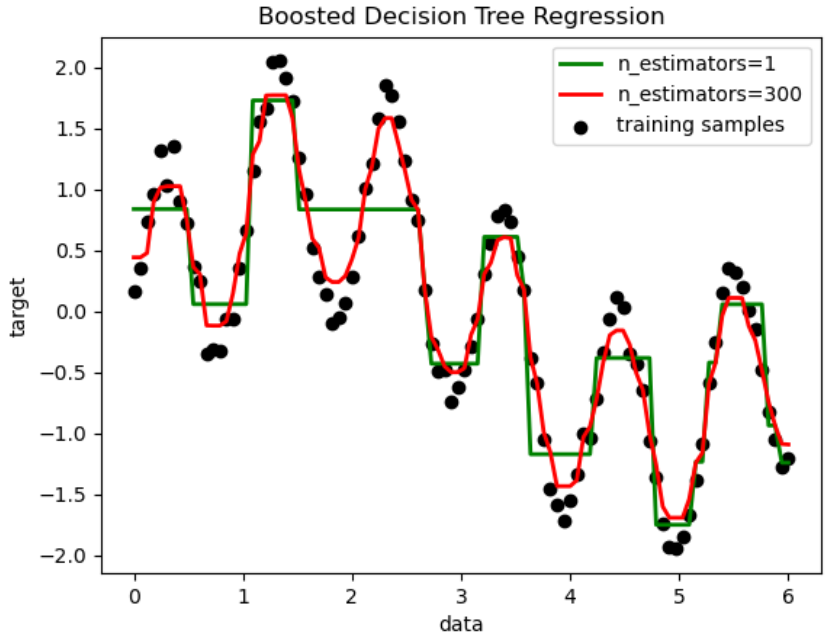
模块与包

- 自定义包
  - lib 文件夹
    - fib.py
  - main.py

```
import lib.fib as fb
print(fb.fiber1(10))
print(fb.fiber2(9))
```

数据科学常用的包与模块

- NumPy
  - 支持向量化运算的科学计算库
- Matplotlib
  - 功能强大的数据可视化库
- scikit-learn (sklearn)
  - Python 机器学习库



NumPy 是什么

- 如题，如何对一维，二维，三维序列，每个元素加1？
  [1,2,3,4,5]
- 循环？ 1层？2层？3层？N层
- 随着循环的嵌套，程序会越来越慢，逻辑越复杂

- NumPy 可以完美解决这个问题，

```
import numpy as np
a = np.array([1,2,3,4,5])
a = a+1
```

- 优点：经过C语言底层算法优化，速度快。向量化运算，逻辑清晰，编程简单
- 支持大量的线性代数运算

NumPy 的创建

- 向量或矩阵的创建:

  a = np.array([2,3,6])

  b = np.arange(5)

  c = np.linspace(0, 2*np.pi, 5)

  d = np.array([[11, 12, 13, 14, 15], [16, 17, 18, 19, 20], [21, 22, 23, 24, 25], [26, 27, 28 ,29, 30], [31, 32, 33, 34, 35]])

  e = np.zeros((2,3))

  f = np.ones((3,1))

  g = np.random.rand(3,2)

  h = np.random.randint(0,10,(3,2))

NumPy 的使用

- 获取向量的值与属性
  - 直接print()
  - 切片 [:,2],[1:3,2:5],[::2,::2] 等 (支持多维切片)
  - 形状与维度
    - a.shape
    - a.ndim

NumPy 的使用

- Numpy的数学运算

  +,-,*,/,//,%,**

  <<,>>,&,^,|,~

  ==, >,<,>=,<=,!=

- 线代运算 a.dot(b), np.cross(a,b),

  np.linalg.norm(a), np.outer(a,b)

- 函数的向量化运算:

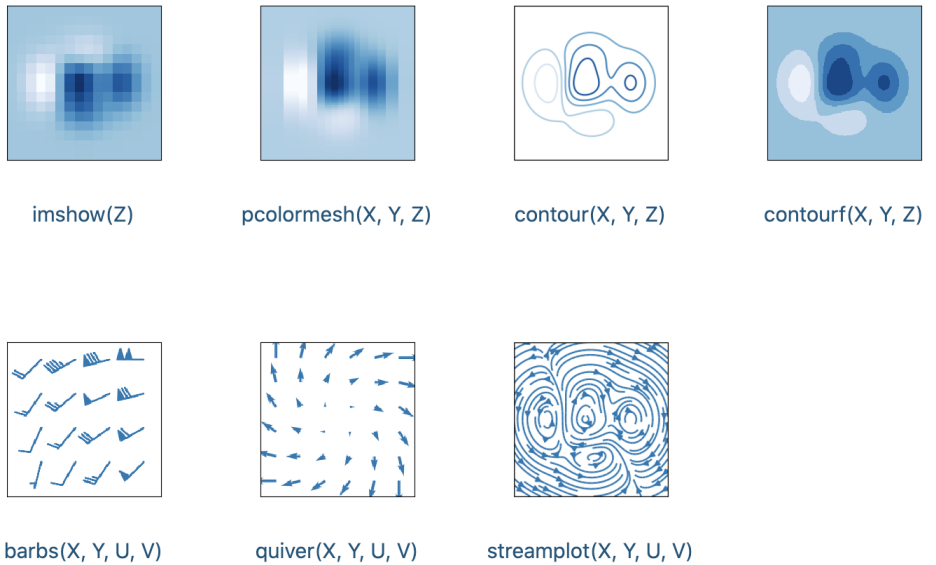
  np.sin(), np.cos(), np.tan() …

NumPy 的使用

- Numpy的其他操作
  - max, min, sum
  - 布尔屏蔽
    - a = np.arange(0,100,10)
    - print(a[a>=50])
  - Where函数
    - a = np.arange(0, 100, 10)
    - b = np.where(a < 50)
    - c = np.where(a >= 50)[0]
    - print(b) # >>>(array([0, 1, 2, 3, 4]),)
    - print(c) # >>>[5 6 7 8 9]

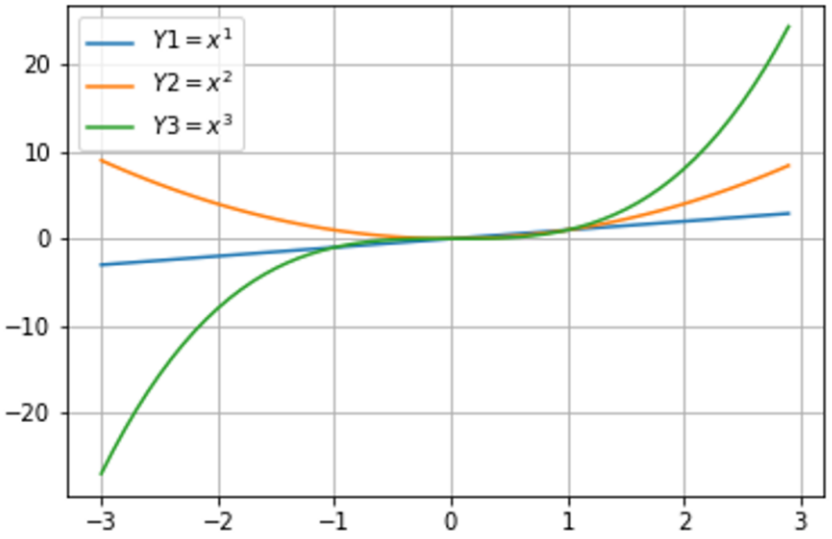
Matplotlib 介绍

- 曲线图 plot()
- 散点图 scatter()
- 直方图 bar()
- 显示图像 imshow()
- 三维图
- 等高线图 contour()
- ...



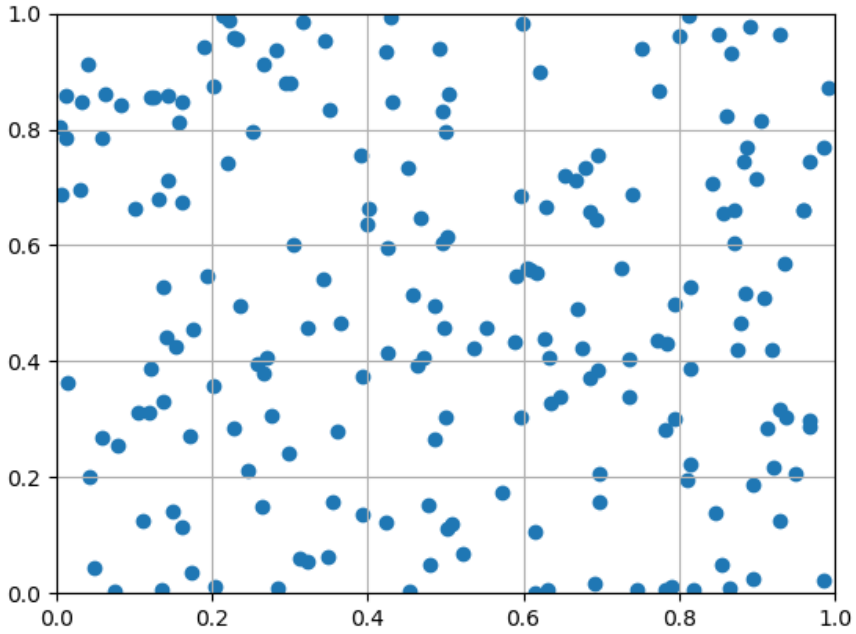
Matplotlib 介绍-曲线图

```
import numpy as np
import matplotlib.pyplot as plt
X = np.arange(-3,3,0.1)
Y_1 = X**1
Y_2 = X**2
Y_3 = X**3
plt.figure()
plt.plot(X,Y_1,label = '$Y1 = x^1$')
plt.plot(X,Y_2,label = '$Y2 = x^2$')
plt.plot(X,Y_3, label = '$Y3 = x^3$')
plt.legend()
plt.grid()
plt.savefig('power function')
plt.show()
```



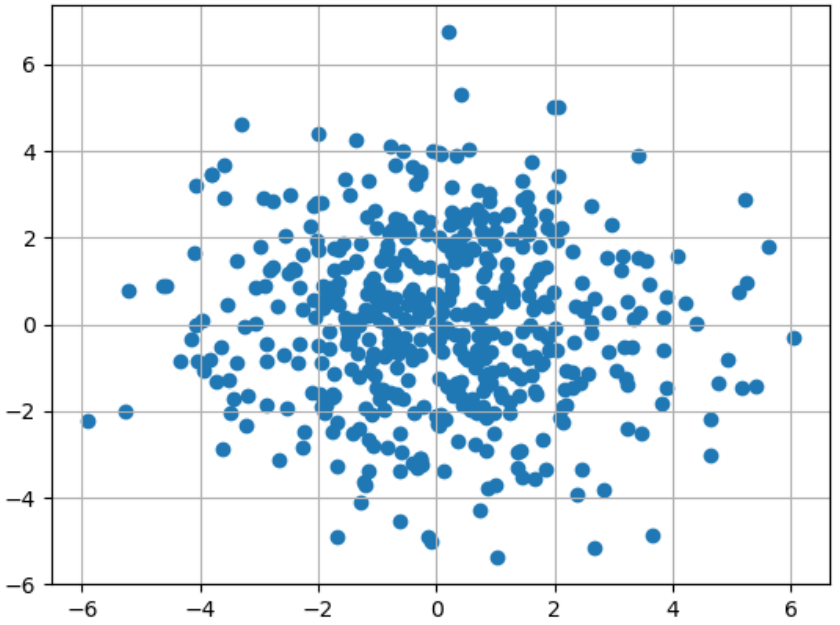
Matplotlib 介绍-散点图

```
import numpy as np
import matplotlib.pyplot as plt


data = np.random.random((200, 2))
x, y = data[:, 0], data[:, 1]
plt.scatter(x, y)
plt.grid()
plt.xlim(0, 1)
plt.ylim(0, 1)
plt.savefig('scatter.png')
plt.show()
```



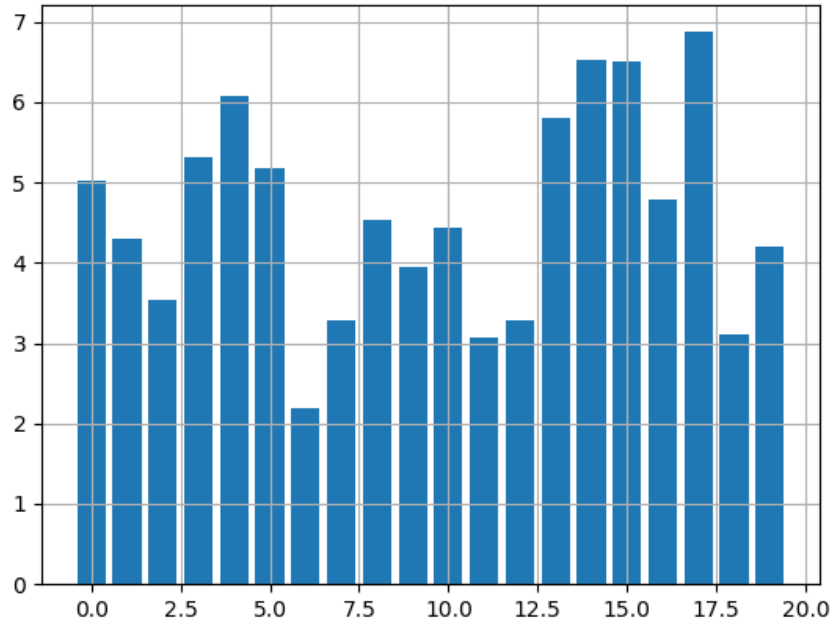
Matplotlib 介绍-散点图

```
import numpy as np
import matplotlib.pyplot as plt

x = np.random.normal(0, 2, 500)
y = np.random.normal(0, 2, 500)

plt.scatter(x, y)
plt.grid()
plt.savefig('scatter.png')
plt.show()
```



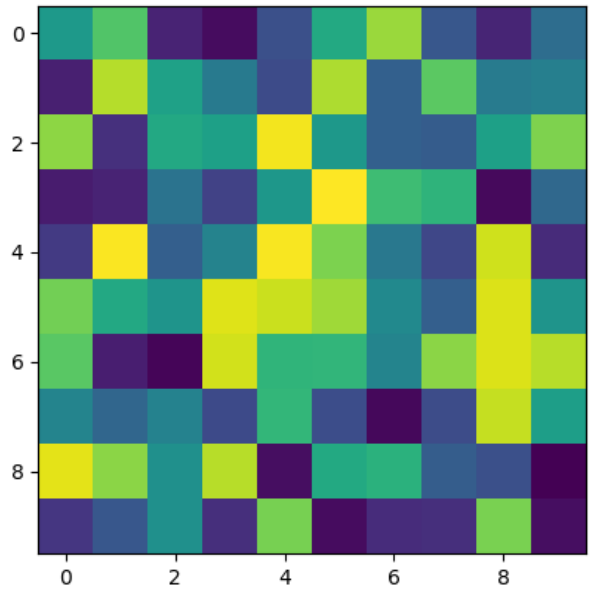
Matplotlib 介绍-直方图

```
import numpy as np
import matplotlib.pyplot as plt
x = np.arange(0, 8)
y = np.random.uniform(2, 7, len(x))
plt.bar(x, y)
plt.grid()
plt.savefig('bar.png')
plt.show()
```



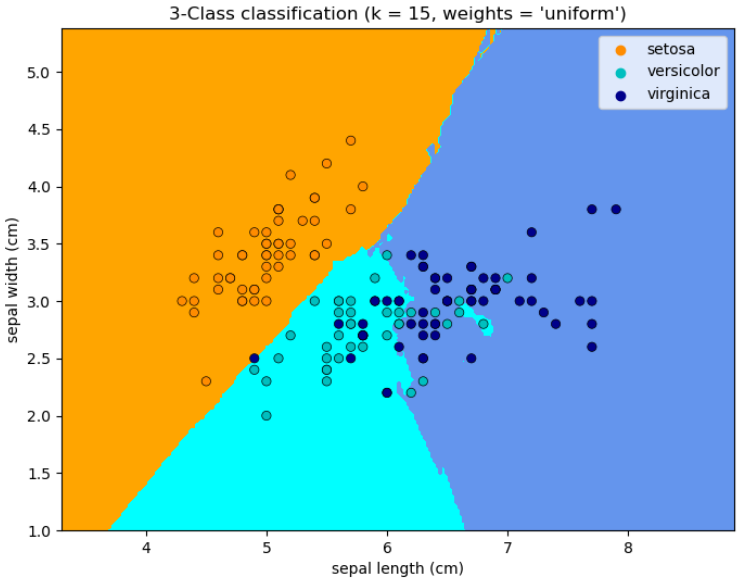
Matplotlib 介绍-显示图像

```
import numpy as np
import matplotlib.pyplot as plt


x = np.random.random((10, 10))
plt.imshow(x)
plt.savefig('image.png')
plt.show()
```



Sklearn 介绍

- Sklearn 是学术界以及工业界最常用的机器学习工具
- 能够帮助我们简化模型训练，验证，测试，部署的过程
- 支持监督算法，无监督算法，模型选择，数据预处理，模型持久化等功能



Sklearn 介绍

- 非常强大，我们会在下一章，用Numpy, Matplotlib, Sklearn 解决 iris 与 MNIST 数据分类问题