Python 学习笔记

Yusong

目录

1 Basic grammar **1**
1.1 type . . . 1
1.2 condition statements . . . 1
1.3 iteration . . . 1
1.4 function . . . 2
1.4.1 定义函数 . . . 2
1.4.2 调用函数 . . . 2
1.4.3 可变参数 . . . 3
1.4.4 静态类型函数 . . . 3
1.4.5 变量的作用域 . . . 4
1.4.6 High-order function . . . 4
1.4.7 self-reference . . . 6
1.4.8 function currying . . . 6
1.4.9 匿名函数 . . . 7
1.5 input and output . . . 7
1.5.1 打开关闭文件 . . . 7
1.5.2 读取写入文件 . . . 8

2 Container **8**
2.1 list . . . 8
2.2 tuple . . . 9
2.3 set . . . 9
2.4 dict . . . 10
2.5 String . . . 10
2.5.1 编码解码 . . . 10
2.5.2 常用操作 . . . 11
2.5.3 格式化字符串 . . . 11

3 regular expression **13**
3.1 grammar . . . 13
3.1.1 元字符 . . . 13
3.1.2 限定符 . . . 14
3.1.3 brackets . . . 14
3.1.4 notice . . . 14
3.2 re module . . . 14
3.2.1 匹配模式 . . . 15
3.2.2 匹配函数 . . . 15
3.2.3 替换字符串 . . . 15

3.2.4 切分字符串 16

4 Class **16**
4.1 定义类模板 16
4.1.1 基本结构 16
4.1.2 访问限制 17
4.1.3 属性 17
4.2 类的使用 18
4.2.1 创建类的实例 18
4.2.2 类成员的使用 18
4.3 继承 18

5 module **18**
5.1 import module 18

6 Maths and Statistics **19**
6.1 numpy 19
6.1.1 定义矩阵 19
6.1.2 特殊函数 19
6.1.3 矩阵运算 19
6.2 Pandas 20
6.2.1 Series 类型 20
6.2.2 DaraFrame 类型 20
6.2.3 读写数据 20
6.2.4 基本操作 21
6.2.5 统计操作 21

7 debug **21**
7.1 doctest 21
7.2 assert 22

8 visualizations **22**
8.1 matplotlib 22
8.1.1 画布预处理 22
8.1.2 基本图表 23
8.1.3 极坐标图表 24
8.1.4 三维图表 25
8.2 wordcloud 25

9 web spider **26**
9.1 基本操作 26
9.1.1 网络请求 26
9.1.2 请求 headers 处理 27

1 Basic grammar

1.1 type

Listing 1.1: variable type

```
>>> str = "python"
>>> dir(str) #获取相关用法
>>> type(str) #获取变量类型
>>> help(str) #获取 guide
```

Listing 1.2: basic functions

```
print(round(3.75, 1)) #四舍五入保留一位小数
float_number = 3.3; int_number = int(float_number)
```

1.2 condition statements

Listing 1.3: condition statement

```
def absolute_value(x):
  """Return the absolute value of x."""
  if x < 0:
    return -x
  elif x == 0:
    return 0
  else:
    return x

def absolute_value(x):
  return -x if x < 0 else x
```

- False values in Python: False, 0, ' ', None
- True values in Python: Anything else

1.3 iteration

Listing 1.4: iteration

```
i, total = 0, 0
while i < 3:
  total += i
  i += 1
```

```
        for i in range(3):
          total += i

```

1.4 function

函数用于处理需要多次重复运行的同一任务。python 的函数默认返回 None。

1.4.1 定义函数

Listing 1.5: definition of functions

```
        def my_func(string1,string2):
          print(string1)
          print(string2)

        def my_func1(string1='Hello',string2): #可以指定默认值
          ''' 功能:打招呼
              string1:默认hello
              string2:对象名字 '''
          print(string1)
          print(string2)
          return 1

        def my_info(*args, **kwargs): #这里 *args 会创建一个元组，kwargs 会创建一个字典
          print(args)
          print(kwargs)

```

1.4.2 调用函数

以下是几种正确的调用方法

Listing 1.6: use of function

```
        my_func('Hello','world')
        str1='Hello';str2='world'
        my_func(str1,str2)
        my_func(string1=str1,string2=str2) #写明形式参数名称为关键字参数
        my_func(str1,string2=str2) #允许前面的形参不写名称，后面的写明
        my_func(string2=str2,string1=str1) #写明形式参数名时可以交换顺序
        info_value = ['Math', 'Art']; dic_value = {'name': 'John', 'age': 22}
        my_info('Math', 'Art', name='John', age=22)
        my_info(*info_value, **dic_value)

```

以下是几种错误的调用方法

Listing 1.7: mistakes in using functions

```
my_func(string1=str1,str2) #前面写明形参名称，后面必须写
my_func(string2=str2,str1)

```

1.4.3 可变参数

可变参数是 Python 特有的设计，可变参数也被称为不定长参数，即传入函数中的实际参数可以是 0 个、1 个或多个。

Listing 1.8: variable parameters

```
def greet(*names): #这里会将输入接受并放在一个元组中
  print("Hello",end=' ')
  for item in names:
    print(item,end=' ')

if __name__=='__main__':
  greet('Tom','Jerry')

  param=['Tom','Jerry'] #也可以调用列表
  greet(*param) #这里不能写形参名

```

另一种方法会将输入以“形参名: 变量名”的形式存为一个字典。

Listing 1.9: other variable parameters

```
def greet(**names): #输入转为字典
  print('Hello')
  for key,value in names.items():
    if value=='male':
      print('Mr',end=' ')
    else:
      print('Miss',end=' ')
    print(key)

if __name__=='__main__':
  greet(Tom='male',Jerry='female') #直接调用

  dict={'Tom':'male','Jerry':'female'} #也可以用字典调用
  greet(**dict) #这里也不能用关键字参数

```

1.4.4 静态类型函数

Python 的函数默认是动态类型的，可以接受所有类型的输入。当然也可以将函数定为静态类型的 (和 C++ 类似)。

Listing 1.10: static functions

```
def square(number:int|float)->int|float:
  return num**2

```

1.4.5 变量的作用域

- 在函数内的定义的变量是局部变量，外部不能调用。
- 函数外部定义的变量是全局变量，可以在任意位置调用该变量。
- 函数内定义的变量可以通过 global 关键字定为全局变量。
- environment 就是 frame 的顺序，当调用某变量时，会从里向外逐层寻找该变量。
- 在一个函数中，一个变量名只能始终为一个全局变量，或者始终是一个局部变量。
- 在内部函数中的变量名前加上 nonlocal，则内部函数的该变量与外部函数的变量是同一个。

Listing 1.11: local and global

```
message='Hello'
def my_func():
  global message
  cnt = 0
  def in_func():
    nonlocal cnt
  message='World'
  cnt++
print(message) #输出'World'

```

1.4.6 High-order function

High-order function 是一类返回函数的函数，用于处理同类型的任务。

Listing 1.12: High-order function

```
def make_adder(n):
  """Return a function that takes K and return K + N.

  >>> add_three = make_adder(3)
  >>> add_three(4)
  """
  def adder(k):
    return k + n
  return adder
```

```
10
11          add_three = make_adder(3)
12          print(make_adder(100)(4))
13
```

High-order function 在音频领域经常被使用，用于制作波形。

Listing 1.13: mario

```
1           from wave import open
2           from struct import Struct
3           from math import floor
4
5           frame_rate = 11025
6
7           def encode(x):
8               i = int((1 << 14) * x)
9               return Struct('h').pack(i)
10
11          def play(sampler, name='song.wav', seconds=2):
12              out = open(name, 'wb')
13              out.setnchannels(1)
14              out.setsampwidth(2)
15              out.setframerate(frame_rate)
16              t = 0
17              while t < seconds * frame_rate:
18                  sample = sampler(t)
19                  out.writeframes(encode(sample))
20                  t += 1
21              out.close()
22
23          def tri(frequency, amplitude=0.3):
24              period = frame_rate // frequency
25              def sampler(t):
26                  saw_wave = t / period - floor(t / period + 0.5)
27                  tri_wave = 2 * abs(2 * saw_wave) - 1
28                  return amplitude * tri_wave
29              return sampler
30
31          c_freq, e_freq, g_freq = 261.63, 329.63, 392.00
32
33          def both(f, g):
34              return lambda t: f(t) + g(t)
35
36          def note(f, start, end, fade=0.01):
37              def sampler(t):
38                  seconds = t / frame_rate
39                  if seconds < start:
40                      return 0
41                  elif seconds > end:
42                      return 0
43                  elif seconds > end - fade:
44                      return (end - seconds) / fade * f(t)
45                  elif seconds < start + fade:
46                      return (seconds - start) / fade * f(t)
47                  else:
```

```
                    return f(t)
                return sampler

            c, e, g = tri(c_freq), tri(e_freq), tri(g_freq)
            g_low = tri(g_freq / 2)
            z = 0
            song = note(e, z, z + 1/8)
            z += 1/8
            song = both(song, note(e, z, z + 1/8))
            z += 1/4
            song = both(song, note(e, z, z + 1/8))
            z += 1/4
            song = both(song, note(c, z, z + 1/8))
            z += 1/8
            song = both(song, note(e, z, z + 1/8))
            z += 1/4
            song = both(song, note(g, z, z + 1/8))
            z += 1/2
            song = both(song, note(g_low, z, z + 1/8))
            z += 1/2

            play(song)

```

1.4.7 self-reference

self-reference function 会返回自身或者与自身相关的函数，这样可以递归的执行不定长度的任务。

Listing 1.14: self-reference

```
        def print_all(x):
          print(x)
          return print_all
        print_all(1)(2)(3) #这里会全部输出来

```

1.4.8 function currying

function currying 将一个接受多个参数的函数分解为一系列每个只接受一个参数的函数，即一个函数链。

Listing 1.15: function currying

```
          def curry2(f):
            def g(x):
              def h(y):
                return f(x, y)
              return h
            return g

```

```
from operator import add
m = curry2(add)
add_three = m(3)
print(add_three(2))

```

1.4.9 匿名函数

Python 的匿名函数也就是 lambda 表达式。匿名函数可以是另一个函数的参数，如 sort 函数的 key 变量。

- 只有 def 得到的函数会有一个本征名，在 shell 中输入变量名即可看到这一差异。
- def 函数可以有多步运行过程，而 lambda 表达式只能有一个表达式。

语法为 result=lambda [arg1[arg2,···,argn]]:expression

Listing 1.16: lambda function

```
result = lambda r:math.pi*r*r
area = (lambda r: math.pi * r * r)(3)
print(result(5))

def area(r):
  return r * r * math.pi
result = area

```

1.5 input and output

1.5.1 打开关闭文件

Listing 1.17: open and close files

```
file=open(<filename>[,mode]) #打开文件
file.close() #关闭文件

with open(<filename>[,mode]) as file #用 with 语句打开文件可以避免文件打开错误引起的异常
  with-body #执行 with 语句后的一些相关操作语句，可以用 pass 代替

```

Table 1.1: Methods of Opening a File

符号	说明	注意
r rb r+ rb+	只读模式，指针放在开头 二进制只读模式 可以读取，也可以从文件的开头覆盖 二进制读写模式	文件必须存在
w wb w+ wb+	只写模式 二进制只写模式 只读模式打开文件并清空 二进制只读模式并清空文件	若文件存在，则覆盖，否则创建新文件
a ab a+ ab+	追加模式 二进制追加模式 追加且可读 (注意指针位置) 二进制追加且可读	若文件存在，则指针放在末尾，否则创建文件

1.5.2 读取写入文件

Listing 1.18: read and write files

```
file.write(string) #将 string 变量的内容写入文件
string=file.read(9) #读取从指针位置开始的 9 格字符
line=file.readline() #读出一行
lines=file.readlines() #读出所有行，返回一个字符串列表
file.seek(offset[,whence]) #移动指针位置
```

- offset: 移动字符个数
- whence: 指定从什么位置开始计算，0 表示开头，1 表示当前位置，2 表示文件末尾 (只能在二进制模式下使用)

2 Container

2.1 list

Listing 2.1: basic operations of list

```
#创建列表
empty_list = []; empty_list = list()
student1=["Hermione","Harry","Ron"]
```

```
        student2=["Draco","Padma"]
        student_score=list(range(10))
        #对列表元素进行操作
        student="Potter"
        student1.append(student) #在列表的末尾追加
        student1.insert(2, student) #在指定 index 处插入
        student2.extend(student1) #列表的拼接，这样可以比 + 更快
        del student[-1] #删除最后一个元素
        student_popped=student1.pop(1) #删除指定 index 处的元素，默认删除最后一个
        student_popped=student1.remove('Padma') #寻找第一个指定元素并删除，找不到会报错
        student_reversed=student1.reverse() #反转元素顺序
        seq_Ron=student.index("Ron") #查找第一个匹配的元素并输出，找不到会报错
        #列表拼接、扩展、排序、遍历
        student1=student1+student2
        student3=sorted(student1)
        student3=sorted(student1, key=str.lower()) #不区分大小写排序
        student1.sort(reverse=True) #True stands for down
        if "Hermione" in student1:
          print("Hermione is in student1")
        for a in student1: #遍历列表
        for index, item in enumerate(student1):
        for index, student in enumerate(student1, start=1): #这里 index 可以从 1 开始编号
        students = ', '.join(student1) #将 list 中的元素合并成一个字符串，以逗号为分隔符
        #列表统计
        number=student1.count("Padma") #统计元素出现的数量
        ind=student2.index("Padma") #找到元素首次出现的位置
        min_one, max_one = min(student_score), max(student_score)
        total=sum(student_score) #列表求和

```

2.2 tuple

元组是不可变的序列，但可以重新赋值

Listing 2.2: basic operation of tuple

```
        #元组创建
        empty_tuple = (); empty_tuple = tuple() #创建空元组
        a=tuple(rage(10,20,3)); a=(); a=(1,2,3)

```

2.3 set

Python 的 set 是无序集合，这与 C++ 不同。

Listing 2.3: basic operation of set

```
        students=[
          {"name":"Hermione","house":"Gryffindor"},
          {"name":"Harry","house":"Gryffindor"},
          {"name":"Ron","house":"Gryffindor"},
          {"name":"Draco","house":"Slytherin"},
```

```
  {"name":"Padma","house":"Ravenclaw"},
]
houses=set() #创建空集合，注意 houses= 会创建一个字典
houses.pop() #删除第一个元素，注意顺序是随机的
houses.clear() #清空集合
for student in students:
  houses.add(student["house"])
for house in sorted(houses):
  print(house)
if "Slytherin" in houses:
  print("Slytherin is in houses.")

```

2.4 dict

Listing 2.4: basic operation of dictionary

```
#创建字典
students1={
  "Harry":"Gryffindor",
  "Ron":"Gryffindor",
  "Draco":"Slytherin",
  "Padma":"Ravenclaw",
}
name=["Harry","Ron","Draco","Padma"]
house=["Gryffindor","Gryffindor","Slytherin","Ravenclaw"]
student1=dict(zip(name,sign)) #zip 是映射，这是通过列表创建字典的方法
student1=dict((("Harry","Gryffindor"),("Ron","Gryffindor"),
    ("Draco","Slytherin"),("Padma","Ravenclaw"))) #这一句与上一句等效
student1=dict(Harry="Gryffindor",Ron="Gryffindor",Draco="Slytherin",Padma="Ravenclaw")
student_empty=dict.fromkeys(name) #创建值为空的字典
#访问，删除，加入，遍历
ron_house=student1["Ron"] #访问字典，找不到会报错
ron_house=student1.get("Ron") #访问字典，找不到会返回 None
david_house=student1.get("David", "Not Found") #访问字典，找不到返回 Not Found
ron_house=student1.pop("Ron") #删除元素并返回值
del student1["Ron"] #删除元素
student1["Hermione"]="Gryffindor" #加入元素
student1.update({"Hermione":"Gryffindor"}) #合并两字典
for key,value in student1.items() #遍历字典
for key in student1.keys(); for key in student1 #遍历键数组
for value in student1.values() #遍历值数组

```

2.5 String

2.5.1 编码解码

Listing 2.5: create a string

```
name="Harry"; text='Hello world!'
byte=name.encode('GBK') #编码，返回编码的 16 进制值
name1=byte.decode('GBK') #解码

```

2.5.2 常用操作

Listing 2.6: basic operations of string

```
length=len(name) #返回字符串长度
size=len(name.encode()) #返回内存大小
text.split(' ') #切分字符串
strnew=string.join(house) #合并字符串
sub='ab'
str_replace=sub.replace('a', 'c') #字符串替换
num=string.count(sub) #检索子字符串出现次数
start=string.find(sub) #子字符串首次出现的索引，若没有出现，返回-1
str_lower=str.lower() #转为小写，不会改变原字符串
str_upper=str.upper() #转为大写
str_strip=str.strip(['@']) #删去字符串左右两侧的' ',\t,\r,\n 等（默认），以及'@'(可选)
str_lstrip=str.lstrip(['@']) #删去字符串左侧的特定字符
str_rstrip=str.rstrip(['@']) #删去字符串右侧的特定字符

```

2.5.3 格式化字符串

格式化字符串也就是先制定一个模板，并在模板中以占位符为变量留下位置。一种实现方式是与 C 语言相似的占位符，更适合 python 的做法是使用 format 函数。

Listing 2.7: formatting string1

```
'格式化字符串：%[-][+][0][m][.n]'val

```

可选项如下

- -: 左对齐，正数前面无符号，负数前面有符号
- +: 右对齐，正数前面加正号，负数前面加负号
- 0: 右对齐，正数前面无符号，负数前面加负号，用 0 补足位数
- m: 占有宽度
- .n: 小数位数

Table 2.1: placeholder

格式字符	说明	格式字符	说明
%s	字符串	%c	单个字符
%x	十六进制整数	%o	八进制整数
%f %F	浮点数	%d %i	十进制整数
%%	字符%	%e	指数 (基底为 e)

format 函数使用模板

Listing 2.8: format string

```
'{[index][:[[fill]align][sign][#][width][.precision][type]]}'
```

- fill: 可选，填充字符
- align: 可选，< 左对齐，> 右对齐，^ 内容居中
- sign: 可选，+-, 用法与占位符选项相同
- #: 可选，进制前缀显示
- width: 可选，字段宽度
- .precision: 可选，小数位数
- type: 可选，指定类型

Table 2.2: type options

格式字符	说明	格式字符	说明
S	字符串类型	b	十进制整数转为二进制
D	十进制整数	o	十进制整数转为八进制
C	按 ASCII 码转为字符	x X	十进制整数转为十六进制
e E	转为科学计数法	f F	转为浮点数，默认 6 位小数
g G	自动切换	%	转为百分数

下面给出一些例子

Listing 2.9: examples of format string

```
number, name, url = 7, "百度", "baidu"
template='编号:{:0>9s}\t公司名称:{:s}\t官网:http://www.{:s}.com'
context=template.format(number, name, url)
context_f=f'编号:{number}\t公司名称:{name}\t官网:http://www.{url}.com'

```

3 regular expression

3.1 grammar

正则表达式由元字符，限定符，选择字符，排除字符等组成。

3.1.1 元字符

元字符中的‘.’‘$’‘^’, 限定符中的‘?’‘+’‘*’ 以及‘\’ 若需要匹配，则应该使用转义字符。

Table 3.1: meta-character

元字符	含义
.	匹配除换行符以外的任意字符 [^\n\r]
\w	匹配字母、数字、下划线或汉字
\s	匹配任意空白字符 ([\f\n\r\t\v])
\d	匹配数字
^	匹配字符串的开始
\b	匹配单词的开始或结束
$	匹配字符串的结束

3.1.2 限定符

Table 3.2: qualifier

限定符	含义
?	匹配 0-1 次
+	匹配至少 1 次, 会贪婪匹配
*	匹配 0 次或多次, 会贪婪匹配
+? *?	匹配规则同上，但是非贪婪
{n}	匹配 n 次
{n,}	匹配至少 n 次
{n,m}	匹配至少 n 次，至多 m 次
|	匹配左右其一

3.1.3 brackets

- 方括号表示字符集合，如 [aeiou] 匹配元音字符，[a-z] 匹配小写字符。[\u4e00-\u9fa5] 匹配汉字。
- 方括号内的 ^ 表示排除字符。
- 小括号可以改变限定符的作用范围, 如 (thir|four)th 相当于 thirth|fourth, (\.[0-9]{1,3}){3} 会将括号中的内容重复三次。
- 小括号还可以基于匹配模式从字符串中提取子字符串，组成一个元组。

3.1.4 notice

Python 中使用正则表达式需要将部分\转义, 由于需要转义的\可能很多，可以使用模式字符串，即在字符串前加上 r 或 R，例如:r‘\bm\w*\b’。

3.2 re module

Python 中的 re 模块可以通过正则表达式处理字符串。

3.2.1 匹配模式

Table 3.3: matching pattern

标志	含义
A ASCII	只匹配 ASCII 范围内的字符
I IGNORECASE	不区分大小写
M MULTILINE	将 ^ 和 $ 用于每一行的开头和结尾
S DOTALL	. 匹配所有字符，包括换行符
X VERBOSE	忽略未转义的空格和注释

3.2.2 匹配函数

Listing 3.1: string matching

```
import re #导入模块

pattern=r'mr_\w+' #模式字符串
string='MR_SHOP mr_shop' #待匹配字符串

#match 方法可以从字符串开始处开始匹配，若匹配失败，返回 None
match=re.match(pattern,string,re.I) #不区分大小写匹配字符串，返回一个 match 对象
print(match.start());print(match.end()) #输出匹配字符串起始位置
print(match.span()) #输出匹配字符串起止位置元组
print(match.string) #输出匹配前的字符串，即 string
print(match.group()) #输出匹配的数据

#search 方法可以搜索字符串中第一个可以匹配的子串
match=re.search(pattern,string,re.I) #不区分大小写匹配字符串，返回一个 match 对象

#findall 方法可以搜索字符串中所有可以匹配的子串
match=re.findall(pattern,string,re.I) #不区分大小写匹配字符串，返回一个 match 对象
print(match) #输出匹配子串的列表

```

3.2.3 替换字符串

sub 方法可以实现 vim 中批量搜索并替换字符串的操作。
模板为:re.sub(pattern,repl,string,count,flags)

- pattern: 模式字符串
- repl: 替换后的子字符串
- string: 原始字符串

- count: 可选，最大替换次数，默认为 0，表示替换所有的匹配
- flags: 可选，见Table 3.3

Listing 3.2: string substitution

```
import re

pattern=r'1[34578]\d{9}' #模式字符串
string='电话号码是:13611111111'
result=re.sub(pattern,'1xxxxxxxxxx',string)

```

3.2.4 切分字符串

re.split 方法可以根据正则表达式切分字符串，匹配正则表达式的子串将被当作分隔符，并将分割结果以列表的形式返回。

模板为:re.split(pattern,string,[maxsplit],[flags])

- pattern: 模式字符串
- string: 待切分的字符串
- maxsplit: 可选，最大拆分次数
- flags: 可选，见Table 3.3

Listing 3.3: string segmentation

```
import re

pattern=r'[?&]'
url='http://www.mingrisoft.com/login.jsp?username="mr"&pwd="mrsoft"'
result=re.split(pattern,url)
print(result)

```

4 Class

4.1 定义类模板

4.1.1 基本结构

```
class User:
  ''' 用户类 ''' #类的说明
  student_user='学生用户' #这里的变量是静态数据成员, 为整个类所有,Python
  teacher_user='教室用户'
  student_number=0
  teacher_number=0
  def __init__(self,name,number): #构造函数
    self.__name=name #这里名字设置为私有成员, 这里的变量是非静态成员, py
    self.number=number
  def __str__(self); #输出运算的重载
    print(self.number,' ',self.__name)
  @property
  def name(self);
    return self._name
  @name.setter
  def name(self,name)
    self._name=name
```

4.1.2 访问限制

Python 没有对属性和方法的访问权限进行限制。为了保证类内部某些属性不被外部访问，可以在属性或方法名前 (或前后) 加上双下划线。

- __foo__: 首尾双下划线表示定义特殊方法，一般是系统定义名字。
- __foo: 双下划线表示私有成员,只允许所在的类调用。

4.1.3 属性

Python 中，可以通过 @property 将一个方法转换为属性，转换后可以直接通过方法名调用该方法，无需添加 ()。这样做可以简化代码，也为属性添加安全保护,即添加了 @property 的属性是只读的 (这是由于 return 时经过了复制传递)。

```
class User:
  ''' 用户类 ''' #类的说明
  @property
  def name(self);
    return self.__name #这个name属性被设定为只读的
```

被 property 保护的属性也可以设置为在某种条件下可以被修改。

```
  @name.setter #这里name是属性名称
  def name(self,name)
```

```
        if len(name)<20:
            self.__name=name
```

4.2 类的使用

4.2.1 创建类的实例

```
student = User('Jerry',7) #注意如果构造函数无需额外参数，那也要写一个()
```

4.2.2 类成员的使用

```
print(student) #这里会调用__str__函数
```

4.3 继承

Python 的继承默认是公有继承。

```
class Student(User)
    def __init__(self,grade,name,number):
        super().__init__(name,number) #调用基类的构造函数
        self.__grade=grade
```

子类可以重写父类中的方法，这与 C++，Java 没什么区别，只是 Python 没有虚函数机制。

5 module

5.1 import module

Listing 5.1: import module

```
import math
import math as ma
from math import sqrt
from math import *
```

6 Maths and Statistics

6.1 numpy

6.1.1 定义矩阵

```
import numpy as np

a=np.array([0.1*i for i in range(100)])
a=np.array([[i+5*j for i in range(5)] for j in range(5)])
a=np.arange(25).reshape(5,5) #reshape可以重新规定矩阵型号
a=np.linspace(0,10,100) #第三个参数是生成的列表长度
a=np.arange(0,10,0.1) #第三个参数是步长
a=np.logspace(0.9.10) #生成10的0-9次幂
a=np.eye(3) #生成三维单位阵
a=np.diag([1,2,3,4,5]) #生成对角阵
a=np.random.rand(2,3) #生成2*3随机矩阵,0-1均匀分布
a=np.random.random((2,3)) #生成2*3随机矩阵, 用元组表示大小
a=np.random.randint(low,high,size=(2,3)) #生成2*3随机整数矩阵
```

6.1.2 特殊函数

```
X,Y=np.meshgrid(x,y) #将x,y扩展为一个矩阵
x,y=np.outer(x,y) #得到矩阵x^{t}y
x=np.append(x1,x2) #拼接array
a=X[1] #取出矩阵的一行
a=X[:,1] #取出矩阵的一列
X[:,0]=a #更改矩阵的一列
(n,m)=np.where(X>1) #查找满足条件的元素坐标
a=X[n,m] #取出满足条件的元素
l=np.argwhere(X>1) #n为坐标组成的二维array
```

6.1.3 矩阵运算

```
c=np.dot(x,y) #矩阵乘法
c=x*y #对应元素相乘
c=np.dot(a,np.linalg.inv(b)) #矩阵右除
c=np.dot(a,np.linalg.inv(a),(b)) #矩阵左除
c=np.transpose(a);c=a.T #矩阵转置
```

```
result=np.linalg.inv(a) #求逆矩阵
result=np.linalg.det(a) #求行列式
result=np.linalg.matrix_rank(a) #求矩阵的秩
matrix.sum(axis=0) #求和，1行0列
```

6.2 Pandas

6.2.1 Series 类型

Series 类型是一维数组，由 index 和 value 组成。

```
import pandas

data=['A','B','C']
index=['a','b','c']

series = pandas.Series(data) #默认index从0开始编号
series_with_index = pandas.Series(data,index=index) #规定index
print(series.index,series.value) #会输出数组
print(series_with_index['a']) #调用Series的元素
```

6.2.2 DaraFrame 类型

DataFrame 的每列的名称为键，每个键对应一个数组，这个数组为值。

```
import pandas

data = {'a':[1,2,3,4,5],'b':[6,7,8,9,10],'c':[11,12,13,14,15]}
index = ['A','B','C','D','E']
data_frame = pandas.DataFrame(data) #创建DataFrame对象,默认index从0开始
data_frame = pandas.DataFrame(data,index=index) #规定index
data_frame = pandas.DataFrame(data,columns=['a','b']) #指定列
print(data_frame)
```

6.2.3 读写数据

Pandas 模块可以将 csv 或 excel 文件转为 DataFrame 变量, 也可以将 DataFrame 变量写入 csv 或 excel 文件。

```
import pandas

data = pandas.read_csv(<filename>)
```

```
data = pandas.read_excel(<filename>)
data.to_csv(<new_filename>,columns=['A','B'],index=False) #不写入行索引
data.to_excel(<new_filename>,columns=['A','B'],index=False) #写入excel文
```

6.2.4 基本操作

```
data_frame['d']=[50,60,70,80,90] #增添数据

data_frame.drop([0,1],inplace=True) #按index删除，inplace表示对原数据删
data_frame.drop(labels='a',axis=1,inplace=True) #按column删除，axis=1表

data_frame['a'][1] = numpy.nan; data_frame['b']=[7,8,9,10,11] #修改数据
data_frame.a[1] = numpy.nan; data_frame.b = [7,8,9,10,11] #这与上面是等
```

6.2.5 统计操作

预处理部分

```
null_num = data_frame.isnull().sum() #统计空缺值数量，isnull在空缺值返回
not_null_num = data_frame.nornull().sum() #统计非空缺值数量
data_frame.dropna(axis,inplace=True) #删除包含空缺值的整行数据
data_frame.fillna(0,inplace=True) #修改空缺值
data_frame.fillna({'A':0,'B':1,'C':2},inplace=True) #每列空缺值用指定的
```

统计函数

```
average = data_frame.mean() #求每列的平均值，输出一个Series
score = data_frame.a+data_frame.b-data_frame.c #可以直接做向量运算
data_frame.sort_values(['a'],axis=0,ascending=False,inplace=True) #排序
```

7 debug

7.1 doctest

doctest 可以检查函数的输出，在代码注释样例中给出一组输入输出，若结果错误会报错。

Listing 7.1: doctest hello.py

```
from operator import floordiv, mod

def divide_exact(n, d):
```

```
            """Return the quotient and remainder of  dividing N by D
            >>> q, r = divide_exact(2013, 10)
            >>> q
            201
            >>> r
            2
            """
            return floordiv(n, d), mod(n, d)

```

7.2 assert

类似于 C++assert 断言，不需要导入库。

Listing 7.2: assert in python

```
        def area_square(r):
          assert r > 0, 'A length must be positive'
          return r * r

```

Listing 7.3: doctest bash

```
        python3 -m doctest hello.py
        **********************************************************************
        File "/mnt/d/Desktop/python/program file/test/test_basic/Pythonproject1/hello.py", line 8, in hello.divide_exact
        Failed example:
            r
        Expected:
            2
        Got:
            3
        **********************************************************************
        1 items had failures:
          1 of   3 in hello.divide_exact
        ***Test Failed*** 1 failures.

```

8 visualizations

8.1 matplotlib

8.1.1 画布预处理

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

```
plt.figure(figsize=(10,20),facecolor,edgecolor)
plt.title("title")
plt.xlabel("x");plt.ylabel("y")
plt.style.use("seaborn-v0_8")
plt.legend() #显示图例
plt.xticks(ticks=[2*i+1 for i in range(10)],labels=[2*i+1 for i in range
plt.xlim(2,22)
plt.grid(axis=both, #axis=x or y or False
        linestyle="dashed", #or dotted or dashdot
        color="#FFFFFF"
) #添加网格线
plt.axhline(5,color,linestyle,linewidth) #水平参考线
plt.axvline(10,color,linestyle,linewidth) #垂直参考线
plt.axhspan(5,7,color,linestyle,linewidth) #水平参考区域
plt.axvspan(10,12,color,linestyle,linewidth) #垂直参考区域
plt.annotate(text,xy=(5,10), #待注释点坐标
            xytext=(7,12), #注释文本位置
            color="#FFFFFF",fontsize=16,
            ha="center", #水平居中
            va="bottom", #垂直对齐
            arrowprops={"arrowstyle":"->", #or "-"
                        "color":"#FFFFFF"}
) #显示注释点
plt.text(7,12,text) #显示无箭头注释
```

8.1.2 基本图表

```
plt.plot(x,y,color="red",
        linestyle="dashed", #or dotted or dashdot
        linewidth=3,
        marker=".", #or "," "o" "+" "x"
        markersize=8,
        markerfacecolor="blue",
        markeredgecolor="cyan"
)
plt.bar(x,y,width,bottom=3, #柱形底部高度
        hatch="/") #or "1" "\" "\\" "//"
plt.barh(x,y)
plt.hist(x,bins) #直方图, bins可以是整数（条数）或列表
plt.scatter(x,y,s) #s is a list, which stands for the size of the dots
```

```
plt.pie(x,colors=["red","blue","yellow"],
        autopct=%1.1f%%, #整数部分一位，小数部分一位
        explode=[0,0.5,0], #将第二块拉出0.5
        shadow=True,labels=["一月","二月","三月"],
) #饼图
plt.pie(x,colors=["red","blue","yellow"],
        autopct=%1.1f%%, #整数部分一位，小数部分一位
        explode=[0,0.5,0], #将第二块拉出0.5
        shadow=True,labels=["一月","二月","三月"],
        radius=1.0,wedgeprops={"width":0.6} #内外圆半径
) #圆环图
plt.boxplot(x,showmeans=True, #显示均值
            flierprops={"marker":"o", #or "," "+" "x"
                        "markerfacecolor":"red",
                        "markeredgecolor":"black",
                        "markersize":8
            }, #异常点样式
            patch_artist=True, #自定义箱型
            boxprops={"facecolor":"red",
                      "edgecolor":"yellow"
            } #箱型样式
) #箱型图
plt.stackplot(x,y1,y2,y3,color=["red","yellow","blue"])
#面积图(可堆叠)
plt.errorbar(x,y,yerr=[lower_errors,upper_errors],
            ecolor=blue, #color of the errorbars
            elinewidth=3, #width of the errorbars
            capsize=2 #横杠大小
)
plt.imshow(x, #x是个二维列表
          cmap=plt.cm.cool #设置颜色
) #绘制热力图
plt.colorbar() #显示图例
```

8.1.3 极坐标图表

```
plt.polar(theta,r) #雷达图
plt.thetagrid(angles,labels) #角刻度标签
plt.rgids(radii,rotation,labels) #r方向刻度标签
```

```
ax=plt.axes(polar=True) #建立极坐标画布
ax.bar(x=theta,height=data,width=0.4,color="rainbow") #绘制南丁格玫瑰图
ax.bar(x=theta,height=100,width=0.4,color="white") #绘制中心空白
ax.text(angle,height,text) #添加注释
ax,grid(False)
plt.thetagrids(angles=[],labels=[]) #刻度标签
plt.rgrids(radii=[20],rotation,labels=['20'])
```

8.1.4 三维图表

```
from mpl_tookits.mplot3d import Axes3D
fig=plt.figure()
ax1=plt.axes(projection="3d")
ax1.scatter3D(x,y,z,cmap="blue")
ax1.plot3D(x,y,z,"gray")
ax1.plot_surface(X,Y,Z,rstride=0.1,cstride=0.1) #步长越短越清晰
ax1.contour(X,Y,Z,zdir='x',offset=-3,cmap="cold") #绘制等高线，投影在x=
ax1.bar3d(X,Y,height,width,depth,Z,color="red",shade=True) #绘制柱状图,
```

8.2 wordcloud

```
import matplotlib.pyplot as plt
import wordcloud as wc

text_data = """
Python is a popular programming language.
It is widely used for web development, data analysis, and artificial intel
Word clouds are fun visualizations of text data.
Generate a word cloud using the wordcloud module.
"""

# 生成词云对象
wordcloud = wc.WordCloud(width=800, height=400, background_color='white')

# 显示词云图
plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.show()
```

9 web spider

9.1 基本操作

Python 自带的 urllib 和 urllib3 模块可以实现一些网络爬虫的常用操作。

9.1.1 网络请求

利用 request 模块可以实现 get 请求方式获取网页内容。

```
import urllib.request #网络请求子模块

response = urllib.request.urlopen('http://www.baidu.com') #打开网页
html = response.read() #读取网页代码
print(html)
```

也可以实现 post 请求方式获取网页内容。

```
import urllib.parse #url解析和引用模块
import urllib.request

#使用urlencode方法对数据进行处理,并将处理后的数据设置为utf-8编码
data = bytes(urllib.parse.urlencode({'word':'hello'}), encoding='utf8')
response = urllib.request.urlopen('http://httpbin.org/post', data=data)
html = response.read()
print(html)
```

urllib3 是一个更强大的 Python 库。

```
import urllib3

http = urllib3.PoolManager() #创建对象, 用于处理连接和安全等细节
response = http.request('GET','https://www.baidu.com/') #连接网站
print(response.data) #输出读取内容
```

也可以用 post 方法连接网页。

```
import urllib3

http = urllib3.PoolManager() #创建对象, 用于处理连接和安全等细节
response = http.request('POST', 'http://httpbin.org/post', fields={'word
print(response.data) #输出读取内容
```

另外有个更加人性化的第三方库 requests。

```
import requests

data={'word':'hello'}
response = requests.get('http://www.baidu.com',params=data) #get方法访问
response = requests.post('http://httpbin.org/post',data=data) #post方法
print(response.contents) #以字节流形式输出网页源码
print(response.text) #以文本形式输出网页源码
```

9.1.2 请求 **headers** 处理

请求 headers 处理是为了绕开网站的反爬设置。

```
import requests

url = 'http://www.bilibili.com/'
headers = {'User-Agent':'Mozilla/5.0 (Windows NT 10.0; Win64; x64) Apple
#这里User-Agent的内容要从网络监视器中复制过来
response = requests.get(url,headers=headers)
print(response.content)
```