Python_Practice

Python's basic syntax by HaleyKwok

Python

- 解释型语言，开发过程中没有了编译这个环节。
- 交互式语言，可以在一个 Python 提示符 >>> 后直接执行代码。
- 面向对象语言，支持面向对象的风格或代码封装在对象的编程技术。
- 支持广泛的应用程序开发，从文字处理到 WWW 浏览器再到游戏。

基本数据类型

六个标准的数据类型

- non-immutable 不可变数据（3 个）：Number（数字）、String（字符串）、Tuple（元组）；
- mutable 可变数据（3 个）：List（列表）、Dictionary（字典）、Set（集合）。

1. Number

1. type()用来查询变量所指的对象类型：

```
a, b, c, d = 20, 5.5, True, 4+3j
print(type(a), type(b), type(c), type(d))

#return
<class 'int'> <class 'float'> <class 'bool'> <class 'complex'>
```

2. 使用del语句删除一些数字对象的引用：

```
var1 = 1
var2 = 10

del var1, var2
```

3. Python 支持三种不同的数值类型：

- 整型(int) 是正或负整数，不带小数点。布尔(bool)是整型的子类型
- 浮点型(float) 由整数部分与小数部分组成
- 复数((complex)) - 复数由实数部分和虚数部分构成，可以用a + bj,或者complex(a,b)表示， 复数的实部a和虚部b都是浮点型

4. 对数据内置的类型进行转换：

- int(x) 将x转换为一个整数。
- float(x) 将x转换到一个浮点数。
- complex(x) 将x转换到一个复数，实数部分为 x，虚数部分为 0。
- complex(x, y) 将 x 和 y 转换到一个复数，实数部分为 x，虚数部分为 y。x 和 y 是数字表达式。

```
a = 1.0
int(a)
#return
1
```

5. 数字运算、数字函数、随机数函数、三角函数、数字常量...

2. String

string、list 和 tuple 都属于 sequence（序列）

1. 访问字符串中的值
   indexing 索取: 变量[]
   slicing 切片: 变量[头下标:尾下标]； [start: end: step]

```
t = ['a','b','c','d','e']
print(t[0])
# ['a']
print(t[1:3])
# ['b','c']
print(t[:4])
# ['b','c','d']
print(t[3:])
# ['d','e']
print(t[:])
# ['a','b','c','d','e']
print(t[1:4:2])
# ['b','d']
```

如果第三个参数为负数表示逆向读取，以下实例用于翻转字符串：

```
t = ['a','b','c','d','e']
print(t[::-1])
# ['e', 'd', 'c', 'b', 'a']
```

2. 字符串更新
   截取string的一部分并与其他字段拼接：

```
str1 = 'Hello, world!'
print("NEW:", str1[:6] + 'Admin!') #, is included as the string

#return
NEW: Hello, Admin!
```

3. 转义字符 (\n \f \t \r..)
4. 运算符
5. 格式化 (%d %s...)

```
print("I am %s, I am $d years old." %('Haley', 10))
#return
I am Haley, I am 10 years old.
```

或者：

```
name = 'Haley'
year = 10
print(f"I am {name}, I am {year} years old.")
#return
I am Haley, I am 10 years old.
```

3. Tuple ()

元组（tuple）与list类似，但元组的元素不能修改。元组写在小括号 () 里，元素之间用逗号隔开：

```
tuple = ('abcd', 786, 2.23, 'csdn', 70.2)
othertuple = (123, 'csdn')

print(othertuple * 2) #加号 + 是列表连接运算符
print(tuple + othertuple) #星号 * 是重复操作

#return
(123, 'csdn'. 123, 'csdn')
('abcd', 786, 2.23, 'csdn', 70.2, 123, 'csdn')
```

1. 元组与string类似，可以被索引且下标索引从0开始，-1 为从末尾开始的位置。也可以进行截取。
2. 只包含一个元素时，需要在元素后面添加逗号，否则括号会被当作运算符使用：

```
tup1 = (50) #int
type(tup1)

tup1 = (50,) #tuple
type(tup1)

#return
<class 'int'>
<class 'tuple'>
```

3. 修改元组
   虽然tuple的元素不可改变，但它可以包含可变的对象，比如list列表，或对元组进行连接组合：

```
tuple = ('abcd', 123, 999, ['name','number'])
```

```
tup1 = (12, 34.56)
tup2 = ('abc', 'xyz')

tup3 = tup1 + tup2
print(tup3)

#return
(12, 34.56, 'abc', 'xyz')
```

4. 删除元组
   使用del语句来删除整个元组：删除后，输出变量会有异常信息：

```
tup = (12, 34.56)
del tup
print(tup)
#return
NameError: name 'tup' is not defined
```

5. 运算符
6. 内置函数（len..）****

---

1. List []

List（列表） 是 Python 中使用最频繁的数据类型，可以实现大多数集合类的数据结构

1. 和string一样，列表同样可以被indexing和slicing
2. list as sequences，列表中的元素是可以改变的：

```
list = ['abcd', 999, 9.99, 'google', 99.9]
otherlist = [123, 'instagram']

print(otherlist * 2) #加号 + 是列表连接运算符
print(list + otherlist) #星号 * 是重复操作

# return
[123, 'instagram, 123, 'instagram']
['abcd', 999, 9.99, 'google', 99.9, 123, 'instagram']
```

3. 更新或添加列表

对列表的数据项进行修改或更新，使用 append() 方法来添加列表项：

```
#update element
list = ['abcd', 999, 9.99, 'google', 99.9]
print(list[1])
list[1] = 888
print('NEW:', list[1])
#return
999
NEW: 888

#add new element
list1 = ['Google', 'Taobao', 'MS']
list1.append('Apple')
print('NEW:', list1)
#return
['Google', 'Taobao', 'MS', 'Apple']
```

4. 删除列表元素
   使用 del 语句来删除列表的的元素：

```
#delete
list = ['abcd', 999, 9.99, 'google', 99.9]
del list[1]
print('NEW:', list)
#return
NEW: ['abcd', 9.99, 'google', 99.9]
```

5. Nested list嵌套列表
   使用嵌套列表即在列表里创建其它列表：

```
list1 = ['a', 'b', 'c']
list2 = [1, 2, 3]
list = [list1, list2]

list
list[0]
list[0][1]
#return
[['a', 'b', 'c'], [1, 2, 3]]
['a', 'b', 'c']
'b'
```

6. 列表函数&方法(len, append, insert, pop, remove, reverse, sort...)

数据结构（data structure）

1. 列表的大部分方法

```
a = [66.25, 333, 333, 1, 1234.5]
print(a.count(333), a.count(66.25), a.count('x'))

a.insert(2, -1)
a.append(333)
print(a)

a.index(333)

 a.remove(333)
print(a)

a.reverse()
print(a)

a.sort()
print(a)

# return
2 1 0
[66.25, 333, -1, 333, 1, 1234.5]
[66.25, 333, -1, 333, 1, 1234.5, 333]
[66.25, 333, -1, 333, 1, 1234.5, 333]
[66.25, -1, 333, 1, 1234.5, 333]
[333, 1234.5, 1, 333, -1, 66.25]
[-1, 1, 66.25, 333, 333, 1234.5]
```

3. 将列表当做堆**栈**使用 Using Lists as Stacks：
   最先**进**入的元素最后一个被**释**放（后**进**先出）。用 append() 方法可以把一个元素添加到堆**栈顶**。用不指定索引的 pop() 方法可以把一个元素从堆**栈顶释**放出来

```
stack = [3,4,5]
stack.append(6)
stack.append(7)

print(stack) #insert 6 7
stack.pop()
print(stack) #remove 7

# return
[3, 4, 5, 6, 7]
[3, 4, 5, 6]
```

4. 将列表当作队列使用 Using Lists as Sequences：

```
from collections import deque
queue = deque(["Eric", "John", "Michael"])
queue.append("Terry")           # Terry arrives
queue.append("Graham")          # Graham arrives
queue.popleft()                 # The first to arrive now leaves

queue.popleft()                 # The second to arrive now leaves
print(queue)                    # Remaining queue in order of arrival

# return
deque(['Michael', 'Terry', 'Graham'])
```

---

2. Dictionary (key: value)

字典（dictionary）是另一个非常有用的内置数据类型。
list是有序的对象集合，dict是无序的对象集合。

1. indexing

```
dict['one'] = "hello, world!"
dict[2] = "hw"
print(dict['one'])
print(dict[2])

# return
hello, world!
hw
```

2. **键**(key)必须使用不可**变**类型；在同一个字典中，**键**(key)必须是唯一的。

```
d = {key1 : value1, key2 : value2, key3 : value3 }
```

字典是一种映射类型，字典用 { } **标识**，它是一个无序的 **键**(key) : **值**(value) 的集合

```
tinydict = {'name': 'google', 'code':1, 'site': 'www.google.com'}

print(tinydict)
print(tinydict.keys())
print(tinydict.values())

# return
{'name': 'google', 'code': 1, 'site': 'www.google.com'}
dict_keys(['name', 'code', 'site'])
dict_values(['google', 1, 'www.google.com'])
```

3. 修改字典

   向字典添加新内容的方法是增加新的**键/值对**，修改或**删**除已有**键/值对**如下**实**例：

```
tinydict = {'Name': 'CSDN', 'Age': 23, 'Class': 'First'}
tinydict['Age'] = 24           #update
tinydict['School'] = 'Google'  #add

print(tinydict['Age'])
print(tinydict['School'] = 'Google')
# return
24
Google
```

4. **删**除字典元素

   显示**删**除一个字典用del命令：

```
tinydict = {'Name': 'CSDN', 'Age': 23, 'Class': 'First'}
del tinydict['Name'] #delect key:Name

tinydict.clear() #clean all
del tinydict #clean dict

#return
Traceback (most recent call last):
File "/runoob-test/test.py", line 9, in <module>
print ("tinydict['Age']: ", tinydict['Age'])
NameError: name 'tinydict' is not defined
```

5. 字典**键**的特性

- 不允**许**同一个**键**出**现**两次。**创**建**时**如果同一个**键**被**赋值**两次，后一个**值**会被**记**住
- **键**必**须**不可**变**，可以用数字，字符串或元**组**充当，而用list就不行

6. 字典内置函数&方法(len, str, type...)
7. 无限极嵌套

```
district={
    'Kowloon City':{
        'Whampoa':['A','B','C','D','E'],
        'Mong Kok':['F','G','H','I'],
        'Tsim Sha Tsui':['J','K','L',],
        'Sham Shui Po':['M','N','O'],
        'Yau Tsim Mong':['P','Q','R']
    },
    'Hong Kong Island':{
        'the Central':['S','T','U','W','X'],
        'Causeway Bay':['Y','Z','A'],
        'Cyberport':['B','C','D','E']
    }
}
for i in district['Kowloon City']:
        print(i)

for i in district['Kowloon City']['Whampoa']:
        print(i)

# return
Whampoa
Mong Kok
Tsim Sha Tsui
Sham Shui Po
Yau Tsim Mong

A
B
C
D
E
```

3. Set {}

集合（set）是由一个或数个形态各异的大小整体组成的，构成集合的事物或对象称作元素或是成员，进行成员关系测试和删除重复元素。

1. 创建集合：大括号 { } 或者 set() 函数
   创建一个空集合：用 set() 而不是 { }，因为 { } 是用来创建一个空字典。

```
sites = {'Google', 'Taobao', 'MS', 'Facebook', 'Zhihu', 'Baidu'}

print(sites)

if 'MS' in sites:
        print('MS in set')
else:
        print('MS not in set')

a = set('abracadabra')
b = set('alacazam')

print(a)
print(a-b) #差集
print(a|b) #并集
print(a&b) #交集
print(a^b) #不同时存在的元素

#return
{'Zhihu', 'Baidu', 'Taobao', 'MS', 'Google', 'Facebook'}
MS in set
{'b', 'c', 'a', 'r', 'd'}
{'r', 'b', 'd'}
{'b', 'c', 'a', 'z', 'm', 'r', 'l', 'd'}
{'c', 'a'}
```

2. 添加元素

add()

```
addset = set(('Google', 'Taobao', 'MS'))
addset.add('Meta')
print(addset)

# return
{'MS', 'Google', 'Meta', 'Taobao'}
```

update()

```
updateset = set(('Google', 'Taobao', 'MS'))
updateset.update({1,2,3})
print(updateset)

updateset = set(('Google', 'Taobao', 'MS'))
updateset.update([1,4], [1,5,6])
print(updateset)

# return
{'Google', 1, 2, 3, 'MS', 'Taobao'}
{1, 4, 'MS', 5, 6, 'Google', 'Taobao'}
```

3. 移除元素

不存在的元素会**报错**：

remove()

```
removeset = set(('Google', 'Taobao', 'MS'))
removeset.remove('MS')
print(removeset)

removeset = set(('Google', 'Taobao', 'MS'))
removeset.remove('Meta')
print(removeset)

# return
{'Taobao', 'Google'}

Traceback (most recent call last):
  File "/Users/haleyk/Documents/Python_Practice/Python_Practice/set.py", line 20, in <modu
    removeset.remove('Meta')
KeyError: 'Meta'
```

不存在的元素不会**报错**：

discard()

```
discardset = set(('Google', 'Taobao', 'MS'))
discardset.discard('Meta') #没有这个element
print(discardset)

# return
{'Google', 'Taobao', 'MS'}
```

4. 清空集合

   clear()
5. 判断元素是否在集合中存在

```
thisset = set(('Google', 'Taobao', 'MS'))
'Meta'
```

注释

单行注释以 # 开头，例如：

```
# 这是一个注释
print("Hello, World!")
```

多行注释多行注释用三个单引号 ''' 或者三个双引号 """ 将注释括起来，例如：

```
'''
单引号
单引号
'''

"""
双引号
双引号
"""
```

运算符（operators）

• 算术运算符 (+ - * / % ** //)

• 比较（关系）运算符 (== != > < >= <=)

• 赋值运算符 (= += -= *= /= %= **= //= :=)

• 逻辑运算符 (and or not)

• 位运算符 (& ^ | ~ << >>)

> "<<" 左移动运算符：运算数的各二进位全部左移若干位，由"<<"右边的数指定移动的位数，高位丢弃，低位补0。
> a << 2 输出结果 240 ， 二进制解释： 1111 0000

> ">>" 右移动运算符：把">>"左边的运算数的各二进位全部右移若干位，">>"右边的数指定移动的位数。
> a >> 2 输出结果 15 ， 二进制解释： 0000 1111

• 成员运算符 (in, not in)
• 身份运算符 (is, is not)
> is 和 == 的区别： is 用于判断两个变量引用对象是否为同一个，后者用于判断引用变量的值是否相等
• 运算符优先级

条件控制（if..elif..else）

语法：

```
if condition1:
        statement1
elif condition2:
        statement2
else condition3:
        statement3
```

```
number = 7
guess = -1
print('Guess a number!')
while guess != number:
        guess = int(input('Please input your guessing number:'))
        if guess == number:
                print('Congrats!')
        elif guess < number:
                print('The number is too small...')
        elif guess > number:
                print('The number is too large...')
```

```
print('II. Number Guessing Game')
print('Number Guessing Game!')

a = 1
i = 0
while a != 20:
   a = int (input ('Please enter the number you guessed:'))
   i += 1
   if a == 20
      if i < 3:
         print('Good, you guessed it so quickly!')
      else:
         print('Finally, you guessed correctly, congratulations!)
   elif a < 20:
      print("You guessed a smaller number, don't be discouraged, keep trying!")
   else:
      print("You guessed a bigger number, don't be discouraged, keep trying!")
```

循环语句（for & while）

- while：

syntax：

```
while condition:
        statements
```

```
n = 100

sum = 0
counter = 1
while counter <= n:
        sum = sum + counter
        counter += 1
print(f'{sum}')

#return
5050
```

- while else：

syntax：

```
while expr:
    statement(s)
else:
    additional_statement(s)
```

```
count = 0
while count < 5:
        print(count, " is smaller than 5")
        count = count + 1
else:
        print(count, " is larger or equal to 5")

# return
0  is smaller than 5
1  is smaller than 5
2  is smaller than 5
3  is smaller than 5
4  is smaller than 5
5  is larger or equal to 5
```

- for：

syntax：

```
for variable in sequence:
        statements
else:
        statements
```

```
n = 0
sum = 0
for n in range(0,101):
    sum += n
print(sum)
# return
5050
```

```
sites = ['Baidu', 'Google', 'Meta', 'Taobao']
for site in sites:
        if site == 'Meta':
                print('Facebook')
                break #break the sentence and output Finish!
        print('looping ' + site)
else:
        print('no looping data')
print('Finish!')

# return
looping Baidu
looping Google
Facebook
Finish!
```

循环嵌套：

```
for i in range(1,6):
        for j in range(1, i+1):
                print("*",end='')
        print('\r')

# return
*
**
***
****
*****
```

```
#外边一层循环控制行数
#i是行数
i=1
while i<=9:
     #里面一层循环控制每一行中的列数
     j=1
     while j<=i:
          multiple =j*i
          print("%d*%d=%d"%(j,i,multiple), end="  ")
          j+=1
     print("")
     i+=1
# return
1*1=1
1*2=2  2*2=4
1*3=3  2*3=6  3*3=9
1*4=4  2*4=8  3*4=12  4*4=16
1*5=5  2*5=10  3*5=15  4*5=20  5*5=25
1*6=6  2*6=12  3*6=18  4*6=24  5*6=30  6*6=36
1*7=7  2*7=14  3*7=21  4*7=28  5*7=35  6*7=42  7*7=49
1*8=8  2*8=16  3*8=24  4*8=32  5*8=40  6*8=48  7*8=56  8*8=64
1*9=9  2*9=18  3*9=27  4*9=36  5*9=45  6*9=54  7*9=63  8*9=72  9*9=81
```

- break/continue：

break 语句可以跳出 for 和 while 的循环体。如果你从 for 或 while 循环中终止，任何对应的循环 else 块将不执行；
continue 语句被用来告诉 Python 跳过当前循环块中的剩余语句，然后继续进行下一轮循环。

syntax：

```
while expr:
        statement
        statement
        break
        statement
        statement
        continue
        statement
        statement
statment
```

```
sites = ['Baidu', 'Google', 'Meta', 'Taobao']
for site in sites:
        if len(site) != 4:
                print(f'hello, {site}')
                continue
        if site == 'Taobao':
                break
print('done!')

# return
hello, Baidu
hello, Google
hello, Taobao
Done!
```

- pass：
  空语句，是为了保持程序结构的完整性；pass 不做任何事情，一般用做占位语句

syntax：

```
class MyEmptyClass:
        pass
```

```
for letter in 'Google':
        if letter == 'o':
                pass
                print('pass')
        print('the letter is', letter)
print('Goodbye!')

# return
the letter is G
pass
the letter is o
pass
the letter is o
the letter is g
the letter is l
the letter is e
Goodbye!
```

Iterator and Generator

1. iterator是一个可以记住遍历的位置的对象，有两个基本的方法：iter() 和 next()。

```
list = [1,2,3,4]
it = iter(list)  #create iteration
print(next(it))  # output the next elemnt
print(next(it))
# return
1
2
```

迭代器对象可以使用常规for语句进行遍历：

```
list = [1,2,3,4]
it = iter(list)
for x in it:
        print(x, end=" ")
print()

# return
1 2 3 4
```

也可以使用 next() 函数：

```
import sys

list = [1,2,3,4]
it = iter(list)

while True:
        try:
                print(next(it))
        except StopIteration:
                sys.exit()
# return
1
2
3
4
```

2. 创建一个迭代器
   把一个类作为一个迭代器使用需要在类中实现两个方法 **iter**() 与 **next**() 。

```
class MyNumbers:
        def __iter__(self):
                self.a = 1
                return self

        def __next__(self):
                x = self.a
                self.a += 1
                return x

myclass = MyNumbers()
myiter = iter(myclass)

print(next(myiter))
print(next(myiter))
print(next(myiter))
print(next(myiter))
print(next(myiter))

# return
1
2
3
4
5
```

3. 停止StopIteration

```python
class MyNumbers:
        def __iter__(self):
                self.a = 1
                return self

        def __next__(self):
            if self.a <= 20:
                x = self.a
                self.a += 1
                return x
            else:
                raise StopIteration

myclass = MyNumbers()
myiter = iter(myclass)

for x in myiter:
        print(x)

# return
1
2
3
4
5
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
```

3. generator
   生成器是一个返回迭代器的函数，只能用于迭代操作，生成器就是一个迭代器；在调用生成器运行的过程中，每次遇到 yield 时函数会暂停并保存当前所有的运行信息，返回 yield 的值, 并在下一次执行 next() 方法时从当前位置继续运行。
   调用一个生成器函数，返回的是一个迭代器对象。

斐波那契数列（Fibonacci sequence），又称黄金分割数列，指的是这样一个数列：1、1、2、3、5、8、13、21、34、……在数学上，斐波那契数列以如下被以递推的方法定义：F(0)=0，F(1)=1, F(n)=F(n - 1)+F(n - 2)（n ≥ 2，n ∈ N*）在现代物理、准晶体结构、化学等领域，斐波纳契数列都有直接的应用。

```
import sys
def fibonacci(n):
        a,b,counter = 0, 1, 0
        while True:
                if (counter > n):
                        return
                yield a
                a,b = b, a+b
                counter += 1
f = fibonacci(10)

while True:
        try:
                print(next(f), end=" ")
        except StopIteration:
                sys.exit()
# return
0 1 1 2 3 5 8 13 21 34 55
```

函数（def）

1. 用户自定义函数：

```
def 函数名（参数列表）:
    函数体
```

```
def max (a,b):
        if a > b:
                return a
        else:
                return b
a = 4
b = 5
print(max(a, b))

# return
5
```

2. 调用

```
def printme(str):
        print(str)
        return

printme('123')
printme('abc')

# return
123
abc
```

3. 可更改(mutable)与不可更改(immutable)对象

- 不可变类型：类似 C++ 的值传递，func(a)传递的只是 a 的值，没有影响 a 对象本身。如果在 fun(a) 内部修改 a 的值，则是新生成一个 a 的对象。
- 可变类型：类似 C++ 的引用传递，func(la)将 la 真正的传过去，修改后 fun 外部的 la 也会受影响
- python 中一切都是对象，不能说值传递还是引用传递，应该说传不可变对象和传可变对象。

4. 参数(parameters)
   - 必需参数
   - 关键字参数
   - 默认参数

• 不定长参数

一个函数能处理比当初声明时更多的参数：
加了星号 * 的参数会以元组(tuple)的形式导入，存放所有未命名的变量参数

syntax：

```
def functionname([formal_args,] *var_args_tuple ):
   "函数_文档字符串"
   function_suite
   return [expression]
```

```
def printinfo(arg1, *vartuple):
        print(arg1)
        print(vartuple)

printinfo(70,60,50)
# return
70
(60,50)
```

还有一种就是参数带两个星号 **，加了两个星号的参数会以字典(dict)的形式导入：
syntax：

```
def functionname([formal_args,] **var_args_dict ):
   "函数_文档字符串"
   function_suite
   return [expression]
```

```
def printinfo(arg1, **vartuple):
        print(arg1)
        print(vartuple)

printinfo(70,a=60,b=50)

# return
70
{'a': 60, 'b': 50}
```

5. lambda 匿名函数
   只是一个表达式，函数体比 def 简单很多；lambda 函数拥有自己的命名空间，且不

能访问自己参数列表之外或全局命名空间里的参数

syntax：

```
lambda [arg1 [,arg2,.....argn]]:expression
```

```
x = lambda a : a + 10
print(x(5))

# return
15
```

```
sum = lambda arg1, arg2: arg1 + arg2

print(sum(10, 20))
print(sum(20, 20))

# return
30
40
```

遍历（traversal）

1. items()

```
knights = {'gallahad': 'the pure', 'robin': 'the brave'}
for k in knights.items():
    print(k)

# return
('gallahad', 'the pure')
('robin', 'the brave')
```

2. enumerate()

```
for i,v in enumerate(['sin', 'cos', 'tan']):
        print(i,v)

# return
0 sin
1 cos
2 tan
```

3. range()

```
list = [2, 3, 4]
for i in range(len(list)):
    print i,list[i]

# return
```

4. iter()

模块（module）

用 import 或者 from...import 来导入相应的模块（module）

将整个模块导入，格式为： import somemodule

```
import time
```

将整个模块导入，再命名，格式为： import somemodule as sm

```
import numpy as np
```

从某个模块中导入某个函数，格式为： from somemodule import somefunction

```
from matplotlib import pyplot as plt
```

从某个模块中导入多个函数，格式为： from somemodule import firstfunc, secondfunc, thirdfunc

```
from sympy import symbols, Eq, solve
```

将某个模块中的全部函数导入，格式为： from somemodule import *

```
from numpy import *
```

name 属性

一个模块被另一个程序第一次引入时，其主程序将运行。如果我们想在模块被引入时，模块中的某一程序块不执行，我们可以用__name__属性来使该程序块仅在该模块自身运行时执行：

```
# file name: module.py

if __name__ == '__main__':
   print('the program is running')
else:
   print('another module')

# return
$ python3 module.py
the program is running
$ python3
>>> import module
another module
```

输入和输出

f.write()

f.read()

f.readline()

f.readlines()

f.write()

f.close()

...

错误和异常 (try,except,else,finally,raise,assert)

1. 异常处理 try/except

如果没有异常发生，忽略 except 子句，try 子句执行后结束：

```
while True:
        try:
                x = int(input('Please input a number.'))
                break
        except ValueError:
                print('Error!')
```

2. try/except...else
   else 子句将在 try 子句没有发生任何异常的时候执行：

- 使用except而不带异常类型

  syntax：

```
try:
    正常的操作
   ......................
except:
    发生异常，执行这块代码
   ......................
else:
    如果没有异常执行这块代码
```

```
#! /usr/bin/python
# -*- coding: UTF-8 -*-
import sys
for arg in sys.argv[1:]:
try:
    fh = open("testfile", "w")
    fh.write("This is a test file for testing for exceptions!!!")
except IOError:
    print("Error: file not found or failed to read file")
else:
    print("Content was written to file successfully")
    fh.close()
```

- 使用except而带多种异常类型

  syntax：

```
try:
    正常的操作
   ......................
except(Exception1[, Exception2[,...ExceptionN]]):
   发生以上多个异常中的一个，执行这块代码
   ......................
else:
    如果没有异常执行这块代码
```

- 处理子句中调用的函数（甚至间接调用的函数）里抛出的异常

```
def this_fails():
        x = 1/0
try:
        this_fails()
except ZeroDivisionError as err:
        print('Handling run-time error:', err)

# return
Handling run-time error: division by zero
```

3. try-finally
   无论是否发生异常都将执行最后的代码

   syntax：

```
try:
<语句>
finally:
<语句>    #退出try时总会执行
raise
```

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-

try:
    fh = open("testfile", "w")
    fh.write("testing")
finally:
    print("Error!")

# return
Error!
```

4. raise
   raise语句可以触发异常

   syntax：

```
raise [Exception [, args [, traceback]]]
```

```
def functionName(level):
    if level < 1:
        raise Exception("Invalid level!", level) # after raising an error, no execution af

# return
```

命名空间和作用域

局部的命名空间去 -> 全局命名空间 -> 内置命名空间

```
#global
var1 = 5
def some_func():
        # local
        var2 = 6
        def some_inner_func():
                # built-in
                var3 = 7
```

```
total = 0
def sum(arg1, arg2):
        total = arg1 + arg2
        print('local:', total)
        return total

sum(10,20)
print('global:', total)

# return
local: 30
global: 0
```

global：

```
a = 10
def test():
        global a
        b = a + 1
        a = b
        print(b)
test() #local
print(a) #global

# return
11
11
```

面向对象（OOP）

类定义对象可以包含任意数量和类型的数据：

```
class ClassName:
        statement1
        ..
        statement2
```

类有一个名为 **init**() 的特殊方法（构造方法），该方法在类实例化时会自动调用

empCount 变量是一个类变量，它的值将在这个类的所有实例之间共享。你可以在内部类或外部类使用 Employee.empCount 访问。

self 代表类的实例，self 在定义类的方法时是必须有的，虽然在调用时不必传入相应的参数

```
class Employee:
   # 所有员工的基类
   empCount = 0

   def __init__(self, name, salary):
      self.name = name
      self.salary = salary
      Employee.empCount += 1

   def displayCount(self):
     print("Total Employee %d" % Employee.empCount)

   def displayEmployee(self):
      print("Name : ", self.name,  ", Salary: ", self.salary)

# 创建 Employee 类的第一个对象
emp1 = Employee("Zara", 2000)
# 创建 Employee 类的第二个对象
emp2 = Employee("Manni", 5000)

# 使用点号 . 来访问对象的属性。使用如下类的名称访问类变量：
emp1.displayEmployee()
emp2.displayEmployee()
print("Total Employee %d" % Employee.empCount)

# return
Name :  Zara , Salary:  2000
Name :  Manni , Salary:  5000
Total Employee 2
```

使用以下函数的方式来访问属性：

syntax：

```
getattr(obj, name[, default]) : 访问对象的属性。
hasattr(obj,name) : 检查是否存在一个属性。
setattr(obj,name,value) : 设置一个属性。如果属性不存在，会创建一个新属性。
delattr(obj, name) : 删除属性。
```

继承（inheritance）：

```
class 派生类名(基类名)
    ...
```

```
class SubClassName (ParentClass1[, ParentClass2, ...]):
    ...
```

```
class A:        # 定义类 A
.....

class B:         # 定义类 B
.....

class C(A, B):   # 继承类 A 和 B
.....
```

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-

class Parent:        # 定义父类
   parentAttr = 100
   def __init__(self):
      print "调用父类构造函数"

   def parentMethod(self):
      print '调用父类方法'

   def setAttr(self, attr):
      Parent.parentAttr = attr

   def getAttr(self):
      print "父类属性 :", Parent.parentAttr

class Child(Parent): # 定义子类
   def __init__(self):
      print "调用子类构造方法"

   def childMethod(self):
      print '调用子类方法'

c = Child()          # 实例化子类
c.childMethod()      # 调用子类的方法
c.parentMethod()     # 调用父类方法
c.setAttr(200)       # 再次调用父类的方法 - 设置属性值
c.getAttr()          # 再次调用父类的方法 - 获取属性值

# return
调用子类构造方法
调用子类方法
调用父类方法
父类属性 : 200
```

运算符重载（overloading）：

```python
#!/usr/bin/python

class Vector:
   def __init__(self, a, b):
      self.a = a
      self.b = b

   def __str__(self):
      return 'Vector (%d, %d)' % (self.a, self.b)

   def __add__(self, other):
      return Vector(self.a + other.a, self.b + other.b)

v1 = Vector(2,10)
v2 = Vector(5,-2)
print(v1 + v2)

# return
Vector (7, 8)
```

错题本

```
indian = ['samosa', 'daal', 'naan']
chinese = ['egg role', 'pot sticker', 'fried rice']
italian = ['pizza', 'pasta', 'risotta']

foodstyle = str(input('enter the food style:'))

'''
list name
'''
if foodstyle == 'indian':
    print(indian)
elif foodstyle == 'chinese':
    print(chinese)
elif foodstyle == 'italian':
    print(italian)



# return
# enter the food style: chinese
# ['egg role', 'pot sticker', 'fried rice']

print('-----------------------------------------------------------')

indian = ['samosa', 'daal', 'naan']
chinese = ['egg role', 'pot sticker', 'fried rice']
italian = ['pizza', 'pasta', 'risotta']

dish = str(input('enter the dish:'))

'''
element
'''
if dish in indian:
    print('indian')

elif dish in chinese:
    print('chinese')
elif dish in italian:
    print('italian')
else:
    print('Error! Please enter again!')

print('-----------------------------------------------------------')

# return
```

```
# enter the dish:samosa
# indian
```

Reference 参考资料

Python 基础教程

Zelle, J. M. (2004). *Python programming: an introduction to computer science.* Franklin, Beedle & Associates, Inc..

Codebasics by Dhaval Patel