作业1 标程

```
# O(n log n) Solution
words = input().split()
word_count = {}
for w in words:
    word_count[w] = 1 + word_count.get(w, 0)
for w, c in word_count.items():
    print(w, c)

# O(n²) Solution
words = input().split()
for w in set(words):
    print(w, words.count(w))
```

Python编程基础

王启睿

内容

- 函数的声明和调用
- 类的声明和使用
- 异常抛出和处理

内容

- 函数的声明和调用
- 类的声明和使用
- 异常抛出和处理

函数的声明和调用

```
def append_int_to_str (i: int, s:string) -> string:
    string_of_i = str(i)
    return s + string_of_i
```

函数的声明和调用 - def

- def <函数名>(<参数列表>):
- 函数名：下划线分割的小写单词
- 参数列表：逗号分隔的参数变量名

函数的声明和调用 – Function Annotations

- def <函数名>(<参数>: <类型>,…) -> <返回值类型>:
- 标注类型不是必须的
- 写明类型可以方便阅读、帮助IDE实现语法检查和自动补全等功能
- 没有实质性的限制

函数的声明和调用 – 参数传递

```
def my_print(x, y):
    print(x+y)


my_print(2,3)          # 函数内 x=2, y=3
                       # 屏幕显示 5
```

函数的声明和调用 – 有默认值的参数

```
def my_print(x, y=3):
    print(x+y)
```

```
my_print(2)          # 函数内 x=2, y=3 显示 5
my_print(2,4)        # 函数内 x=2, y=4 显示 6
```

函数的声明和调用 – 有默认值的参数

```
def my_print(x, y=3, z=6):
    print(x+y+z)
```

```
my_print(2)          # x=2, y=3, z=6 显示 11
my_print(2, 4)       # x=2, y=4, z=6 显示 12
my_print(2, z=0)     # x=2, y=3, z=0 显示 5
```

* 本页内容较难，不必急于掌握

函数的声明和调用 – 元组参数

```
def foo(a, *b):
    print('a='str(a)+' b='+str(b))
```

```
foo(0)                  # a=0 b=()
foo(0,1)                # a=0 b=(1,)
foo(0,1,2)              # a=0 b=(1,2)
```

* 本页内容较难，不必急于掌握

函数的声明和调用 – 元组参数

- `def foo(a, *b):`
- `    print('a='str(a)+' b='+str(b))`

- `q=(0,1,2)`
- `foo(*q)  # a=0 b=(1,2)`
- 上面的代码等价于 `foo(0,1,2)`

* 本页内容较难，不必急于掌握

函数的声明和调用 – 字典参数

```
def foo(a, **b):
    print('a='str(a)+' b='+str(b))
```

```
foo(a=0)                # a=0 b={}
foo(a=0,b=1)            # a=0 b={'b': 1}
foo(a=0,b=1,c=2)        # a=0 b={'b': 1, 'c': 2}
```

* 本页内容较难，不必急于掌握

函数的声明和调用 – 字典参数

- `def foo(a, **b):`
- `    print('a='str(a)+' b='+str(b))`

- `q={'a':0,'b':1,'c':2}`
- `foo(**q)` `# a=0 b={'b': 1, 'c': 2}`
- 上面的代码等价于 `foo(a=0,b=1,c=2)`

函数的声明和调用 – 函数的返回值

- return <expr>
- 使用return来返回一个值
- 如果返回值中有逗号，那么返回的实际上是一个元组
- 如果函数执行完后没有遇到return语句，那么返回的实际上是None（Python中的一个特殊值，类型是NoneType）

* Python技巧 – 元组的应用

- 交换两个变量：
- `a, b = b, a`

- 函数返回值：
- `def quot_rem(x, y):`
- `    a = x//y`
- `    return a, x-a*y`
- `q, r = quot_rem(5, 3)        # q=1, r=2`

注：lambda函数与一般函数仍有区别，例如不能被pickle序列化等

函数的声明和调用 – lambda函数

- `f=lambda x:x+1`
- 相当于：
- `def f(x):`
- `    return x+1`
- 可以用于构造简单的函数，例如sort传入自定义函数

函数的声明和调用 – lambda函数

- 例子：给列表a中的元素加1
- `b=list(map(lambda x:x+1, a))`
- `# 更加推荐的写法是`
- `b = [x+1 for x in a]`

* 本页内容较难，不必急于掌握

函数的声明和调用 – lambda函数的capture

- `x=3`
- `f=lambda y:x+y`
- `f(4)` `# 7`
- `x=4`
- `f(4)` `# 8`
- `del x`
- `f(4)` `# NameError`

- lambda函数中使用外部变量，会记住变量的scope和name，而不是值

内容

- 函数的声明和调用
- 类的声明和使用
- 异常抛出和处理

类的声明和使用

```
class Square:                          # 类名
    def __init__(self, a):             # 构造函数
        self.a = a
    def area(self):                    # 成员函数
        return self.a ** 2
```

类的声明和使用

```
class Square:
    def __init__(self, a):    # self类似于c++中的*this
        self.a = a            # 构造函数中可以增加成员变量
    def area(self):
        return self.a ** 2    # 使用成员变量
```

类的声明和使用

```
class Square:
    dimension = 2          # 另一类声明成员的方法
    def __init__(self, a):
        self.a = a

    def area(self):
        return self.a ** self.dimension
```

- 这种写法下，我们可以直接访问Square.dimension

类的声明和使用

- class Foo:
- 	def __init__(self, <param>):
- 		do_whatever

- a = Foo(<param>)

类的声明和使用 – 继承

```
class Parent:
    pass
class Child(Parent):
    def __init__(self):
        super().__init__()
# 如果基类有构造函数，而子类也有构造函数，往往需要在子类中手动调用基类的构造函数
# 或者写成 Parent.__init__(self)
```

注：双下划线成员实际上是可以被访问的，形如foo._Classname__privatevar

类的声明和使用 – 成员的访问权限

- 以字母开头的成员可以被外部访问
- 以双下划线（__）开头但不以双下划线结尾的成员不能够被外部访问，也不会被子类中的定义覆盖
- 以单下划线（_）开头的成员可以被外部访问和在子类中覆盖，但尽量不要这么做（部分IDE会对此做出警告）

类的声明和使用 – 成员函数类型

- 普通成员函数声明方法和一般函数没有区别
- class Foo:
- 	def bar(self, params):
- 		lines_of_code

- a=Foo()
- A.bar(2)

* 本页内容较难，不必急于掌握

类的声明和使用 – 成员函数类型

- 静态成员函数（无法获得实例的信息）
- class Foo:
- ```
      @staticmethod
  ```
- ```
      def bar():          # 括号内不再需要self
  ```
- ```
          pass
  ```
  ```

* 本页内容较难，不必急于掌握

类的声明和使用 – 成员函数类型

- 类的成员函数（只能获得实例的类型信息，不能获得它的成员变量）
- class Foo:
- ```
      @classmethod
  ```
- ```
      def bar(cls):  # 括号内第一个变量为cls
  ```
- ```
          pass       # cls为实例的类（Foo或Foo的子类）
  ```
  ```

类的声明和使用 – 特殊的成员函数

- `__init__` # 构造函数
- `__del__` # 析构函数
- `__enter__` # with块进入函数
- `__exit__` # with块退出函数

类的声明和使用 – with块应用例子：文件操作

```
fp = open('foo.txt')
    for line in fp:
        print(line)
fp.close()
```

- 可以写成：

```
with open('foo.txt') as fp:          # 再也不用担心忘记close()啦
    for line in fp:
        print(line)
```

类的声明和使用 – Python赋值机制

- 运行下面的代码，观察并思考结果：

```
class Foo:
    def __init__(self, x):
        self.bar = x

a = Foo(3)
print(a.bar)
b = a
b.bar = 4
print(a.bar)            # 4
```

类的声明和使用 – Python赋值机制

- 运行下面的代码，观察并思考结果：

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

类的声明和使用 – Python赋值机制

- Python中，使用等号赋值时，实际上是将等号右边的引用赋给了左边
- 从而，对其中一个进行操作的时候，另一个也会改变
- 但是，对其中一个重新赋值后，二者就无关了
- 对于类和之前讲过的集合类型都是如此

类的声明和使用 – Python赋值机制

- 运行下面的代码，观察并思考结果：

```
a=[1, 2, 3]
print(a[0])
b=a                          # 此时，对b进行修改会影响a
del b[0]
print(a)                     # [2, 3]
b=[]                         # 重新给b赋值，b已和a无关
print(a)                     # [2, 3]
```

内容

- 函数的声明和调用
- 类的声明和使用
- 异常抛出和处理

异常抛出和处理

- try:
- lines_of_code
- except <TypeOfException> [as <var>]:
- lines_of_code
- finally:
- lines_of_code

异常抛出和处理

```
try:
    a = 3/0
except ZeroDivisionError:
    print('oops, divide by zero.')
```

异常抛出和处理

```
try:
    a = 3/0
except ZeroDivisionError as e:
    print(e)          # division by zero
```

异常抛出和处理

```
try:
    a = 3/0
except ZeroDivisionError as e:
    cleaning_stuff
    raise      # 重新抛出相同的异常，给上层处理
```

异常抛出和处理 – 自定义异常类

- 继承Exception类
- 其他和类一样

注：准确说，基本所有的异常都是Exception的子类，因为语法本身只要求抛出的是BaseException的子类。通常实践中，我们应该继承Exception而非BaseException

异常抛出和处理 – 捕获任何异常

- 注意到，所有的异常都是Exception的子类，所以我们可以使用下面的语句来捕获所有类：

```
except Exception:

    clean_up_or_pass
```

作业2

- 运行下面的代码，观察并解释结果：

```
class Foo:
    sv = 3

a=Foo(); b=Foo();
print(a.sv, b.sv, Foo.sv)
a.sv = 4
print(a.sv, b.sv, Foo.sv)
Foo.sv = 5
print(a.sv, b.sv, Foo.sv)
```

下节内容

- 多文件的项目
- 加载模块和编写模块
- 常见模块的功能