Python 进阶训练营

尹会生

极客大学

③面向对象与设计模式

C 目 录 CONTENTS

01 变量作用域

02 变量的引用

03 数据结构

04 函数

学习目标

- 掌握 Python 变量和函数底层原理
- 掌握高效的数据结构

变量

命令问题

关键字

import keyword
keyword.kwlist

避免的名称

单字符名称，除了计数器和迭代器
包/模块名中的连字符(-)
双下划线开头并结尾的名称（Python 保留，例如__init__）

建议命名规范

General Naming Convention

Type	Public	Internal
Packages	`lower_with_under`	
Modules	`lower_with_under`	`_lower_with_under`
Classes	`CapWords`	`_CapWords`
Exceptions	`CapWords`	
Functions	`lower_with_under()`	`_lower_with_under()`
Global/Class Constants	`CAPS_WITH_UNDER`	`_CAPS_WITH_UNDER`
Global/Class Variablesi	`lower_with_under`	`_lower_with_under`
Instance Variables	`lower_with_under`	`_lower_with_under` (protected) or `__lower_with_under` (private)
Method Names	`lower_with_under()`	`_lower_with_under()` (protected) or `__lower_with_under()` (private)
Function/Method Parameters	`lower_with_under`	
Local Variables	`lower_with_under`	

极客大学

变量

入口

```
def main():
    ...

if __name__ == '__main__':
    main()
```

变量作用域

高级语言对变量的使用：

- 变量声明
- 定义类型（分配内存空间大小）
- 初始化（赋值、填充内存）
- 引用（通过对象名称调用对象内存数据）

Python 和高级语言有很大差别，在模块、类、函数中定义，才有作用域的概念。

Python 作用域遵循 LEGB 规则。

变量赋值

面试题 1

```
a = 123
b = 123
c = a
a = 456
c = 789
c = b = a
```

基本数据类型的可变类型与不可变类型区分

面试题 2

```
a = [1,2,3]
b = a
a.append(4)
print(b)
```

基本数据类型的可变类型与不可变类型区分

面试题 3

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

a和b分别是什么值？

```
a = [1, 2, 3]
b = a
a[0],a[1],a[2] = 4, 5, 6
```

a和b分别是什么值？

面试题 4

思考哪种类型可以做字典的 key？为什么？

深拷贝、浅拷贝

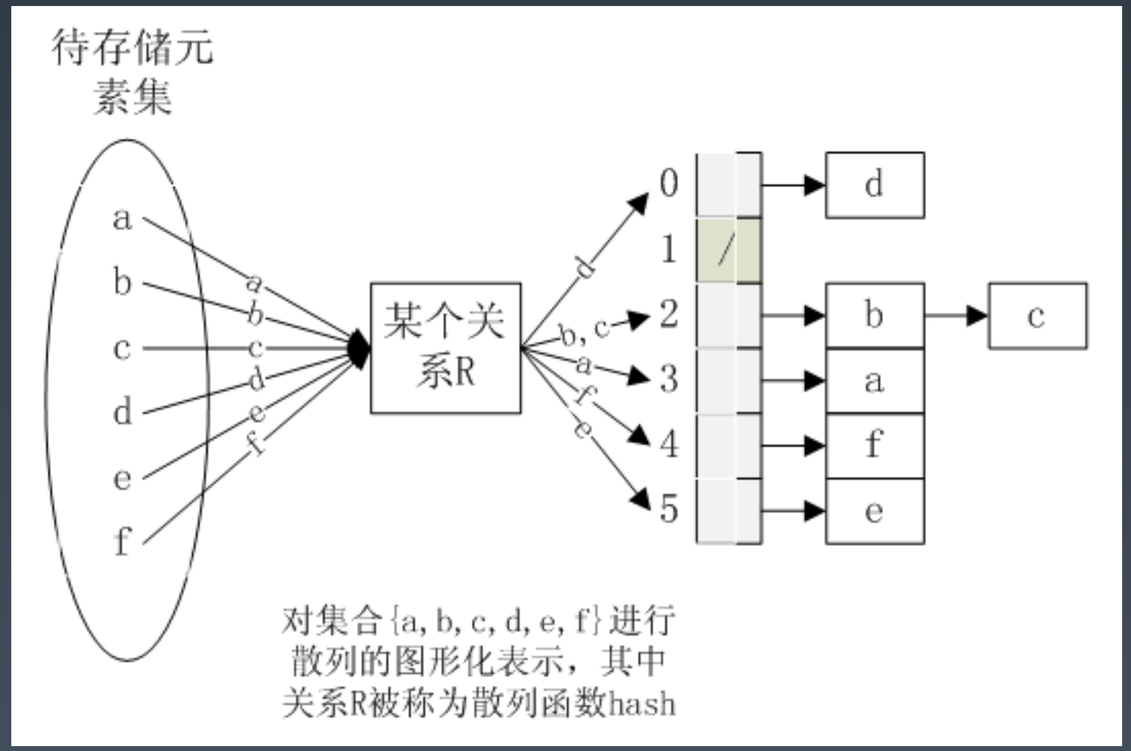
非容器（数字、字符串、元组）类型没有拷贝

```
import copy
copy.copy(object)
copy.deepcopy(object)
```

序列

扁平序列	容器序列
字符串	列表、元组
只能容纳一种类型	可以存放不同类型的数据

字典与哈希



对集合{a, b, c, d, e, f}进行散列的图形化表示，其中关系R被称为散列函数hash

极客大学

推导式

推导式是构建列表、字典、集合和生成器便捷方式

一个列表推导式的例子：

```
mylist = []

for i in range(1, 11):
    if i > 5:
        mylist.append(i**2)

print(mylist)
```

转换为列表推导式：

```
mylist = [i**2 for i in range(1, 11) if i > 5]
```

推导式

推导式语法：
[表达式 for 迭代变量 in 可迭代对象 if 条件]

其他常见用法：

循环嵌套

```
mylist = [str(i)+j for i in range(1, 6) for j in 'ABCDE']
```

字典转换为列表

```
mydict = {'key1' : 'value1', 'key2' : 'value2'}
mylist = [key + ':' + value for key, value in mydict.items()]
print(mylist)
```

字典 key 和 value 互换

```
{value: key for key, value in mydict.items()}
```

推导式

字典推导式

```
mydict = {i: i*i for i in (5, 6, 7)}
print(mydict)
```

集合推导式

```
myset = {i for i in 'HarryPotter' if i not in 'er'}
print(myset)
```

元组推导式要显式使用 tuple()，不能直接使用()

生成器

```
mygenerator = (i for i in range(0, 11))
print(mygenerator)
print(list(mygenerator))
```

字符串的连接和拆分

常用连接方法

```
'time' + 'geekbang' + 'org’

'%s %s %s' %('time', 'geekbang', 'org')

mylist = ['time', 'geekbang', 'org']
'.'.join(mylist)
```

拆分方式

```
mystring = 'time.geekbang.org'
print(mystring.split('.')[0])

var1,var2 = mystring.split('.')[1:3]
print(var1)
print(var2)
```

使用 collections 扩展内置数据类型

collections 提供了加强版的数据类型

https://docs.python.org/zh-cn/3.6/library/collections.html

namedtuple -- 带命名的元组

```
import collections
Point = collections.namedtuple('Point', ['x', 'y’ ])
p = Point(11, y=22)
print(p[0]  + p[1])
x, y = p
print(p.x + p.y)
print(p)
```

deque 双向队列

Counter 计数器

函数的可变长参数

一般可变长参数定义如下：

```
def func(*args, **kargs):
    pass
```

kargs　获取关键字参数
args　获取其他参数

示例：

```
def func(*args, **kargs):
    print(f'args: {args}’ )
    print(f'kargs:{kargs}’ )

func(123, 'xyz', name='xvalue' )
```

Lambda 表达式

Lambda 只是表达式，不是所有的函数逻辑都能封装进去

```
k = lambda x:x+1
print(k(1))
```

Lambda 表达式后面只能有一个表达式

- 实现简单函数的时候可以使用 Lambda 表达式替代
- 使用高阶函数的时候一般使用 Lambda 表达式

高阶函数

高阶：参数是函数、返回值是函数

常见的高阶函数：map、reduce、filter、apply

apply 在 Python2.3 被移除，reduce 被放在 functools 包中

推导式和生成器表达式可以替代 map 和 filter 函数

高阶函数

map(函数， 序列) 将序列中每个值传入函数，处理完成返回为 map 对象

```
number = list(range(11))

def square(x):
    return x**2

print(list(map(square, number)))

print(dir(map(square, number)))
```

filter(函数，序列)将序列中每个值传入函数，符合函数条件的返回为 filter 对象

THANKS!