[陈方杰]

**零基础入门学习Python**

AMUSI

**零基础入门学习Python**

**资源**

[bbs论坛](http://bbs.fishc.com/forum-243-1.html)

[blog.fishc](http://blog.fishc.com/category/python)

[测试题](https://wenku.baidu.com/view/4b0c8d2faa00b52acec7caae.html)

[Python编程规范](http://zh-google-styleguide.readthedocs.io/en/latest/google-python-styleguide/contents/)

# 章节1 就这么愉快的开始吧

## 课时1 我和Python的第一次亲密接触

### 测试题

1. Python是什么类型的语言？

解释1：Python是一种面向对象的解释型计算机程序设计语言

解释2：Python是一种开放源代码的脚本编程语言

1. IDLE是什么？

IDLE是纯Python下使用Tkinter编写的开发开发Python程序的基本IDE。启动IDLE后，首先映入眼帘的是Python shell，我们可通过它在IDLE内部执行Python命令。

Linux安装IDLE

* IDLE\_Python2.x: sudo apt-get install idle
* IDLE\_Python3.x: sudo apt-get install idle3

1. print()作用是什么？

格式化输出，可以直接输出常量和变量（包括整型、浮点型和字符串等）

1. Python中表示乘法的符号是什么？

\*表示乘法

1. 为什么print(“I love Python” \* 5)可以正常运行，但print(“I love Python” + 5)却报错？

Python中不能将两个类型完全不同的对象加在一起。”I Love Python”是字符串，5是整型，两个对象类型不一致，+加在一起输出会让Python解释器不理解输出的对象是什么？而\*则告诉了Python解释器，字符串\*5即打印5次。

1. 如果我需要在一个字符串中嵌入一个双引号，正确的做法是？

加上反斜杠（\）转义字符，如\”。

>>> print("\"I Love Python\"")

"I Love Python"

>>>

1. 为什么我们要使用Python3？Python2到底有什么问题？看起来很多程序员依然都在使用Python2？

官方声称Python2的维护截至到2020年，可见Python3才是未来的趋势，目前Python2的库很多，但Python3也在日益强大，从深度学习框架使用的Python版本就可以看出来。

注：如果想让系统兼容Python2和Python3，可以使用[Python3和Python2兼容库](https://pythonhosted.org/six/)

### 动动手

1. 动手试试在IDLE3/Python中直接输入5+8与输入print(5+8)有何不同？

>>> 5+8

13

>>> print(5+8)

13

>>>

看起来是一致的输出。

1. 在交互模式中，使用Python计算一年有多少秒？

>>> print("一年有 %d 秒" % (365\*24\*60\*60))

一年有 31536000 秒

>>>

1. 设置你的操作系统的环境变量，以便可以轻松进入Python环境：

**Window**

Win+R输入cmd打开终端（或者输入cmder打开终端）

输入python，即进入Python环境（我使用的是Anaconda的Python3.5）

**Linux**

Ctl+Alt+T打开终端

a)输入python2，即进入python2.7.12环境（ubuntu16.04自带）

b)输入python3，即进入python3.6.2环境（安装的Anaconda3）

c)输入idle，即进入python2.7.12环境（ubuntu16.04自带）

d)输入idle3，即进入python3.6.2环境（安装的Anaconda3）

## 课时2 用Python设计第一个游戏

### 知识点1

Start

Judge

End

**Code**

print(".............Amusi...........")  
temp = input("Can you guess which number I am thinking?")  
guess = int(temp)  
if guess == 8:  
 print("Congratulations!")  
else:  
 print("Sorry")  
print("Game Over...")

### 知识点2

**内建函数BIF(build-in functions)**

运行下述代码，输出’’小写的字符串都是内建函数，并可使用help函数查看内建函数信息，如help(input)。

**代码**

print(dir(\_\_builtins\_\_))  
print(help(input))

**结果**

['ArithmeticError', 'AssertionError', 'AttributeError', 'BaseException', 'BlockingIOError', 'BrokenPipeError', 'BufferError', 'BytesWarning', 'ChildProcessError', 'ConnectionAbortedError', 'ConnectionError', 'ConnectionRefusedError', 'ConnectionResetError', 'DeprecationWarning', 'EOFError', 'Ellipsis', 'EnvironmentError', 'Exception', 'False', 'FileExistsError', 'FileNotFoundError', 'FloatingPointError', 'FutureWarning', 'GeneratorExit', 'IOError', 'ImportError', 'ImportWarning', 'IndentationError', 'IndexError', 'InterruptedError', 'IsADirectoryError', 'KeyError', 'KeyboardInterrupt', 'LookupError', 'MemoryError', 'ModuleNotFoundError', 'NameError', 'None', 'NotADirectoryError', 'NotImplemented', 'NotImplementedError', 'OSError', 'OverflowError', 'PendingDeprecationWarning', 'PermissionError', 'ProcessLookupError', 'RecursionError', 'ReferenceError', 'ResourceWarning', 'RuntimeError', 'RuntimeWarning', 'StopAsyncIteration', 'StopIteration', 'SyntaxError', 'SyntaxWarning', 'SystemError', 'SystemExit', 'TabError', 'TimeoutError', 'True', 'TypeError', 'UnboundLocalError', 'UnicodeDecodeError', 'UnicodeEncodeError', 'UnicodeError', 'UnicodeTranslateError', 'UnicodeWarning', 'UserWarning', 'ValueError', 'Warning', 'ZeroDivisionError', '\_\_build\_class\_\_', '\_\_debug\_\_', '\_\_doc\_\_', '\_\_import\_\_', '\_\_loader\_\_', '\_\_name\_\_', '\_\_package\_\_', '\_\_spec\_\_', 'abs', 'all', 'any', 'ascii', 'bin', 'bool', 'bytearray', 'bytes', 'callable', 'chr', 'classmethod', 'compile', 'complex', 'copyright', 'credits', 'delattr', 'dict', 'dir', 'divmod', 'enumerate', 'eval', 'exec', 'exit', 'filter', 'float', 'format', 'frozenset', 'getattr', 'globals', 'hasattr', 'hash', 'help', 'hex', 'id', 'input', 'int', 'isinstance', 'issubclass', 'iter', 'len', 'license', 'list', 'locals', 'map', 'max', 'memoryview', 'min', 'next', 'object', 'oct', 'open', 'ord', 'pow', 'print', 'property', 'quit', 'range', 'repr', 'reversed', 'round', 'set', 'setattr', 'slice', 'sorted', 'staticmethod', 'str', 'sum', 'super', 'tuple', 'type', 'vars', 'zip']

input(prompt=None, /)

Read a string from standard input. The trailing newline is stripped.

The prompt string, if given, is printed to standard output without a trailing newline before reading input.

If the user hits EOF (\*nix: Ctrl-D, Windows: Ctrl-Z+Return), raise EOFError.

On \*nix systems, readline is used if available.

### 测试题

1. 什么是BIF？

Build-in Functions内置函数，为了方便程序员快速编写脚本程序。

注：输入dir(\_\_builtins\_\_)查看python的内置函数列表，可用help(name)查看内置函数的介绍。

1. 用课堂上小甲鱼教的方法数一数Python3提供多少个BIF？

在Python或IDLE中，输入入dir(\_\_builtins\_\_)可以查看Python提供的内置方法列表，其中小写的就是BIF。根据计算（其实就是数数），得知目前Python3提供80个BIF。

1. 在Python看来：”FishC”和”fishc”一样吗？

不一样，Python对于字母的大小写敏感，所以对于Python，FishC和fishc是两个完全不同的名字。

1. 在小甲鱼看来，Python中什么是最重要的？你赞同吗？

缩进！在小甲鱼看来，缩进是Python的灵魂，缩进的严格要求使得Python的代码显得非常精简并且有层次。（C/C++有{}大括号来约束结构体，Python则靠缩进）

1. “=”和“==”各自表示什么含义？如何避免在编程中不小心把”==”误写成”=”的问题？

Python不允许if条件中赋值，所以if c = 1:会报错，而C/C++则会先进行赋值，后继续判断，这违背了程序员的意思。

注：无论是Python还是C/C++等其它语言，在判断语句中，好的习惯是将常量放在=左边，将变量放在=右边，如if 1 == c:，或者if(1==c)（前提是一方是常量，另一方是变量）。

1. 你听过“拼接”这个词吗？

将两个字符串“相加”，被称为拼接字符串。如”I” + ”Love” + “Python”会得到“ILovePython”。

>>> "I" + "Love" + "Python"

'ILovePython'

### 动动手

1. 编写程序：hello.py，要求用户输入姓名并打印“您好，姓名！”

name = input("请输入您的姓名:")  
print("您好, " + name + "!")

1. 编写程序：calc.py，要求用户输入1到100之间数字并判断，输入符合要求“厉害了，Word哥”，不符合要求则打印“你走吧～”

numsStr = input("请您输入0~100之间的数字: ")  
nums = int(numsStr)  
if 0 <= nums <= 100:  
 print("厉害了, Word哥")  
else:  
 print("你走吧~")

1. 请写下这一节课你学到的内容：格式不限

* 缩进的重要性
* 内置函数知识点
* input函数的使用
* 简单的if-else结构体

# 章节2 成为高手前必须知道的基础知识

## 课时3 小插曲之变量和字符串

### 知识1 变量

**注意事项**

* 在使用变量之前，需要对其先赋值
* 变量名可以包含字符、数字、下划线，但变量名不能以数字开头（和C/C++一致）
* 等号（=）是赋值的含义，左边是变量名，右边是值（不能写反了）
* 变量的[命名理论](http://zh-google-styleguide.readthedocs.io/en/latest/google-python-styleguide/contents/)
* 用Python创建一个字符串时，需要在字符两边加上引号，可以是单引号，也可以是双引号，但必须成对出现。如'9'和"9"
* 如果字符串中需要出现单引号或双引号怎么办？

答：使用反斜杠（\），也就是转义字符（\），对字符串中的引号进行转义

* 反斜杠的弊端，即需要打印\时候，会遇到一些问题，如下：

>>> str = "C:\now"

>>> str

'C:\now'

>>> print(str)

C:

ow

答：使用原始字符串，即只需要在字符串前加一个英文字母r即可：

>>> str = r"C:\now"

>>> print(str)

C:\now

**代码**

# string variable  
teacher = "Amusi"  
print(teacher)  
teacher = "Lucy"  
print(teacher)  
# int variable  
first = 3  
second = 8  
third = first + second  
print(third)  
#  
myteacher = "Amusi"  
yourteacher = "Lucy"  
ourteacher = myteacher + yourteacher  
print(ourteacher)

**输出结果**

Amusi

Lucy

11

AmusiLucy

### 知识2 长字符串

* 如果希望得到一个跨越多行的字符串，如：

Amusi love Lucy,

as same as Lucy love Amusi,

Forever love...

答：可以使用三重引号字符串"""contents"""

# 长字符串  
longStr = """Amusi love Lucy,  
as same as Lucy love Amusi,  
Forever love..."""  
print(longStr)

### 测试题：

1. 以下哪个变量的命名不正确？为什么？
2. MM\_520 (B)\_MM520\_ (C)520\_MM (D)\_520\_MM

答：(C)选项命名不正确，因为不能以数字开头来命名变量

1. 在不上机的情况下，以下代码你能猜到屏幕会打印什么内容吗？

>>> myteacher = "Amusi"

>>> yourteacher = "Lucy"

>>> print(myteacher)

答：输出Amusi

1. 在不上机的情况下，以下代码您能猜到屏幕会打印什么内容吗？

>>> myteacher = "Amusi"

>>> yourteacher = myteacher

>>> myteacher = "Lucy"

>>> print(yourteacher)

答：输出Amusi

1. 在不上机的情况下，以下代码您能猜到屏幕会打印什么内容吗？

>>> first = 88

>>> second = "520"

>>> first = second

>>> print(first)

答：输出520（字符串）

4.除了使用反斜杠（\）进行字符转义，还有什么方法打印出：Let’s go!这个字符串？

1. 如果非要在原始字符串结尾输入反斜杠，可以如何灵活处理？

答：可以使用原始字符串，即在字符串前加上r。如r”C:/now”

1. 变量名的命名方式有什么好的习惯？

变量名的介绍要尽量详细，如people\_head，增加程序可读性。

### 动动手：

1. 使用变量，计算一年有多少秒？

year\_days = 365  
day\_hours = 24  
hour\_mins = 60  
min\_second = 60  
year\_seconds = year\_days \* day\_hours \* hour\_mins \* min\_second  
print("一年有 %d 秒"% (year\_seconds))

1. 关于三重引号字符串（长字符串），其实可以使用”\n”符号来替换，但后者需要添加多个，较为麻烦。
2. 回忆上述内容

答：

* 变量名的命名
* 变量的赋值
* 三重引号字符串

## 课时4 改进我们的小游戏

### 知识1 条件分支

* 第一个改进要求：猜错的时候程序提示用户当前的输入比答案大了还是小了

表4.1 Python比较操作符

|  |  |
| --- | --- |
| > | 左边大于右边 |
| >= | 左边大于等于右边 |
| < | 左边小于右边 |
| <= | 左边小于等于右边 |
| == | 左边等于右边 |
| ！= | 左边不等于右边 |

注：Python有>=（<=）操作符，但C/C++没有这类操作符，若要完成此功能，需要用&&与操作符将两个操作符，即>和=（<和=）连接。如 a < b && b < c

**测试代码**

>>> 1 < 3

True

>>> 1 > 3

False

>>> 1 == 1

True

>>> 1 != 2

True

>>> 1 == 2

False

>>> 8 >= 5

True

* Python条件分支语法

if 条件 :

条件为真（True）执行的操作

else :

条件为假（False）执行的操作

注：格外关心缩进

**代码**

# 条件分支

guessStr = input("请输入一个数值: ")

guess= int(guessStr)

if guess == 8:

print("好厉害，猜对咯!")

else:

if guess > 8:

print("偏大了...")

else:

print("偏小了...")

print("游戏结束！")

* while循环

**语法**

while 条件:

条件为真（True）执行的操作

**代码**

# 条件分支

guessStr = input("请输入一个数值: ")

guess= int(guessStr)

counts = 0

while guess != 8:

if counts ==3: # 限制输入次数

break

counts += 1

guessStr = input("猜错啦，请重新输入一个数值: ")

guess= int(guessStr)

if guess == 8:

print("好厉害，猜对咯!")

else:

if guess > 8:

print("偏大了...")

else:

print("偏小了...")

print("游戏结束！")

* and逻辑与操作符

**语法**

True and True -> True

True and False -> False

False and True -> False

False and False -> False

注：逻辑与的语法是只要有一个为False条件，则该语句输出False，只有两个条件同时为True，语句才输出True。

**测试代码**

>>> (3 > 2) and (1 < 2)

True

* random模块

Random模块中有一个randint函数，它会返回一个随机的整数。

**代码**

# 条件分支

import random

secret = random.randint(1,10)

guessStr = input("请输入一个数值: ")

guess= int(guessStr)

#counts = 0

while guess != secret:

#if counts ==3: # 限制输入次数

# break

#counts += 1

guessStr = input("猜错啦，请重新输入一个数值: ")

guess= int(guessStr)

if guess == secret:

print("好厉害，猜对咯!")

else:

if guess > secret:

print("偏大了...")

else:

print("偏小了...")

print("游戏结束！")

### 测试题：

0. 请问以下代码会打印多少次“我爱鱼C！”

while 'A':

    print('Amusi!')

答：程序陷入死循环，会无限次输出。

1. 请问以下代码会打印多少次“我爱鱼C！”

i = 10

while i:

    print('我爱鱼C!')

    i = i - 1

答：打印10次，因为10,9,8,...,2,1，当i=0时whilie条件为False，程序结束。

1. 请写出与 10 < cost < 50 等价的表达式

if 10 < cost < 50:

1. Python3 中，一行可以书写多个语句吗？

答：可以，加上(“\n”)换行符就可以实现多个语句; 更简单地是使用多行符号”””xxx”””

1. Python3 中，一个语句可以分成多行书写吗？

答：在需要换行的语句后面加上(\)反斜杠符号，即可实现多行书写。

print("Amusi" \

" love Lucy")

1. 请问Python的 and 操作符 和C语言的 && 操作符 有何不同？【该题针对有C或C++基础的朋友】

答：

用法和效果一致。

表4.1 与操作符性质

|  |  |
| --- | --- |
| Python | C/C++ |
| True and True -> True | True and True -> True |
| True and False -> False | True and False -> False |
| False and True -> False | False and True -> False |
| False and False -> False | False and False -> False |

注：逻辑与的语法是只要有一个为False条件，则该语句输出False，只有两个条件同时为True，语句才输出True。

6. 听说过“短路逻辑（short-circuit logic）”吗？

答：编程语言常用的逻辑：

if a and b: # 如果a为False，那么跳过b的判断，结果直接为False

If a or b: # 如果a为True，那么跳过b的判断，结果直接为True

## 课时5 闲聊之Python的数据类型

### 5.1整型

5.2浮点型

### 5.3 e记法

e记法是float的科学技术法

>>> 9e10

90000000000.0

### 5.4布尔型

True:

False:

### 5.5 字符串

**字符串内建函数操作**

s为字符串

表5.1 字符串内建函数说明

|  |  |
| --- | --- |
| s.isalnum() | 所有字符都是数字或者字母，为真返回 Ture，否则返回 False |
| s.isalpha() | 所有字符都是字母，为真返回 Ture，否则返回 False |
| s.isdigit() | 所有字符都是数字，为真返回 Ture，否则返回 False |
| s.islower() | 所有字符都是小写，为真返回 Ture，否则返回 False |
| s.isupper() | 所有字符都是大写，为真返回 Ture，否则返回 False |
| s.istitle() | 所有单词都是首字母大写，为真返回 Ture，否则返回 False |
| s.isspace() | 所有字符都是空白字符，为真返回 Ture，否则返回 False |

例如：

>>> s = "AMUSI"

>>> s.isupper()

True

>>> s = "Amusi"

>>> s.istitle()

True

>>> s = "520"

>>> s.isdigit()

True

>>> s = "520a"

>>> s.isdigit()

False

>>> s = "lucy"

>>> s.islower()

True

### 5.6 类型转换

转换

字符串

浮点数

整数

str()

float()

int()

图5.1 类型转换

BIF内建函数

* int()： 将字符串或浮点数转换成整数

#

>>> a = "520"

>>> b = int(a)

>>> b

520

# 取整处理

>>> a = 8.99

>>> b = int(a)

>>> b

8

* float()： 将字符串或整数转换成浮点数

# 将字符串转换成浮点型

>>> a = "520"

>>> b = float(a)

>>> b

520.0

# 将整型转换成浮点型

>>> a = 520

>>> b = float(a)

>>> b

520.0

* str()： 将整数或浮点数转换成字符串

>>> a = 5.99

>>> b = str(a)

>>> b

'5.99'

>>> c = str(5e19)

>>> c

'5e+19'

### 5.7 获得关于类型的信息

* type： 获得变量的类型，返回值为变量的类型（int, float, str or bool）
* Isinstance：判断变量的类型，返回值为布尔类型（True or False）

**type()**

>>> a = "520"

>>> type(a)

<class 'str'>

>>> type(5.2)

<class 'float'>

>>> type(8)

<class 'int'>

>>> type(True)

<class 'bool'>

>>> type(5e15)

<class 'float'>

**isinstance()**

>>> a = "Amusi"

>>> isinstance(a, str)

True

>>> isinstance(a, int)

False

>>> b = 520

>>> isinstance(b, int)

True

### 测试题：

1. 在 Python 中，int 表示整型，那你还记得 bool、float 和 str 分别表示什么吗？

答：

bool表示布尔类型

float表示浮点型

str表示字符串

1. 你知道为什么布尔类型(bool)的 True 和 False 分别用 1 和 0 来代替吗？

答：因为计算机只认识01010101二进制数，二进制数只有0，1的组合。因此True和False用1和0代替很合适。

1. 使用 int() 将小数转换为整数，结果是向上取整还是向下取整呢？

答：向下取整，获得下舍整数（如int(8.99)，结果是8）

1. 我们人类思维是习惯于“四舍五入”法，你有什么办法使得 int() 按照“四舍五入”的方式取整吗？

答：

方法一：可以先使用round()函数，再使用int函数

**round()函数语法**

round(x, [, n])

**round()函数参数**

x--数值表达式

n--数值表达式，表示保留小数点后n位

**Round()函数返回值**

返回浮点数x的四舍五入值

**代码实例**

>>> a = round(5.9998, 2)

>>> a

6.0

>>> a = round(5.6825912, 2)

>>> a

5.68

方法二：再加上0.5计算

5.4 “四舍五入”结果为：5，int(5.4+0.5) == 5

5.6 “四舍五入”结果为：6，int(5.6+0.5) == 6

4. 取得一个变量的类型，视频中介绍可以使用 type() 和 isinstance()，你更倾向于使用哪个？

答：取得一个变量的类型，当然是使用type()函数，因为isinstance()是用来判断类型的

1. Python3 可以给变量命名中文名，知道为什么吗？

答： Pyhton3 源码文件默认使用utf-8编码（支持中文）

**代码实例**

>>> 阿木寺 = "Amusi"

>>> print(阿木寺)

Amusi

6. 【该题针对零基础的鱼油】你觉得这个系列教学有难度吗？

暂无

### 动动手：

1. 针对视频中小甲鱼提到的小漏洞，再次改进我们的小游戏：当用户输入错误类型的时候，及时提醒用户重新输入，防止程序崩溃。

答：可使用字符串内建函数s.isdigit()，来判断输入是否全为数字

**代码实例**

# 条件分支

import random

secret = random.randint(1,10)

guessStr = input("请输入一个数值: ")

while not guessStr.isdigit():

guessStr = input("输入类型不符，请重新输入数值: ")

guess= int(guessStr)

#counts = 0

while guess != secret:

#if counts ==3: # 限制输入次数

# break

#counts += 1

guessStr = input("猜错啦，请重新输入一个数值: ")

while not guessStr.isdigit():

guessStr = input("输入类型不符，请重新输入数值: ")

guess= int(guessStr)

if guess == secret:

print("好厉害，猜对咯!")

else:

if guess > secret:

print("偏大了...")

else:

print("偏小了...")

print("游戏结束！")

1. 写一个程序，判断给定年份是否为闰年。（注意：请使用已学过的 BIF 进行灵活运用）

注：这样定义闰年的:能被4整除但不能被100整除,或者能被400整除都是闰年。

numsStr = input("请输入一个年份（整数）： ")

while not numsStr.isdigit():

numsStr = input("输入的类型错误，请再重新输入一个年份（整数）：")

nums = int(numsStr)

if ((nums % 2 == 0) and (nums % 100 !=0)) or (nums % 400 == 0):

print("恭喜!该年份是闰年!")

else:

print("抱歉!该年份不是闰年!")

2. 请写下这一节课你学习到的内容：格式不限，回忆并复述是加强记忆的好方式！

* 类型变换函数：int(), float, str()
* 类型获得/判断函数：type(), isinstance()
* e记法
* str内建函数：isdigit(), islower(), isupper()...

## 课时6 Python之常用操作符

### 6.1 算术操作符

表6.1 算术操作符

|  |  |  |  |
| --- | --- | --- | --- |
| + 加 | / 除 | // 整除 （相当于floor函数） |  |
| - 减 | % 余 |  |  |
| \* 乘 | \*\* 幂 |  |  |

注：Python中floor函数和//效果一致，用于返回数字的下舍整数

**代码实例**

>>> 10 / 8

1.25

>>> 10 **//** 8

1

>>> import math

>>> math.floor(10/8)

1

>>> -3 \*\* 2

-9

>>> 9 / 6

1.5

>>> 9 % 6

3

### 6.2 逻辑操作符

* and
* or
* Not

输出： True/False

**代码实例**

>>> not 4

False

>>> 3 < 4 < 5

True

### 6.3 优先级问题

表2. 操作符优先级

|  |  |
| --- | --- |
| **级别** | **操作符** |
| 1 | 幂运算 \*\* |
| 2 | 正负号 +x -x(前缀) |
| 3 | 算术操作符 \* / // + - % |
| 4 | 比较操作符 < <= > >= == != |
| 5 | 逻辑操作符 not and or |

**测试题**

0. Python 的 floor 除法现在使用 “//” 实现，那 3.0 // 2.0 您目测会显示什么内容呢？

答：使用floor函数之前，需要import math

Python2.x中

>>> import math

>>> math.floor(10 / 6)

1.0

>>> math.floor(2 / 3)

0.0

>>> math.floor(20 / 10)

2.0

>>> math.floor(8.8 / 9.9)

0.0

>>> 3.0 // 2.0

1.0

Python3.x中

>>> import math

>>> math.floor(10/8)

1

>>> import math

>>> math.floor(10 / 6)

1

>>> math.floor(2 / 3)

0

>>> math.floor(20 / 10)

2

>>> math.floor(8.8 / 9.9)

0

>>> 3.0 // 2.0

1.0

由上输出结果可知，两种环境下，3.0 // 2.0结果都为1.0。

注：Python2和Python3返回的情况不太一致，对于Python2来说，floor函数返回值都是float的下舍整数。

1. a < b < c 事实上是等于？

答： a < b and b < c

1. 不使用 IDLE，你可以轻松说出 5 \*\* -2 的值吗？

答：\*\*是幂运算，所以5 \*\* -2 == 5 \*\* (-2)

>>> 5 \*\* -2

0.04

1. 如何简单判断一个数是奇数还是偶数？

答：使用%取余操作符，如果结果等于0即为偶数，如果结果等于1即为奇数

numStr = input("请输入需要查询的数值： ")

num = int(numStr)

if (num % 2) == 0:

print("Even")

else:

print("Odd")

1. 请用最快速度说出答案：not 1 or 0 and 1 or 3 and 4 or 5 and 6 or 7 and 8 and 9

答： not or and 的优先级是不同的：not > and > or

我们按照优先级给它们加上括号：(not 1) or (0 and 1) or (3 and 4) or (5 and 6) or (7 and 8 and 9)

== 0 or 0 or 4 or 6 or 9

== 4

注：“短路逻辑”：3 and 4 == 4，而 3 or 4 == 3

>>> not 1 or 0 and 1 or 3 and 4 or 5 and 6 or 7 and 8 and 9

4

>>> 3 and 99

99

>>> 3 or 99

3

1. 还记得我们上节课那个求闰年的作业吗？如果还没有学到“求余”操作，还记得用什么方法可以“委曲求全”代替“%”的功能呢？

答： 嘿嘿，一开始用的就是“%”取余操作符

numsStr = input("请输入一个年份（整数）： ")

while not numsStr.isdigit():

numsStr = input("输入的类型错误，请再重新输入一个年份（整数）：")

nums = int(numsStr)

if ((nums % 2 == 0) and (nums % 100 !=0)) or (nums % 400 == 0):

print("恭喜!该年份是闰年!")

else:

print("抱歉!该年份不是闰年!")

### 动动手

0. 请写一个程序打印出 0~100 所有的奇数。

答：

# version1

nums = 0

while nums != 100:

if nums % 2 == 1:

print(nums)

nums += 1

# version2

for nums2 in range(0, 100):

if nums2 % 2 == 1:

print(nums2)

1. 我们说过现在的 Python 可以计算很大很大的数据，但是......真正的大数据计算可是要靠刚刚的硬件滴，不妨写一个小代码，让你的计算机为之崩溃？

2.

爱因斯坦曾出过这样一道有趣的数学题：有一个长阶梯，若每步上2阶，最后剩1阶；若每步上3阶，最后剩2阶；若每步上5阶，最后剩4阶；若每步上6阶，最后剩5阶；只有每步上7阶，最后刚好一阶也不剩。

（小甲鱼温馨提示：步子太大真的容易扯着蛋~~~）

题目：请编程求解该阶梯至少有多少阶？

答： 119

for nums in range(0, 100000):

if (nums % 2 == 1) and (nums % 3 == 2) and (nums % 5 == 4) \

and (nums % 6 == 5) and( nums % 7 == 0):

print(nums)

break

1. 请写下这一节课你学习到的内容：格式不限，回忆并复述是加强记忆的好方式！

答：

* “\*\*”幂运算符的用法
* “//”和floor函数向下取整的用法
* “%”取余操作符的用法
* for循环结构体的用法
* range用法

# 章节3 了不起的分支和循环

## 课时7 了不起的分支和循环1

### 打飞机框架

加载背景音乐

播放背景音乐（设置单曲循环）

我方飞机诞生

interval = 0

while True:

if 用户是否关闭按钮:

退出程序

interval += 1

if interval == 50:

Interval = 0

小飞机诞生

小飞机移动一个位置

屏幕刷新

if 用户鼠标产生移动:

我方飞机中心位置 = 用户鼠标位置

屏幕刷新

if 我方飞机与小飞机发生肢体冲突:

我方卒，播放撞击音乐

打印“Game Over“

停止背景音乐，最好淡出

注：判断和循环的巧妙使用

## 课时8 了不起的分支和循环2

### 8.1 if-else-elif知识点

**代码1**

score = int(input("Please input a score: "))

if 100 >= score >= 90:

print("A")

if 90 > score >= 80:

print("B")

if 80 > score >= 60:

print("C")

if 60 > score >= 0:

print("D")

if score > 100 or score < 0:

print("Error input")

**代码2**

score = int(input("Please input a score: "))

if 100 >= score >= 90:

print("A")

else:

if 90 > score >= 80:

print("B")

else:

if 80 > score >= 60:

print("C")

else:

if 60 > score >= 0:

print("D")

else:

if score > 100 or score < 0:

print("Error input")

**代码3**

score = int(input("Please input a score: "))

if 100 >= score >= 90:

print("A")

elif 90 > score >= 80:

print("B")

elif 80 > score >= 60:

print("C")

elif 60 > score >= 0:

print("D")

else:

print("Error input")

分析：代码1计算冗余，代码3计算量最少

注：Python可以通过**缩进**避免”悬挂else“

### 8.2 条件表达式（三元/目操作符）

使用三元操作符可以用一条语句来完成以下的条件判断和赋值操作：

**代码1**

x,y = 8, 99

if x < y:

small = x

else:

small = y

print("The smaller value: %d" % small)

**代码2**

small = x if x < y else y # 三元运算符

print("The smaller value: %d" % small)

### 8.3 assert断言

* assert这个关键字，又称为”断言“。当assert关键字后面的条件为假时，程序自动崩溃并抛出AssertionError的异常。

**代码实例**

>>> assert 99 > 88

>>> assert 88 > 99

Traceback (most recent call last):

File "<pyshell#8>", line 1, in <module>

assert 88 > 99

AssertionError

* 一般来说，我们可以用assert在程序中置入检查点。当需要确保程序中的某个条件一定为真才能让程序正常工作的话，assert关键字的作用就凸显出来了。

### 测试题

1. if not (money < 100): 上边这行代码相当于？

答： if money >= 100:

1. assert 的作用是什么？

答：确保某条件返回值一定为True

1. 假设有 x = 1，y = 2，z = 3，请问如何快速将三个变量的值互相交换？

答：x, y, z = z, y, x

1. 猜猜 (x < y and [x] or [y])[0] 实现什么样的功能？

答： 取x和y中的最小值

这其实是 Python 的作者还没有为 Python 加入三元操作符之前，Python 社区的小伙伴们灵活的使用 and 和 or  搭配来实现三元操作符的功能，这里边有涉及到列表和切片的知识，这部分知识很快就会讲解，迫不及待的朋友可以先稍微预习下。

代码实例：

>>> x = 9

>>> y = 19

>>> (x < y and [x] or [y])[0]

9

>>> x = 19

>>> y = 9

>>> (x < y and [x] or [y])[0]

9

1. 你听说过成员资格运算符吗？

答：

Python 有一个成员资格运算符：in，用于检查一个值是否在序列中，如果在序列中返回 True，否则返回 False。

代码实例：

>>> str = "Amusi"

>>> "Am" in str

True

>>> "si" in str

True

>>> "jj" in str

False

### 动动手

0. 视频中小甲鱼使用 if elif else 在大多数情况下效率要比全部使用 if 要高，但根据一般的统计规律，一个班的成绩一般服从正态分布，也就是说平均成绩一般集中在 70~80 分之间，因此根据统计规律，我们还可以改进下程序以提高效率。

题目备忘：按照100分制，90分以上成绩为A，80到90为B，60到80为C，60以下为D，写一个程序，当用户输入分数，自动转换为ABCD的形式打印。

答： 由题意，只需将60～80分数段的判断条件写在程序最前面即可

score = int(input("Please input a score: "))

if 80 > score >= 60:

print("C")

elif 100 >= score >= 90:

print("A")

elif 90 > score >= 80:

print("B")

elif 60 > score >= 0:

print("D")

else:

print("Error input")

1. Python 的作者在很长一段时间不肯加入三元操作符就是怕跟C语言一样搞出国际乱码大赛，蛋疼的复杂度让初学者望而生畏。不过，如果你一旦搞清楚了三元操作符的使用技巧，或许一些比较复杂的问题反而迎刃而解。

请将以下代码修改为三元操作符实现：

1. x, y, z = 6, 5, 4
2. if x < y:
3. small = x
4. if z < small:
5. small = z
6. elif y < z:
7. small = y
8. else:
9. small = z

**答： 暂无**

## 课时9 了不起的分支和循环3

### 9.1 for循环

**语法**

for 目标 in 表达式:

循环体

**代码实例**

>>> favourite = "Amusi"

>>> for i in favourite:

print(i, end = " ")

A m u s i

>>> member = ["Amusi", "Lucy", "Jing", "JJ"]

>>> for each in member:

print(each, len(member))

Amusi 4

Lucy 4

Jing 4

JJ 4

>>> for each in member:

print(each, len(each))

Amusi 5

Lucy 4

Jing 4

JJ 2

注： Python的for循环相比于C/C++的for循环，更加强大

### 9.2 range()

**语法**

range([start, ] stop[, step=1])

* range有三个参数，其中用中括号括起来的参数表示可选的
* step = 1表示默认值为1
* range()这个BIT的作用是生成一个从start参数的值开始到stop参数的值结束的数字序列

**代码实例**

>>> range(5)

range(0, 5)

>>> list(range(5))

[0, 1, 2, 3, 4]

# version1

>>> for i in range(5):

print(i)

0

1

2

3

# version2

>>> for i in range(2, 9):

print(i)

2

3

4

5

6

7

8

# version3

>>> for i in range(1, 20, 2):

print(i)

1

3

5

7

9

11

13

15

17

19

### 9.3 break和continue

* break: 跳出当前循环体
* Continue： 终止本轮循环，继续下一轮循环

>>> for i in range(10):

if i % 2 != 0:

print(i)

continue

i += 2

print(i)

# output

2

1

4

3

6

5

8

7

10

9

### 测试题

0. 下面的循环会打印多少次"I Love FishC"？

for i in range(0, 10, 2):

    print(“Amusi”)

答：打印5次。因为0,2,4,6,8

>>> for i in range(0, 10, 2):

print("Amusi", i)

Amusi 0

Amusi 2

Amusi 4

Amusi 6

Amusi 8

1. 下面的循环会打印多少次"I Love FishC"？

for i in 5:

    print('I Love FishC')

答： 错误。因为in是“成员资格运算符”

Traceback (most recent call last):

File "<pyshell#39>", line 1, in <module>

for i in 5:

TypeError: 'int' object is not iterable

2. 回顾一下 break 和 continue 在循环中起到的作用？

答：

* break： 跳出当前循环体
* continue: 结束本轮循环，开始新一轮循环

1. 请谈下你对列表的理解？

答： 列表list相当于有序的每个元素的数组

>>> list(range(10))

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

1. 请问 range(10) 生成哪些数？

答：左闭右开。生成0=<n<10范围内的数，即0,1,3,4,5,6,7,8,9

>>> list(range(10))

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

5. 目测以下程序会打印什么？

while True:

    while True:

        break

        print(1)

    print(2)

    break

print(3)

答：

2

3

1. 什么情况下我们要使循环永远为真？

答： 常用于“消息循环”，即不断处理的程序，如接收消息

while True:

循环体

7. 【学会提高代码的效率】你的觉得以下代码效率方面怎样？有没有办法可以大幅度改进(仍然使用while)？

i = 0

string = "AmusiLoveLucy.com"

while i < len(string):

    print(i)

    i += 1

答： 上述代码len调用次数较多，其实只需调用一次即可

i = 0

string = "AmusiLoveLucy.com"

length = len(string)

while i < length:

    print(i)

    i += 1

### 动动手

0. 设计一个验证用户密码程序，用户只有三次机会输入错误，不过如果用户输入的内容中包含"\*"则不计算在内。

程序演示如图：

答：

1. 编写一个程序，求 100~999 之间的所有水仙花数。

注：如果一个 3 位数等于其各位数字的立方和，则称这个数为水仙花数。例如：153 = 1^3 + 5^3 + 3^3，因此 153 就是一个水仙花数。

答：

for i in range(100, 1000):

sum = 0

temp = i

while temp:

sum = sum + (temp%10) \*\* 3

temp //= 10         # 注意这里要使用地板除哦~

if sum == i:

print(i)

2. 三色球问题

有红、黄、蓝三种颜色的求，其中红球 3 个，黄球 3 个，绿球 6 个。先将这 12 个球混合放在一个盒子中，从中任意摸出 8 个球，编程计算摸出球的各种颜色搭配。

答：

print("red\tyellow\tblur")

for red in range(1,4):

for yellow in range(1, 4):

for green in range(1, 7):

if 8 == red + yellow + green:

print(red,"\t",yellow,"\t",green)

1. 请写下这一节课你学习到的内容：格式不限，回忆并复述是加强记忆的好方式！

答：

* for循环体
* Break知识点
* Continue知识点
* range知识点

# 章节4

## 课时10 列表：一个打了激素的数组1

### 10.1 创建一个普通的列表

**代码实例**

>>> member = ["Amusi", "Lucy", "Jing", "Nancy"]

>>> member

['Amusi', 'Lucy', 'Jing', 'Nancy']

>>> number = ["1, 2, 3, 4, 5"]

>>> number

['1, 2, 3, 4, 5']

>>> min = [1, "2", "Amusi", 3.1415926, 9]

>>> min

[1, '2', 'Amusi', 3.1415926, 9]

>>> empty = []

>>> empty

[]

注：C/C++的数组元素必须为统一的类型

### 10.2向列表添加元素

* append extend insert

**1. append()函数**

在list尾部添加一个元素。只有一个参数，为元素

代码实例

>>> member = ["Amusi", "Lucy", "Jing", "Nancy"]

>>> member

['Amusi', 'Lucy', 'Jing', 'Nancy']

>>> member.append("Jason")

>>> member

['Amusi', 'Lucy', 'Jing', 'Nancy', 'Jason']

>>> len(member)

5

**2. extend()函数**

在list尾部添加一个子list。只有一个参数，为列表

代码实例

>>> member.extend(["Jackie", "Trista"])

>>> member

['Amusi', 'Lucy', 'Jing', 'Nancy', 'Jason', 'Jackie', 'Trista']

>>> len(member)

7

>>> member = []

>>> member.extend("Jason")

>>> member

['J', 'a', 's', 'o', 'n']

**3. insert()函数**

在list中某个位置添加一个元素。有两个参数，参数1位置，参数2元素

代码实例

>>> member.insert(0, "Marry")

>>> member

['Marry', 'Amusi', 'Lucy', 'Jing', 'Nancy', 'Jason', 'Jackie', 'Trista']

>>> len(member)

8

### 测试题

0. 列表都可以存放一些什么东西？

答： 整型，浮点型，字符串，布尔型等类型的元素

1. 向列表增加元素有哪些方法？

答： append() extend() insert()

2. append() 方法和 extend() 方法都是向列表的末尾增加元素，请问它们有什么区别？

答：

* append()只能添加单个元素
* extend()可以添加多个元素，即子列表

1. member.append([“Amusi”, “Lucy”]) 和 member.extend(“Amusi”, “Lucy”]) 实现的效果一样吗？

答： 不一样。前者是向列表中添加[“Amusi”, “Lucy”]这单个元素，后者是向列表中添加了子列表，即两个元素“Amusi”和“Lucy”

>>> member = []

>>> member.append(["Amusi", "Lucy"])

>>> member

[['Amusi', 'Lucy']]

>>> member.extend(["Amusi", "Lucy"])

>>> member

[['Amusi', 'Lucy'], 'Amusi', 'Lucy']

4. 有列表 name = [“A”, “u”, “s”, “i”]，如果小甲鱼想要在元素 ”A” 和 “u” 之间插入元素 “m”，应该使用什么方法来插入？

答： 使用insert()函数

>>> name = ["A", "u", "s", "i"]

>>> name

['A', 'u', 's', 'i']

>>> name.insert(1, "m")

>>> name

['A', 'm', 'u', 's', 'i']

### 动动手

**0. 自己动手试试看，并分析在这种情况下，向列表添加数据应当采用哪种方法比较好？**

假设给定以下列表：

member = [“Amusi”, “Lucy”, “Nancy”, “Jackie”, “Trista”]

要求将列表修改为：

member = [“Amusi”, 98, “Lucy”, 100, “Nancy”, 120, “Jackie”, 88, “Trista”, 99]

方法一：使用 insert() 和 append() 方法修改列表。

方法二：重新创建一个同名字的列表覆盖

答： 当然方法一好，因为方法二较为暴力

>>> member = ["Amusi", "Lucy", "Nancy", "Jackie", "Trista"]

>>> member

['Amusi', 'Lucy', 'Nancy', 'Jackie', 'Trista']

>>> member.insert(1, 98)

>>> member.insert(3, 100)

>>> member.insert(5, 120)

>>> member.insert(7, 88)

>>> member.append(99)

>>> member

['Amusi', 98, 'Lucy', 100, 'Nancy', 120, 'Jackie', 88, 'Trista', 99]

**1. 利用 for 循环打印上边 member 列表中的每个内容，如图：**

>>> for each in member:

print(each)

Amusi

98

Lucy

100

Nancy

120

Jackie

88

Trista

99

1. **请写下这一节课你学习到的内容：格式不限，回忆并复述是加强记忆的好方式！**

**答：**

* append(): 在列表的尾部添加一个元素
* extend(): 在列表的尾部添加一个子list

>>> member = []

>>> member.extend("Jason")

>>> member

['J', 'a', 's', 'o', 'n']

* insert(): 在列表的某个位置处添加一个元素

## 课时11 列表：一个打了激素的数组2

### 11.1从列表中获取元素

跟数组一样，我们可以通过元素的索引值(index)从列表中获取单个元素，需要用”[]”中括号包含index。

注：列表索引值是从0开始的

**代码实例**

>>> member = ["Amusi", "Lucy", "Nancy", "Jackie", "Trista"]

>>> member[0]

'Amusi'

>>> member[1]

'Lucy'

>>> temp = member[0]

>>> member[0] = member[1]

>>> member[1] = temp

>>> member

['Lucy', 'Amusi', 'Nancy', 'Jackie', 'Trista']

### 11.2 从列表中删除元素

* Remove()

1. **remove()**

删除列表中存在的元素，参数1元素

>>> member

['Lucy', 'Amusi', 'Nancy', 'Jackie', 'Trista']

>>> member.remove("Nancy")

>>> member

['Lucy', 'Amusi', 'Jackie', 'Trista']

1. **del语句**

删除列表中指定位置的元素

>>> member

['Lucy', 'Amusi', 'Jackie', 'Trista']

>>> del member[3]

>>> member

['Lucy', 'Amusi', 'Jackie']

1. **pop()**

删除列表尾部的元素，也可以删除指定位置的元素

语法

list名.pop(\*)

list名.pop(位置)

**代码实例**

>>> member

['Lucy', 'Amusi', 'Jackie']

# 删除列表尾部的元素

>>> member.pop()

'Jackie'

>>> member

['Lucy', 'Amusi']

#

>>> member

['Lucy', 'Amusi', 'Love', 'Jie']

# 删除列表指定位置的元素

>>> member.pop(1)

'Amusi'

>>> member

['Lucy', 'Love', 'Jie']

### 11.3 列表分片(slice)

利用索引值，每次我们可以从列表中获取一个元素，但我们总是贪心的，如果一次性需要获得多个元素，有没有办法实现呢？利用列表分片slice，我们可以简单实现这个要求。

**语法**

list名[\*:\*]

**代码实例**

>>> member

['Lucy', 'Love', 'Jie', 'Amusi']

>>> member[1:3]

['Love', 'Jie']

>>> member[:4]

['Lucy', 'Love', 'Jie', 'Amusi']

>>> member[:]

['Lucy', 'Love', 'Jie', 'Amusi']

>>> member[:2]

['Lucy', 'Love']

>>> member[:3]

['Lucy', 'Love', 'Jie']

>>> member[1:]

['Love', 'Jie', 'Amusi']

>>> member2 = member[:]

>>> member2

['Lucy', 'Love', 'Jie', 'Amusi']

## 课时12 列表：一个打了激素的数组3

### 12.1 比较操作符

使用比较操作符（>,<,==,>=,<=,!=）来比较两个列表。默认是先比较两个列表中各个首元素的值（会自动转换成ASCII进行比较），如果两者相同，继续比较两个列表中下一个对应元素的值。

**语法**

list名1 比较操作符 list名2

返回值：True/False

**实例代码**

# 比较整型

>>> list1 = [123]

>>> list2 = [234]

>>> list1 > list2

False

>>> list2 > list1

True

>>> list1 = [123, 456]

>>> list2 = [123, 456]

>>> list1 > list2

False

>>> list1 >= list2

True

# 比较字符串

>>> list6 = ["Amusi", "Lucy"]

>>> list7 = ["amusi", "lucy"]

>>> list6 < list7

True # 因为a的ASCII（91），所以大于A的ASCII码（650

>>> list6 > list7

False

### 12.2 逻辑操作符

使用逻辑操作符（and, or, not）来判断两个条件。

**语法**

条件1 逻辑操作符 条件2

返回值：True/False

**实例代码**

>>> list1 = [123, 456]

>>> list2 = [123, 456]

>>> list3 = [123, 456]

>>> (list1 <= list2) and (list1 == list3)

True

### 12.3 连接操作符

使用连接操作符(+)来将两个列表连接。

**语法**

列表名1 + 列表名2 + 列表名3 + ... + 列表名n

**实例代码**

>>> list1 = ["Amusi"]

>>> list2 = ["Lucy"]

>>> list3 = [100]

>>> list4 = [3.14]

>>> list5 = list1 + list2 +list3 +list4

>>> list5

['Amusi', 'Lucy', 100, 3.14]

### 12.4 重复操作符

使用重复操作符(\*)实现列表的复制。

**语法**

列表名 \* 数值

列表名 \*= 数值

**代码实例**

>>> list1 = ["Amusi"]

>>> list1 \* 3

['Amusi', 'Amusi', 'Amusi']

>>> list1 \*= 3

>>> list1

['Amusi', 'Amusi', 'Amusi']

### 12.5 成员关系操作符

使用成员关系操作符(in)实现元素是否存在的判断。

**语法**

元素名 in 列表

**代码实例**

>>> list1

['Amusi', 'Amusi', 'Amusi']

>>> "Amusi" in list1

True

>>> 123 in list1

False

>>> 123 not in list1

True

# 列表中的列表

>>> list2 = [123, ["Amusi", "lucy"], 456]

>>> 123 in list2

True

>>> "Amusi" in list2

False

>>> "Amusi" in list2[1]

True

>>> list2[1][1]

'lucy'

>>> list2[1][0]

'Amusi'

### 12.6 列表的BIF

使用dir(list)查看列表所有的BIF

>>> dir(list)

['\_\_add\_\_', '\_\_class\_\_', '\_\_contains\_\_', '\_\_delattr\_\_', '\_\_dir\_\_', '\_\_doc\_\_', '\_\_eq\_\_', '\_\_format\_\_', '\_\_ge\_\_', '\_\_getattribute\_\_', '\_\_getitem\_\_', '\_\_getnewargs\_\_', '\_\_gt\_\_', '\_\_hash\_\_', '\_\_init\_\_', '\_\_init\_subclass\_\_', '\_\_iter\_\_', '\_\_le\_\_', '\_\_len\_\_', '\_\_lt\_\_', '\_\_mod\_\_', '\_\_mul\_\_', '\_\_ne\_\_', '\_\_new\_\_', '\_\_reduce\_\_', '\_\_reduce\_ex\_\_', '\_\_repr\_\_', '\_\_rmod\_\_', '\_\_rmul\_\_', '\_\_setattr\_\_', '\_\_sizeof\_\_', '\_\_str\_\_', '\_\_subclasshook\_\_', 'capitalize', 'casefold', 'center', 'count', 'encode', 'endswith', 'expandtabs', 'find', 'format', 'format\_map', 'index', 'isalnum', 'isalpha', 'isdecimal', 'isdigit', 'isidentifier', 'islower', 'isnumeric', 'isprintable', 'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'maketrans', 'partition', 'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines', 'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper', 'zfill']

* count函数： 计算某元素在列表中出现的次数

**语法**

list名.count(\*)

**代码实例**

>>> list1 = [123,456]

>>> list1 \*= 3

>>> list1 \*= 5

>>> list1

[123, 456, 123, 456, 123, 456, 123, 456, 123, 456, 123, 456, 123, 456, 123, 456, 123, 456, 123, 456, 123, 456, 123, 456, 123, 456, 123, 456, 123, 456]

>>> list1.count(123)

15

* index函数： 计算某元素在列表/某段列表中出现的位置

**语法**

list名.index(\*)

list名.index(\*, begin, end)

**代码实例**

>>> list1

[123, 456, 123, 456, 123, 456, 123, 456, 123, 456, 123, 456, 123, 456, 123, 456, 123, 456, 123, 456, 123, 456, 123, 456, 123, 456, 123, 456, 123, 456]

>>> list1.count(123)

15

>>> list1.index(123)

0

>>> list1.index(123, 3, 7)

4

* reverse函数： 将列表中所有的元素逆序重新排列

**语法**

list名.reverse()

**代码实例**

>>> list1

[123, 456, 123, 456, 123, 456, 123, 456, 123, 456, 123, 456, 123, 456, 123, 456, 123, 456, 123, 456, 123, 456, 123, 456, 123, 456, 123, 456, 123, 456]

>>> list1.reverse()

>>> list1

[456, 123, 456, 123, 456, 123, 456, 123, 456, 123, 456, 123, 456, 123, 456, 123, 456, 123, 456, 123, 456, 123, 456, 123, 456, 123, 456, 123, 456, 123]

* sort函数: 将列表排序。默认从小到大

**语法**

list名.sort()

sort(func, key)

**代码实例**

>>> list1 = [9, 10, 5, 1, 4, 6, 5, 8]

# 从小到大

>>> list1.sort()

>>> list1

[1, 4, 5, 5, 6, 8, 9, 10]

# 从大到小

>>> list1.sort(reverse = True)

>>> list1

[10, 9, 8, 6, 5, 5, 4, 1]

### 12.6 分片的拷贝

>>> list1 = [10, 9, 8, 6, 5, 5, 4, 1]

>>> list2 = list1[:] # 分片

>>> list3 = list1 # 直接赋值

>>> list1

[10, 9, 8, 6, 5, 5, 4, 1]

>>> list2

[10, 9, 8, 6, 5, 5, 4, 1]

>>> list3

[10, 9, 8, 6, 5, 5, 4, 1]

>>> list1.sort()

>>> list1

[1, 4, 5, 5, 6, 8, 9, 10]

>>> list2

[10, 9, 8, 6, 5, 5, 4, 1] # 不变

>>> list3

[1, 4, 5, 5, 6, 8, 9, 10] # 变化

注：可知分片是实现列表的拷贝，即不再依赖源列表，而是开辟一个新的内存空间，里面存放的是拷贝的列表元素。而直接赋值是指向源列表的内存空间。

list1

list2

List3

10, 9, 8, 6, 5, 5, 4, 1

10, 9, 8, 6, 5, 5, 4, 1

分片和直接赋值

### 测试题

1. 请问如何将下边这个列表的“Amusi”修改为”Lucy”？

list1 = [1, [1, 2, ["Amusi"]], 3, 5, 8, 13, 18]

 答：

>>> list1 = [1, [1, 2, ["Amusi"]], 3, 5, 8, 13, 18]

>>> list1[1][2] = "Lucy"

>>> list1

[1, [1, 2, 'Lucy'], 3, 5, 8, 13, 18]

1. 要对一个列表进行顺序排序，请问使用什么方法？

答：使用sort函数

>>> list1 = [1, 4, 10, 199, 2, 4, 16, 77, 256, 88, 55, 21]

>>> list1.sort()

>>> list1

[1, 2, 4, 4, 10, 16, 21, 55, 77, 88, 199, 256]

>>> list1.sort(reverse = True)

>>> list1

[256, 199, 88, 77, 55, 21, 16, 10, 4, 4, 2, 1]

>>> list1.sort(reverse = True)

>>> list1

[256, 199, 88, 77, 55, 21, 16, 10, 4, 4, 2, 1]

>>> list1.sort()

>>> list1

[1, 2, 4, 4, 10, 16, 21, 55, 77, 88, 199, 256]

  注：可知sort()默认是从小到大排序，而sort(reverse = True)并不是逆序排列而是从大到小排列

1. 要对一个列表进行逆序排序，请问使用什么方法？

答： 逆序排序可使用reverse()函数

>>> list1 = [1, 4, 10, 199, 2, 4, 16, 77, 256, 88, 55, 21]

>>> list1

[1, 4, 10, 199, 2, 4, 16, 77, 256, 88, 55, 21]

>>> list1.reverse()

>>> list1

[21, 55, 88, 256, 77, 16, 4, 2, 199, 10, 4, 1]

4. 列表还有两个内置方法没给大家介绍，不过聪明的你应该可以自己摸索使用的门道吧：copy() 和 clear()

答：

* copy(): 列表的拷贝

>>> list1 = [1, 4, 10, 199, 2, 4, 16, 77, 256, 88, 55, 21]

>>> list2 = list1.copy()

>>> list2

[1, 4, 10, 199, 2, 4, 16, 77, 256, 88, 55, 21]

* clear(): 列表的清除

>>> list1 = [1, 4, 10, 199, 2, 4, 16, 77, 256, 88, 55, 21]

>>> list1

[1, 4, 10, 199, 2, 4, 16, 77, 256, 88, 55, 21]

>>> list1.clear()

>>> list1

[]

1. 你有听说过列表推导式或列表解析吗？

答： 没听过？！没关系，我们现场来学习一下吧，看表达式：

1. >>> [ i\*i for i in range(10) ]

你觉得会打印什么内容？

答：

[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

**列表推导式**（List comprehensions）也叫列表解析，灵感取自函数式编程语言 Haskell。Ta 是一个非常有用和灵活的工具，可以用来动态的创建列表，语法如：

[有关A的表达式 for A in B]

例如

1. >>> list1 = [x\*\*2 for x in range(10)]
2. >>> list1
3. [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

相当于

1. list1 = []
2. for x in range(10):
3. list1.append(x\*\*2)

问题：请先在 IDLE 中获得下边列表的结果，并按照上方例子把列表推导式还原出来。

1. >>> list1 = [(x, y) for x in range(10) for y in range(10) if x%2==0 if y%2!=0]

# output

>>> list1

[(0, 1), (0, 3), (0, 5), (0, 7), (0, 9), (2, 1), (2, 3), (2, 5), (2, 7), (2, 9), (4, 1), (4, 3), (4, 5), (4, 7), (4, 9), (6, 1), (6, 3), (6, 5), (6, 7), (6, 9), (8, 1), (8, 3), (8, 5), (8, 7), (8, 9)]

# 列表推导式

>>> list1 = []

>>> for x in range(10):

for y in range(10):

if (x % 2 == 0) and (y % 2 != 0):

list1.append((x, y))

>>> list1

[(0, 1), (0, 3), (0, 5), (0, 7), (0, 9), (2, 1), (2, 3), (2, 5), (2, 7), (2, 9), (4, 1), (4, 3), (4, 5), (4, 7), (4, 9), (6, 1), (6, 3), (6, 5), (6, 7), (6, 9), (8, 1), (8, 3), (8, 5), (8, 7), (8, 9)]

1. 请写下这一节课你学习到的内容：格式不限，回忆并复述是加强记忆的好方式！

* count(): 计算某元素在列表中出现的次数
* index(): 计算某元素在列表中出现的位置
* sort(): 将列表进行从小达到排序
* reverse(): 将列表逆序排列
* copy(): 将列表拷贝
* clear(): 将列表所有元素清除
* 分片slice: [:]
* 列表推导式: 简化地动态地创建列表

## 课时13 元组：带上了枷锁的列表

元组语法：

tuple\_name = \*,

如：

tuple1 = (1, 2, 3, 4, 5, 6, 7, 8)

tuple2 = 1, 2, 3

注：逗号是元组的重要标志符。用逗号隔开元素

### 13.1 元组与列表的区分

* 元组和列表使用上非常相似，但还是有很大不同
* 列表可以任意更改元素
* 元组是不可改变的，如插入、删除

### 13.2 创建和访问一个列表

* 创建

元组语法：

tuple\_name = \*,

如：

tuple1 = (1, 2, 3, 4, 5, 6, 7, 8)

tuple2 = 1, 2, 3

注： 逗号是元组的重要标志符。用逗号隔开元素

* 访问

Tuple\_name[\*]

注： 和list一致，根据索引访问

**代码实例**

# 创建: 有括号，且有逗号

>>> tuple1 = (1, 2, 3, 4, 5, 6, 7, 8)

>>> tuple1

(1, 2, 3, 4, 5, 6, 7, 8)

>>> type(tuple1)

<class 'tuple'>

# 访问

>>> tuple1[2]

3

>>> tuple1[5:]

(6, 7, 8)

>>> tuple1[:5]

(1, 2, 3, 4, 5)

>>> tuple2 = tuple1[:]

>>> tuple2

(1, 2, 3, 4, 5, 6, 7, 8)

# 不可修改元组的元素

>>> tuple1[1] = 9

Traceback (most recent call last):

File "<pyshell#7>", line 1, in <module>

tuple1[1] = 9

TypeError: 'tuple' object does not support item assignment

# 无括号，但有逗号

>>> tuple3 = 2, 3, 4, 7, 9

>>> type(tuple3)

<class 'tuple'>

# 有括号，无逗号

>>> tuple4 = (1)

>>> type(tuple4)

<class 'int'>

# 有括号，有逗号

>>> tuple4 = (1,)

>>> type(tuple4)

<class 'tuple'>

# 创建空的元组

>>> tuple5 = ()

>>> type(tuple5)

<class 'tuple'>

>>> 8 \* (8)

64

>>> 8 \* (8,)

(8, 8, 8, 8, 8, 8, 8, 8)

### 13.2 更新和删除一个元组

* 更新

>>> temp =("Amusi", "Lucy", "Jackie", "Trista")

>>> temp = temp[:2] + ("And",) + temp[2:]

>>> temp

('Amusi', 'Lucy', 'And', 'Jackie', 'Trista')

注：元组的更新是以切片的方式在原来的元组中拼接插入新的元组

* 删除

del tuple\_name

>>> temp =("Amusi", "Lucy", "Jackie", "Trista")

>>> temp

('Amusi', 'Lucy', 'And', 'Jackie', 'Trista')

# 一般不需要用del删除元组，因为Python有垃圾回收机制

>>> del temp

### 测试题

1. 请用一句话描述什么是列表？再用一句话描述什么是元组？

答：

* 列表: 可任意修改（插入、删除）元素的容器
* 元组: 封闭的列表，一旦定义，就不可改变（不能添加、删除或修改）

1. 什么情况下你需要使用元组而不是列表？

答： 当需要数据不能被修改时，应当使用元组

1. 当元组和列表掉下水，你会救谁？

答： 都救。但列表更加常用

1. 创建一个元组，什么情况下逗号和小括号必须同时存在，缺一不可？

答： 当括号里只有一个元素时

>>> tuple1 = (1)

>>> type(tuple1)

<class 'int'>

>>> tuple1 = (1,)

>>> type(tuple1)

<class 'tuple'>

1. x, y, z = 1, 2, 3 请问x, y, z是元组吗？

答： 不是元组，而是整型

>>> x, y, z = 1, 2, 3

   >>> type(x)

<class 'int'>

# 元组的正确打开方式

>>> x = 1, 2, 3

>>> type(x)

<class 'tuple'>

5. 请写出以下情景中应该使用列表还是元组来保存数据：

1) 游戏中角色的属性： 列表

2) 你的身份证信息： 元组

3) 论坛的会员： 列表

4) 团队合作开发程序，传递给一个你并不了解具体实现的函数的参数： 元组

5) 航天火箭各个组件的具体配置参数： 元组

6) NASA系统中记录已经发现的行星数据： 列表

7. 上节课我们通过课后作业的形式学习到了“列表推导式”，那请问如果我把中括号改为小括号，会不会得到“元组推导式”呢？

答: 不会。因为元组的元素是不可修改的，不允许插入、删除等操作

8. 请写下这一节课你学习到的内容：格式不限，回忆并复述是加强记忆的好方式！

答：

* 元组的定义和操作

## 课时14 字符串: 各种奇葩的内置方法

Reference: [字符串的方法及注释](http://bbs.fishc.com/forum.php?mod=viewthread&tid=38992&extra=page%3D1%26filter%3Dtypeid%26typeid%3D403)

表14.1 字符串的方法及注释

|  |  |
| --- | --- |
| capitalize() | 把字符串的第一个字符改成大写 |
| casefold() | 把整个字符串的所有字符改成小写 |
| center(width) | 把字符串居中，并使用空格填充至长度width的新字符串 |
| count(sub[, start [, end]]) | 返回sub在字符串里边出现的次数，start和end参数表示范围，可选。 |
| encode(encoding=’utf-8’,  error=’strict’) | 以encoding指定的编码格式对字符串进行编码 |
| endswith(sub[, start[,end]]) | 检查字符串是否以sub子字符串结束，如果是返回True，否则返回False。Start和end参数表示范围，可选 |
| enpandtabs([tabsize=8]) | 把字符串中的tab符号(\t)转换成空格，如果不指定参数，默认的空格数是tabsize=8 |
| find(sub[, start[, end]]) | 检测sub子字符串是否包含在字符串中，如果有则返回索引值，否则返回-1，start和end参数表示范围，可选。 |
| index(sub[, start[, end]]) | 跟find方法一致，不过如果sub不在字符串中会产生一个异常（exception）。 |
| isalnum() | 如果字符串至少有一个字符，并且所有字符都是字母或数字则返回True，否则返回False |
| isalpha() | 如果字符串至少有一个字符，并且所有字符都是字母则返回True，否则返回False |
| isdecimal() | 如果字符串只包含十进制数字则返回True，否则返回False |
| isdigit() | 如果字符串中只包含数字则返回True，否则返回False |
| islower() | 如果字符串中至少包含一个区分大小写的字符，并且这些字符都是小写，则返回True，否则返回False |
| isnumeric() | 如果字符串中只包含数字字符，则返回True，否则，返回False |
| isspace() | 如果字符串中只包含空格，则返回True，否则返回False |
| istitle() | 如果字符串是标题化（所有单词都是以大写开始，其余字母均小写），则返回True，否则返回False |
| isupper() | 如果字符串中至少包含一个区分大小写的字符，并且这些字符均是大写，则返回True，否则返回False |
| join(sub) | 以字符串作为分隔符，插入到sub中所有的字符之间 |
| ljust(width) | 返回一个左对齐的字符串，并使用空格填充至长度为width的新字符串 |
| lower() | 转换字符串中的所有大写字符为小写 |
| lstrip() | 去掉字符串左边的所有空格 |
| partition(sub) | 找到子字符串sub，把字符串分成一个3元组(pre-sub,sub,fol\_sub)，如果字符串中不包含sub则返回（“原字符串,””,”） |
| replace(old, new[, count]) | 把字符串中的old子字符串替换成new子字符串，如果count指定，则替换不超过count次。可选 |
| rfind(sub[, start[, end]]) | 类似于find()方法，不过是从右边开始查找 |
| rindex(sub[, start[, end]]) | 类似于index()方法，不过是从右边开始查找 |
| rjust() | 返回一个右对齐的字符串，并使用空格填充至长度为width的新字符串 |
| rpartition() | 类似于partition()方法，不过是从右边开始查找 |
| rstrip() | 删除字符串末尾的空格 |
| split(sep=None, maxsplit=-1) | 不带参数默认是以空格为分割符切片字符串，如果maxsplit参数有设置，则仅分隔maxsplit个子字符串，返回切片后的子字符串拼接的列表 |
| splitnes([keepends]) | 按照’\n’分割，返回一个包含各行作为元素的列表，如果keepends参数指定，则返回前keepends行 |
| startswith(sub[, start[, end]]) | 检查字符串是否以prefix开头，是则返回True，否则返回False。 start和end参数可以指定范围检查，可选 |
| strip([chars]) | 删除字符串前边和后边所有的空格，chars参数可以定制删除的字符，可选 |
| swapcase() | 翻转字符串中的大小写 |
| title() | 返回标题化(所有的单词都是以大写开始，其余字母均小写)的字符串 |
| translate(table) | 根据table规则(可以由str.maketrans(‘a’,’b’)定制)转换字符串中的字符 |
| upper() | 将字符串中所有小写字符转换成大写 |
| zfill(width) | 返回长度为width的字符串，原字符串右对齐，前边用0填充 |

注：中括号[]表示可选

* capitalize(): 把字符串的第一个字符改成大写

>>> str1 = "amusi"

>>> str1

'amusi'

>>> str1.capitalize()

'Amusi'

>>> str1

* casefold(): 把整个字符串的所有字符改成小写

>>> str1 = "AMUsi"

>>> str1.casefold()

'amusi'

* center(width): 把字符串居中，并使用空格填充至长度width的新字符串

>>> str1 = "AMUsi"

>>> str1.center(40)

' AMUsi '

* count(sub[, start [, end]]): 返回sub在字符串里边出现的次数，start和end参数表示范围，可选。

>>> str1

'AMUsi'

>>> str1.count("i")

1

* endswith(sub[, start[,end]]): 检查字符串是否以sub子字符串结束，如果是返回True，否则返回False。Start和end参数表示范围，可选

>>> str1

'AMUsi'

>>> str1.endswith("si")

True

* expandtabs([tabsize=8]): 把字符串中的tab符号(\t)转换成空格，如果不指定参数，默认的空格数是tabsize=8

>>> str1 = "I\tlobe\t Lucy!"

>>> str1

'I\tlobe\t Lucy!'

>>> str1.expandtabs()

'I lobe Lucy!'

* find(sub[, start[, end]]): 检测sub子字符串是否包含在字符串中，如果有则返回索引值，否则返回-1，start和end参数表示范围，可选。

>>> str1 = "I love Lucy!"

>>> str1.find("L")

7

* join(sub): 以字符串作为分隔符，插入到sub中所有的字符之间

>>> str1 = "Amusi"

>>> str1.join("123")

'1Amusi2Amusi3'

* lstrip(): 去掉字符串左边的所有空格

>>> str1 = " I love you!"

>>> str1.lstrip()

'I love you!'

* replace(old, new[, count]): 把字符串中的old子字符串替换成new子字符串，如果count指定，则替换不超过count次。可选

>>> str1 = "I love you!"

>>> str1.replace("you", "Lucy")

'I love Lucy!'

* split(sep=None, maxsplit=-1): 不带参数默认是以空格为分割符切片字符串，如果maxsplit参数有设置，则仅分隔maxsplit个子字符串，返回切片后的子字符串拼接的列表

>>> str1 = "I love Lucy"

>>> str1.split()

['I', 'love', 'Lucy']

* swapcase(): 翻转字符串中的大小写

>>> str1 = "I love Lucy!"

>>> str1.swapcase()

'i LOVE lUCY!'

* title(): 返回标题化(所有的单词都是以大写开始，其余字母均小写)的字符串

>>> str1 = "i love lucy!"

>>> str1.title()

'I Love Lucy!'

* translate(table)： 根据table规则(可以由str.maketrans(‘a’,’b’)定制)转换字符串中的字符

>>> str1 = "aaaaabbbbaaacccc0"

>>> str1.translate(str.maketrans("a", "A"))

'AAAAAbbbbAAAcccc0'

* upper(): 转换字符串中的所有小写字符为大写

>>> str1 = "i love lucy"

>>> str1.upper()

'I LOVE LUCY'

* zfill(width)： 返回长度为width的字符串，原字符串右对齐，前边用0填充

>>> str1 = "Amusi"

>>> str1.zfill(10)

'00000Amusi'

### 测试题

0. 还记得如何定义一个跨越多行的字符串吗（请至少写出两种实现的方法）？

答： 使用三引号字符串(“””\*”””)，或者在每句待换行的最后加上换行符(“\n”)

1. 三引号字符串通常我们用于做什么使用？

答： 跨行注释

1. file1 = open(“C:\windows\temp\readme.txt”, “r”) 表示以只读方式打开“C:\windows\temp\readme.txt”这个文本文件，但事实上这个语句会报错，知道为什么吗？你会如何修改？

答： 因为字符串中存在”\”转移字符。

方法一：在每个”\”前面多加一个”\”。如“C:\\windows\\temp\\readme.txt”

方法二：使用原始字符串，即在字符串前面加上”r”。如r”C:\windows\temp\readme.txt”

1. 有字符串：str1 = "http://www.amusi.com/"，请问如何提取出子字符串：“amusi.com”

答：

>>> str1 = "http://www.amusi.com/"

    >>> str2 = str1[11:20]

>>> str2

'amusi.com'

4. 如果使用负数作为索引值进行分片操作，按照第三题的要求你能够正确目测出结果吗？

答： ？？？

5. 还是第三题那个字符串，请问下边语句会显示什么内容？

>>> str1[5:-10]

答：

>>> str1 = "http://www.amusi.com/"

>>> str1[5:-10]

'//www.'

1. 据说只有智商高于150的鱼油才能解开这个字符串（还原为有意义的字符串）：str1 = “i2sl54ovvvb4e3bferi32s56h;$c43.sfc67o0cm99”

答： ilovefishc.com

>>> str1 ="i2sl54ovvvb4e3bferi32s56h;$c43.sfc67o0cm99"

>>> count = 1

>>> str2 = []

>>> for i in str1:

if(count % 3 ==1 ):

str2.append(i)

count += 1

>>> str2

['i', 'l', 'o', 'v', 'e', 'f', 'i', 's', 'h', 'c', '.', 'c', 'o', 'm']

### 动动手

0. 请写一个密码安全性检查的脚本代码：check.py

# 密码安全性检查代码

#

# 低级密码要求：

#   1. 密码由单纯的数字或字母组成

#   2. 密码长度小于等于8位

#

# 中级密码要求：

#   1. 密码必须由数字、字母或特殊字符（仅限：~!@#$%^&\*()\_=-/,.?<>;:[]{}|\）任意两种组合

# 2. 密码只能由数字开头

#   3. 密码长度不能低于8位

#

# 高级密码要求：

#   1. 密码必须由数字、字母及特殊字符（仅限：~!@#$%^&\*()\_=-/,.?<>;:[]{}|\）三种组合

#   2. 密码只能由字母开头

#   3. 密码长度不能低于16位

str1 = input("please input passward: ")

if str1.isalnum() and len(str1) <= 8:

print("低级密码")

elif str1.isalnum() and str1[0].isdigit() and len(str1) >= 8:

print("中级密码")

elif str1.isalnum() and str1[0].isalpha() and len(str1) >= 16:

print("高级密码")

else:

print("Not sure!")

1. 请写下这一节课你学习到的内容：格式不限，回忆并复述是加强记忆的好方式！

答： 字符串的内建函数

* count()
* upper()
* lower()
* isdigit()
* replace()
* title()
* find()
* capitalize()

## 课时15 字符串: 格式化

**format位置参数语法**

“{1},{2},...,{N}”.format(“\*1”, “\*2”, ...,“\*n”)

**实例代码**

>>> "{0} love {1}.{2}".format("I", "Lucy", "!")

'I love Lucy.!'

>>> "{0:.1f}{1}".format(27.658, "GB")

'27.7GB'

**format关键字参数语法**

“{variable}{variable1}...{variableN}”.format(variable=“\*”,...variablenN=”\*”)

**实例代码**

>>> "{a} love {b} {c}".format(a="I", b="Lucy", c="!")

'I love Lucy !'

### 15.1 字符串格式化符号含义

**语法**

“%\*” % \*

表15.1 字符串格式化符号含义

|  |  |
| --- | --- |
| 符   号 | 说     明 |
| %c | 格式化字符及其ASCII码 |
| %s | 格式化字符串 |
| %d | 格式化整数 |
| %o | 格式化无符号八进制数 |
| %x | 格式化无符号十六进制数 |
| %X | 格式化无符号十六进制数（大写） |
| %f | 格式化定点数，可指定小数点后的精度 |
| %e | 用科学计数法格式化定点数 |
| %g | 根据值的大小决定使用%f或者%e |
| %G | 根据值的大小决定使用%F或者%E |

**代码实例**

# %c

>>> print("%c" % 97)

a

>>> "%c" "%c" "%c" % (97, 98, 99)

'abc'

# %s

>>> print("%s" % "I love Lucy!")

I love Lucy!

# %d

>>> print("%d= %d + %d" % (6 + 8, 6, 8))

14= 6 + 8

# %o

>>> print("%o" % 10)

12

# %f

>>> print("%f" % 123.0)

123.000000

# %e

>>> print("%e" % 123.456)

1.234560e+02

### 15.2 格式化操作符辅助命令

**语法**

“%\*” % \*

表15.2 格式化操作符辅助命令

|  |  |
| --- | --- |
| 符   号 | 说     明 |
| m.n | m是显示的最小总宽度，n是小数点后的位数 |
| - | 用于左对齐 |
| + | 在正数前面显示加号 |
| # | 在八进制数前面显示’0o’，在十六进制前面显示’0x’或’0X’ |
| 0 | 显示的数字前面填充’0’取代空格 |

**代码实例**

# m.n

>>> print("%.1f" % 234.567)

234.6

>>> print("%10.f" % 5)

5

# +

>>> print("%+d" % 100)

+100

# #

>>> print("%#o" % 10)

0o12 # 八进制

>>> print("%#x" % 10)

0xa # 十六进制

# 0

>>> print("%010d" % 5)

0000000005

### 15.3 Python转义字符及其含义

**语法**

\\*

图15.3 Python转义字符及其含义

|  |  |
| --- | --- |
| 符   号 | 说     明 |
| \’ | 单引号 |
| \” | 双引号 |
| \a | 发出系统响铃声 |
| \b | 退格符 |
| \n | 换行符 |
| \t | 横向制表符 |
| \v | 纵向制表符 |
| \r | 回车符 |
| \f | 换页符 |
| \o | 八进制代表的字符 |
| \x | 十六进制代表的字符 |
| \0 | 表示一个空字符 |
| \\ | 反斜杠 |

**代码实例**

>>> print("I\tLove\tLucy!")

I Love Lucy!

### 练习题

0. 根据说明填写相应的字符串格式化符号

|  |  |
| --- | --- |
| 符   号 | 说     明 |
| %c | 格式化字符及其ASCII码 |
| %s | 格式化字符串 |
| %d | 格式化整数 |
| %o | 格式化无符号八进制数 |
| %x | 格式化无符号十六进制数 |
| %X | 格式化无符号十六进制数（大写） |
| %f | 格式化定点数，可指定小数点后的精度 |
| %e | 用科学计数法格式化定点数 |
| %g | 根据值的大小决定使用%f或者%e |
| %G | 根据值的大小决定使用%F或者%E |

1. 请问以下这行代码会打印什么内容？

>>> "{{1}}".format("不打印", "打印")

答： '{1}'。因为{}里面还有一个{}，所以format位置无效

>>> "{{1}}".format("不打印", "打印")

'{1}'

2. 以下代码中，a, b, c是什么参数？

>>> "{a} love {b}.{c}".format(a="I", b="FishC", c="com")

'I love FishC.com'

答： a,b,c是关键字参数

3. 以下代码中，{0}, {1}, {2}是什么参数？

>>> "{0} love {1}.{2}".format("I", "FishC", "com")

'I love FishC.com'

答： 0,1,2是位置参数

4. 如果想要显示Pi = 3.14，format前边的字符串应该怎么填写呢？

''0:.2f”.format('Pi = ', 3.1415)

答：

>>> "Pi = {0:.2f}".format(3.1415)

'Pi = 3.14'

### 动动手

1. 请写下这一节课你学习到的内容：格式不限，回忆并复述是加强记忆的好方式！

答：

* format知识点（位置参数，关键字参数）
* 字符串格式化字符
* 格式化字符
* 转移字符

## 课时16 序列！序列！

列表、元组和字符串统称为序列

* 列表、元组和字符串的共同点

1. 都可以通过一个索引得到每一个元素
2. 默认索引值总是从0开始
3. 可以通过分片的方法得到一个范围内的元素的集合
4. 有很多共同的操作符（重复操作符名、拼接操作符和成员关系操作符等）

### 16.1 list

>>> a = list()

>>> a

[]

>>> b = "Amusi love Lucy!"

>>> b

'Amusi love Lucy!'

>>> b = list(b)

>>> b

['A', 'm', 'u', 's', 'i', ' ', 'l', 'o', 'v', 'e', ' ', 'L', 'u', 'c', 'y', '!']

>>> c = (1, 1, 2, 3, 5, 8, 13, 21, 34)

>>> c = list(c)

>>> c

[1, 1, 2, 3, 5, 8, 13, 21, 34]

### 16.2 max()

返回参数中的最大值

# 整型

>>> c

[1, 1, 2, 3, 5, 8, 13, 21, 34]

>>> len(c)

9

>>> max(c)

34

# 字符串

>>> b

['A', 'm', 'u', 's', 'i', ' ', 'l', 'o', 'v', 'e', ' ', 'L', 'u', 'c', 'y', '!']

>>> max(b)

'y'

>>> numbers = [1, 23, 100, 3, -1, -10, 11, -56, 71]

>>> max(numbers)

100

>>> tuple1 = (1, 23, 111, 17, -181, 19, 119, -122)

>>> max(tuple1)

119

### 16.3 min()

返回参数中的最小值

>>> numbers = [1, 23, 100, 3, -1, -10, 11, -56, 71]

>>> min(numbers)

-56

>>> tuple1 = (1, 23, 111, 17, -181, 19, 119, -122)

>>> min(tuple1)

-181

### 16.4 sum()

**语法**

sum(iterablep[, start=0])

**作用**

返回序列iterable和可选参数start的总和

**代码**实例

# tuple

>>> tuple1 = (3.1, 2.3, 6.8)

>>> sum(tuple1)

12.2

# list

>>> list1 = [1, 2, 4, 100, 99, 111]

>>> sum(list1)

317

### 16.5 sorted()

**语法**

sorted(\*)

**作用**

返回一个排序的列表/元组

**代码实例**

>>> numbers = [99, 1, 77, -11, 22, 11, -56, 1189, 0, 99]

>>> sorted(numbers)

[-56, -11, 0, 1, 11, 22, 77, 99, 99, 1189]

### 16.6 reversed()

**语法**

reversed(\*)

**作用**

将列表/元组逆序排列

**代码实例**

# 列表

>>> numbers

[99, 1, 77, -11, 22, 11, -56, 1189, 0, 99]

>>> list(reversed(numbers))

[99, 0, 1189, -56, 11, 22, -11, 77, 1, 99]

# 元组

>>> tuple1 = (1, 77, -11, 22, 11, -56, 1189, 0, 99)

>>> tuple(reversed(tuple1))

(99, 0, 1189, -56, 11, 22, -11, 77, 1)

### 16.7 enumerate()

**语法**

enumerate(iterable[, start])

**作用**

将列表/元组带索引

**代码实例**

>>> numbers

[99, 1, 77, -11, 22, 11, -56, 1189, 0, 99]

>>> list(enumerate(numbers))

[(0, 99), (1, 1), (2, 77), (3, -11), (4, 22), (5, 11), (6, -56), (7, 1189), (8, 0), (9, 99)]

### 16.8 zip()

**语法**

zip(iter1 [,iter2 [...]])

**作用**

将列表/元组

**代码实例**

>>> a = [1, 2, 3, 4, 5, 6, 7, 8]

>>> b = [4, 5, 6, 7, 8]

>>> zip(a, b)

<zip object at 0x7fb6e03c50c8>

>>> list(zip(a, b))

[(1, 4), (2, 5), (3, 6), (4, 7), (5, 8)]

### 测试题

1. 我们根据列表、元祖和字符串的共同特点，把它们三统称为什么？

答： 序列

1. 请问分别使用什么BIF，可以把一个可迭代对象转换为列表、元祖和字符串？

答： list()、tuple()、str()

1. 你还能复述出“迭代”的概念吗？

答：

1. 你认为调用 max("Lucy love Amusi") 会返回什么值？为什么？

答： 'y'。因为使用max()会自动将字符串内的元素转换成ASCII，再迭代进行比较，计算出其中ASCII的最大值。这里，’y’字符的ASCII值最大，为

>>> max("Lucy love Amusi")

'y'

注：使用ord()函数可以将char转换成int

使用chr()函数可以将int转换成char

>>> ord("y")

121

>>> ord("a")

97

>>> ord("A")

65

>>> chr(121)

'y'

>>> chr(65)

'A'

>>> chr(97)

'a'

### 动动手

0. 猜想一下 min() 这个BIF的实现过程

答: min(\*)

代码实现过程

def min(x):

    least = x[0]

    for each in x:

        if each < least:

            least = each

    return least

print(min('123456789'))

1. 视频中我们说 sum() 这个BIF有个缺陷，就是如果参数里有字符串类型的话就会报错，请写出一个新的实现过程，自动“无视”参数里的字符串并返回正确的计算结果

答：

def sum(x):

    result = 0

    for each in x:

        if (type(each) == int) or (type(each) == float):

            result += each

        else:

            continue

    return result

print(sum([1, 2.1, 2.3, 'a', '1', True]))

1. 请写下这一节课你学习到的内容：格式不限，回忆并复述是加强记忆的好方式！

答：

* 列表、元组和字符串统称为序列
* 序列的若干BIF:

list()、tuple()、min()、max()、min()、sum()、

reversed()、sorted()、enumerate()、zip()...

# 章节5 函数

## 课时17 函数: Python的乐高积木

关键字 def

### 17.1 无参函数

**无参函数语法**

* def \*():

函数执行代码

**代码实例**

>>> def MyFirstFunction():

print("This is the first function I create")

print("Unbelieve!")

print("Thank you!")

>>> MyFirstFunction()

This is the first function I create

Unbelieve!

Thank you!

### 17.2 有参函数

**有参函数语法**

* def \*(\*, \*, ... ,\*):

函数执行代码

**代码实例**

# 一个参数

>>> def MySecondFunction(name):

print(name + ", I love you!")

>>> MySecondFunction("Lucy")

Lucy, I love you!

# 两个参数

>>> def sumNums(num1, num2):

result = num1 + num2

print(result)

>>> sumNums(99, 2)

101

### 17.3 函数的返回值

**关键字**： return

**语法**：在函数执行代码最后使用return返回数据

**代码实例**

>>> def sumNums(num1, num2):

return (num1 + num2)

>>> print(sumNums(99, 2))

101

### 测试题

0. 你有听说过DRY吗？

答： 不重复代码。DRY是程序员们公认的指导原则：Don't Repeat Yourself. 不要再去重复拷贝一段代码了！

1. 都是重复一段代码，为什么我要使用函数（而不使用简单的拷贝黏贴）呢？

答： 函数易移植、可读性高，当有参数需要传递的时候，效果更明显

2. 函数可以有多个参数吗？

答： 可以

1. 创建函数使用什么关键字，要注意什么？

答： 关键字def。注意，函数名后面需要加上小括号()，括号内可以有参数

4. 请问这个函数有多少个参数？

def MyFun((x, y), (a, b)):

    return x \* y - a \* b

答： 无参数。因为类似于这样的写法是错误的！

我们分析下，函数的参数需要的是变量，而这里你试图用“元祖”的形式来传递是不可行的。

5. 请问调用以下这个函数会打印什么内容？

>>> def hello():

         print('Hello World!')

         return

        print('Welcome To FishC.com!')

答：打印’Hello World’。因为程序执行到return就会退出函数体，不再继续打印

### 动动手

1. 编写一个函数power()模拟内建函数pow()，即power(x, y)为计算并返回x的y次幂的值。

答：

>>> def power(x, y):

powerValue = 1

for i in range(y):

powerValue \*= x

return powerValue

>>> power(3, 2)

9

>>> power(101, 2)

10201

1. 编写一个函数，利用欧几里得算法（[脑补链接](http://baike.baidu.com/view/255668.htm" \t "_blank)）求最大公约数，例如gcd(x, y)返回值为参数x和参数y的最大公约数。

答：

>>> def gcd(x, y):

while y:

t = x % y

x = y

y = t

return x

>>> print(gcd(4, 6))

2

>>> print(gcd(99,33))

33

1. 编写一个将十进制转换为二进制的函数，要求采用“除2取余”（[脑补链接](http://baike.baidu.com/view/1426817.htm" \t "_blank)）的方式，结果与调用bin()一样返回字符串形式。

答：

**写法1**

>>> def binstr(num):

output = []

while(num != 0):

b = num % 2

num = num // 2

output.append(str(b))

output.reverse()

result = "".join(output)

return result

>>> binstr(100)

'1100100'

>>> binstr(4)

'100'

**写法2**

def binstr(num):

output = []

result = ""

while(num != 0):

b = num % 2

num = num // 2

output.append(b)

while output:

result += str(output.pop())

return result

1. 请写下这一节课你学习到的内容：格式不限，回忆并复述是加强记忆的好方式！

答： 函数的知识点

* 定义函数的关键字def
* 无参函数
* 有参函数
* 返回值return

## 课时18 函数: 灵活即强大

### 18.1 形参和实参

* 形参：函数定义过程中的name
* 实参：传递进来的值

>>> def output(name):

print(name)

>>> output(123)

123

上述代码中，name为形参，123为实参

### 18.2 函数文档

**语法**

name.\_\_doc\_\_

**代码实例**

>>> def output(name):

"打印输入数据"

print(name)

>>> output.\_\_doc\_\_

'打印输入数据'

>>> def output(name):

"打印输入数据"

"如输入a，则打印a"

print(name)

>>> output.\_\_doc\_\_

'打印输入数据'

注：函数有多行注释，但函数文档只默认输出第一行注释内容

### 18.3 关键字参数

在调用函数时，可以在实参前面写上对应的形参name（这里name即关键字），这样避免了参数的位置问题

**代码实例**

>>> SaySome("阿木寺的心愿", "世界和平")

阿木寺的心愿->世界和平

>>> SaySome("世界和平", "阿木寺的心愿")

世界和平->阿木寺的心愿

>>> SaySome(words = "世界和平", name = "阿木寺的心愿")

阿木寺的心愿->世界和平

### 18.4 默认参数

函数声明/定义时，给形参赋初值。这样，当调用函数没有输入实参，函数会输出默认参数的运算结果。

**代码实例**

>>> def SaySome(name = "阿木寺的心愿", words = "世界和平"):

print(name + "->" + words)

# 使用默认参数

>>> SaySome()

阿木寺的心愿->世界和平

# 使用实参

>>> SaySome("学习","人生价值")

学习->人生价值

### 18.5 收集/可变参数

在形参前加上”\*”。当输入参数数量不确定时，可以使用可变参数

**代码实例**

>>> def test(\*params):

print("参数的长度: ", len(params))

print("第二个参数: ", params[1])

>>> test(99, "Amusi", 1891, 1.9, "Lucy")

参数的长度: 5

第二个参数: Amusi

>>> def test(\*params, exp):

print("参数的长度: ", len(params), exp)

print("第二个参数: ", params[1])

# 特殊情况

>>> test(99, "Amusi", 1891, 1.9, "Lucy")

Traceback (most recent call last):

File "<pyshell#152>", line 1, in <module>

test(99, "Amusi", 1891, 1.9, "Lucy")

TypeError: test() missing 1 required keyword-only argument: 'exp'

# 特殊情况解决： 使用关键字参数

>>> test(99, "Amusi", 1891, 1.9, "Lucy", exp = 100)

参数的长度: 5 100

第二个参数: Amusi

注： print()函数其实就使用了收集/可变参数

### 测试题

1. 请问下面哪个是形参，哪个是实参？

答： x为形参，y为实参

>>> def MyFun(x):

return x \*\* 3

>>> y = 3

>>> print(MyFun(y))

1. 函数文档和直接用”#”为函数写注释有什么不同？

答：函数文档方便程序员查看，直接输入\*.\_\_doc\_\_即可函数文档

1. 使用关键字参数，可以有效避免什么问题出现？

答：避免多参数的顺序问题，使用关键字，可以指定实参和形参的对应

1. 使用help(print)查看print()这个BIF有哪些默认参数？分别起什么作用？

答： 默认参数有sep，end，file，flush

>>> help(print)

Help on built-in function print in module builtins:

print(...)

print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

Prints the values to a stream, or to sys.stdout by default.

Optional keyword arguments:

file: a file-like object (stream); defaults to the current sys.stdout.

sep: string inserted between values, default a space.

end: string appended after the last value, default a newline.

flush: whether to forcibly flush the stream.

1. 默认参数和关键字参数最大的区别是什么？

答

* 默认参数：在函数声明/定义时指定参数的默认值
* 关键字参数：在调用函数时，指定与实参对应的关键字

### 动动手

1. 编写一个符合以下要求的函数：

a)计算打印所有参数的和乘以基数(base = 3)的结果

b)如果最后一个参数base =5，则改变基数

答： 先使用收集/可变参数，即在形参前加上”\*”，再使用默认参数

>>> def func(\*params, base = 3):

sum = 0

for each in params:

sum += each

return sum \* base

>>> func(1,2,3)

18

>>> func(1, 2 , base = 5)

15

1. 编写一个函数findstr()，该函数统计一个长度为2的子字符串在另一个字符串中出现的次数。例如输入的字符串为”AABBCC AABBCC CCBBAA CCBBAA”，子字符串为AB，函数输出结果是“子字符串在目标字符串中出现2次”

答：

>>> def findstr(longstr, substr):

length = len(longstr)

counts = 0

for i in range(length-1):

if(longstr[i] == substr[0]) and (longstr[i+1] == substr[1]):

counts += 1

return ("子字符串在目标字符串中出现%d次" % counts)

>>> findstr("AABBCC AABBCC CCBBAA CCBBAA", "AB")

'子字符串在目标字符串中出现2次'

>>> findstr("AABBCC AABBCC CCBBAA CCBBAA", "AA")

'子字符串在目标字符串中出现4次'

## 课时19 函数: 我的地盘听我的

* 函数有返回值
* 过程没有返回值

注： Python只有函数，没有过程

**代码实例**

>>> def Hello():

print("Hello, Amusi!")

>>> Hello()

Hello, Amusi!

>>> temp = Hello()

Hello, Amusi!

>>> temp

>>> print(temp)

None

>>> type(temp)

<class 'NoneType'>

### 19.1 函数返回值

动态确定变量/函数返回值类型

* 函数返回多个值

# 返回列表

>>> def back():

return [1, "Amusi", 9.99]

>>> back()

[1, 'Amusi', 9.99]

# 返回元组

>>> def back():

return 1, "Amusi", 9.99

>>> back()

(1, 'Amusi', 9.99)

### 19.2 函数变量的作用域

* 局部变量： 作用域仅限于函数体内
* 全局变量： 作用域在整个程序中

注：可以试图在函数体内读取全局变量，但不要试图在函数体内修改全局变量

**代码实例**

def discount(price, rate):

final\_price = price \* rate

return final\_price

old\_price = float(input("请输入原价： "))

rate = float(input("请输入折扣率: "))

new\_price = discount(old\_price, rate)

print("打折后价格为: ", new\_price)

输出

打折后价格为: 80.0

上述代码中，final\_price是局部变量，所以在函数外打印该变量会出错。

>>> print("试图打印局部变量final\_price: ", final\_price)

Traceback (most recent call last):

File "<pyshell#60>", line 1, in <module>

print("试图打印局部变量final\_price: ", final\_price)

NameError: name 'final\_price' is not defined

而old\_price，rate，new\_price是全局变量（函数体内虽然可以访问该变量，但Python会重新创建新的同名局部变量，其值互不影响）

### 测试题

1. 下面程序会输出什么？

答：

>>> def next():

print("我在next()函数里...")

pre()

>>> def pre():

print("我在pre()函数里...")

# 输出

>>> next()

我在next()函数里...

我在pre()函数里...

1. 请问以下函数有返回值吗？

答： 有。Python的任何函数都有返回值，对于显式手动添加返回值的函数，Python会返回一个None对象。

>>> def hello():

print("Hello, Amusi!")

>>> result = hello()

Hello, Amusi!

>>> print(result)

None

1. 请问Python的return语句可以返回多个不同类型的值吗？

答： 可以。return列表或者元组即可实现

# 返回列表

>>> def back():

return [1, "Amusi", 9.99]

>>> back()

[1, 'Amusi', 9.99]

# 返回元组

>>> def back():

return 1, "Amusi", 9.99

>>> back()

(1, 'Amusi', 9.99)

1. 以下程序会打印什么内容？

答：1314520

def fun(var):

var = 1314

print(var, end = "")

var = 520

fun(var)

print(var)

1. 以下程序会打印什么内容？

答： Baby I love you

var = " Hi "

def fun1():

global var

var = " Baby "

return fun2(var)

def fun2(var):

var += "I love you"

fun3(var)

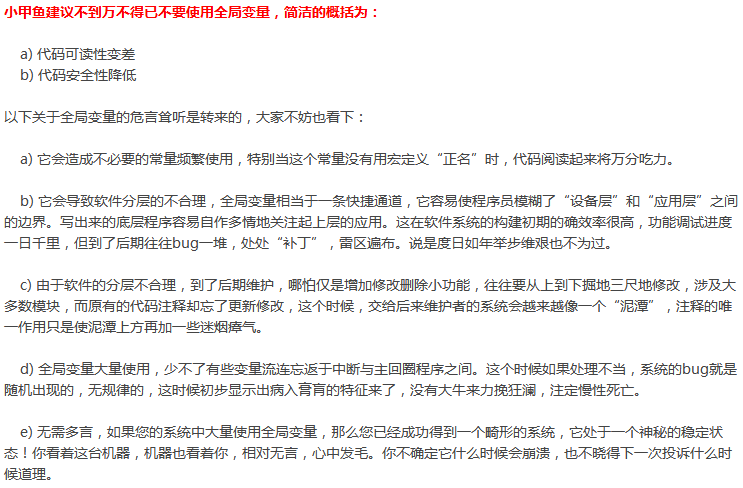
return var

def fun3(var):

var = " Amusi "

print(fun1())

注： 这里再次强调全局变量和局部变量的使用，若要在函数体内访问并修改全局变量，可在函数体再次定义同名变量，但前面需要加上关键字global。但不建议在函数体内对全局变量修改，具体原因可见下述分析：



### 动动手

0. 编写一个函数，判断传入的字符串参数是否为“回文联”（回文联即用回文形式写的对联，即可顺读，也可倒读，例如：上海自来水来自海上）

答：

def palindrome(string):  
 length = len(string)  
 string2 = string  
 flag = 1  
 for i in range(length // 2):  
 if string[i] != string2[length -(i+1)]:  
 flag = 0  
 break  
 if flag == 1:  
 print("输入的字符串是回文联！")  
 else:  
 print("输入的字符串不是回文联！")  
palindrome("上海自来水来自海上")  
palindrome("我爱你因为你爱我")

## 课时20 函数: 内嵌函数和闭包

### 20.1 global关键字

使用global关键字，在函数体内先重新声明与全局变量同名的变量，即可在函数体内修改全局变量值

**代码实例**

# 不使用global关键字

>>> count = 5

>>> def MyFun():

count = 10

print(count)

>>> MyFun()

10

>>> print(count)

5

# 使用global关键字

>>> def MyFun():

global count

count = 10

print(count)

>>> MyFun()

10

>>> print(count)

10

### 20.2 内嵌函数

内嵌函数是在函数体A内部再定义函数体...，内部定义函数体仅可以在函数体A中被调用，而不能被函数体A外部的函数调用。

**代码实例**

>>> def fun1():

print("fun1()正在被调用...")

def fun2():

print("fun2()正在被调用...")

fun2()

# 调用fun1()

>>> fun1()

fun1()正在被调用...

fun2()正在被调用...

### 20.3 闭包closure

闭包(closure)是函数式编程的重要的语法结构.

闭包的概念：闭包（Closure）是词法闭包（Lexical Closure）的简称，是引用了自由变量的函数。这个被引用的自由变量将和这个函数一同存在，即使已经离开了创造它的环境也不例外。所以，闭包是由函数和与其相关的引用环境组合而成的实体。

**代码实例1**

>>> def funX(x):

def funY(y):

return x \* y

return funY

>>> i = funX(8)

>>> i

<function funX.<locals>.funY at 0x7efdd3c02048>

>>> type(i)

<class 'function'> # 可知i是一个函数，所以可以继续调用

>>> i(5)

40

>>> funX(8)(5)

40

**分析**：在一个内部函数Y里，对外部作用域X的变量x进行操作

**代码实例2**

# 无效闭包

>>> def fun1():

x = 5

def fun2():

x \*= x

return x

return fun2()

>>> fun1()

Traceback (most recent call last):

File "<pyshell#42>", line 1, in <module>

fun1()

File "<pyshell#41>", line 6, in fun1

return fun2()

File "<pyshell#41>", line 4, in fun2

x \*= x

UnboundLocalError: local variable 'x' referenced before assignment

注：这里，对于fun2()中调用x，就类似于在函数体调用全局变量（其实x是fun2()函数非全局变量的外部变量），所以不能成功。

# 代码改造——使用列表

>>> def fun1():

x = [5]

def fun2():

x[0] \*= x[0]

return x[0]

return fun2()

>>> fun1()

25

# 代码改造——使用nonlocal

>>> def fun1():

x = 5

def fun2():

nonlocal x

x \*= x

return x

return fun2()

>>> fun1()

25

### 测试题

1. 如果希望在函数中修改全局变量的值，应该使用什么关键字？

答： global

1. 在嵌套的函数中，如果希望内部函数修改外部函数的局部变量，应该使用什么关键字？

答： nonlocal

1. Python的函数可以嵌套，但要注意访问的作用域问题，请问以下代码存在什么问题？

答： 函数体外不能访问函数体内的嵌套函数，因为嵌套函数作用域是局部的。

>>> def outside():

print("I am outside!")

def inside():

print("I am inside!")

>>> inside()

# output

Traceback (most recent call last):

File "<pyshell#64>", line 1, in <module>

inside()

NameError: name 'inside' is not defined

1. 请问为什么代码A没有报错，但代码B却报错了？应该如何修改？

答： 代码A没有报错，因为var = 3，这里的var是inside()的局部变量

代码B报错，因为print(var)，调用了outsize()函数中非全局变量的外部变量var，类似于函数体内部修改全局变量一样，会出错。

**代码A**

>>> def outside():

var = 5

def inside():

var = 3

print(var)

inside()

# 正常输出

>>> outside()

3

**代码B**

>>> def outside():

var = 5

def inside():

print(var)

var = 3

inside()

# 输出错误

>>> outsize()

Traceback (most recent call last):

File "<pyshell#75>", line 1, in <module>

outsize()

NameError: name 'outsize' is not defined

1. 请问如何访问funIn()？

答： 不可以直接访问到funIn()，但可以通过调用funOut()来间接访问funIn()

def funOut():

def funIn():

print("Congratulations!")

return funIn()

>>> i = funOut()

Congratulations!

1. 请问如何访问funIn()？

答： 闭包。即有一种函数的函数关系，所以需要调用两次函数的形式

>>> def funOut():

def funIn():

print("Congratulations!")

return funIn

# 访问方法一

>>> funOut()()

Congratulations!

# 访问方法二

>>> i = funOut()

>>> type(i)

<class 'function'>

>>> i()

Congratulations!

1. 以下是一段“闭包”的代码，请预测输出结果

答： 6, 7, 8。因为此时a是可以直接访问funY()函数，每次调用时，不会先进入funX()，在进入funY()。所以每次x的值都是上一次运算后的值，并不是5。

>>> def funX():

x = 5

def funY():

nonlocal x

x += 1

return x

return funY

>>> a = funX()

>>> print(a())

6

>>> print(a())

7

>>> print(a())

8

### 动动手

* global
* nonlocal
* 内嵌和闭包closure

## 课时21 函数：lambda表达式

用lambda关键字构建匿名函数

### 21.1 Lambda表达式的作用

* Python写一些执行脚本时，使用lambda就可以省下定义函数过程，比如说我们只需要写个简单的脚步来管理服务器时间，我们就不需要专门定义一个函数然后再调用，使用lambda就可以使得代码更加精简
* 对于一些比较抽象并且整个程序执行下来，只需要调用两次函数，有时候给函数起个名字也是比较头疼的问题，使用lambda就不需要考虑命名的问题了
* 简化代码的可读性，由于普通的函数阅读需要跳到开头的def定义部分，使用lambda函数就可以省去这样的步骤

**代码实例**

# 一般函数

>>> def ds(x):

return 2 \* x + 1

>>> ds(5)

11

# 匿名函数

>>> lambda x: 2 \* x + 1

<function <lambda> at 0x7f0c5eb7fe18>

>>> g = lambda x: 2 \* x + 1

>>> g(5)

11

# 一般函数

>>> def add(x, y):

return x + y

>>> add(6, 8)

14

# 匿名函数

>>> lambda x,y : x + y

<function <lambda> at 0x7f0c5d2dbc80>

>>> g = lambda x,y: x + y

>>> g(6, 8)

14

>>> g = lambda a,b,c: a + b + c

>>> g(6,8,10)

24

### 21.2 filter()

filter是一个BIF，把非True的元素过滤掉

**filter语法**

filter(function or None, iterable) --> filter object

参数1：函数名或者None

参数2：可迭代数据，如列表、元组等

**查看filter帮助文档**

>>> help(filter)

Help on class filter in module builtins:

class filter(object)

| filter(function or None, iterable) --> filter object

|

| Return an iterator yielding those items of iterable for which function(item)

| is true. If function is None, return the items that are true.

|

| Methods defined here:

|

| \_\_getattribute\_\_(self, name, /)

| Return getattr(self, name).

|

| \_\_iter\_\_(self, /)

| Implement iter(self).

|

| \_\_new\_\_(\*args, \*\*kwargs) from builtins.type

| Create and return a new object. See help(type) for accurate signature.

|

| \_\_next\_\_(self, /)

| Implement next(self).

|

| \_\_reduce\_\_(...)

| Return state information for pickling.

**代码实例**

# code1

>>> list(filter(None, [1, 0, False, True, 2, 4]))

[1, True, 2, 4]

# code2

>>> def odd(x):

return x % 2

>>> temp = range(10)

>>> show = filter(odd, temp)

>>> list(show)

[1, 3, 5, 7, 9]

>>> list(filter(lambda x: x % 2, range(10)))

[1, 3, 5, 7, 9]

### 21.3 map()

将序列的每个元素的索引值作为函数的参数进行计算，直到到索引最后。

**map()帮助文档**

>>> help(map)

Help on class map in module builtins:

class map(object)

| map(func, \*iterables) --> map object

|

| Make an iterator that computes the function using arguments from

| each of the iterables. Stops when the shortest iterable is exhausted.

|

| Methods defined here:

|

| \_\_getattribute\_\_(self, name, /)

| Return getattr(self, name).

|

| \_\_iter\_\_(self, /)

| Implement iter(self).

|

| \_\_new\_\_(\*args, \*\*kwargs) from builtins.type

| Create and return a new object. See help(type) for accurate signature.

|

| \_\_next\_\_(self, /)

| Implement next(self).

|

| \_\_reduce\_\_(...)

| Return state information for pickling.

**代码实例**

>>> list(map(lambda x: x \* 2, range(10)))

[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]

### 练习题

1. 请使用lambda表达式将下面的函数变成匿名函数

答：

# 原代码

>>> def fun\_A(x, y = 3):

return x \* y

# 输出

>>> fun\_A(6)

18

>>> fun\_A(6, 6)

36

# lambda代码

>>> g = lambda x, y=3: x \* y

# 输出

>>> g(6)

18

>>> g(6, 6)

36

1. 请将下边的匿名函数改写成普通函数

答：

# 原匿名函数

>>> g = lambda x: x if x % 2 else None

>>> g(10)

>>> g(11)

11

# 普通函数

>>> def fun1(x):

if x % 2:

return x

>>> fun1(10)

>>> fun1(11)

11

2. 利用filter()和lambda表达式快速求出100以内所有3的倍数吗?

答：

>>> list(filter(lambda x: x % 3 == 0, range(100)))

[0, 3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 39, 42, 45, 48, 51, 54, 57, 60, 63, 66, 69, 72, 75, 78, 81, 84, 87, 90, 93, 96, 99]

>>> list(filter(lambda x: x if x % 3 == 0 else None, range(100)))

[3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 39, 42, 45, 48, 51, 54, 57, 60, 63, 66, 69, 72, 75, 78, 81, 84, 87, 90, 93, 96, 99]

1. 使用zip可以将两个数以元组的形式绑定在一起，例如：

>>> list(zip([1, 3, 5, 7, 9], [2, 4, 6, 8, 10]))

[(1, 2), (3, 4), (5, 6), (7, 8), (9, 10)]

但如果想使用map和lambda表达式不仅可以以元组的形式绑定，甚至是列表的形式，请问如何写？

答：

>>> list(map(lambda x, y: [x, y], [1, 3, 5, 7, 9], [2, 4, 6, 8, 10]))

[[1, 2], [3, 4], [5, 6], [7, 8], [9, 10]]

1. 预测下述表达式会打印什么？

答：

>>> def make\_repeat(n):

return lambda s: s \* n

>>> double = make\_repeat(2)

>>> print(double(8))

16

>>> print(double("Amusi"))

AmusiAmusi

# 等价于

>>> double = lambda s: s \* 2

>>> print(double(8))

16

>>> print(double("Amusi"))

AmusiAmusi

## 课时22 函数：递归是神马

递归：有进有出

**实例**

* 写一个求阶乘的函数

正整数阶乘指从1乘以2，乘以3乘以4一直乘到所要乘的数。

如所给的数为5,则阶乘是1x2x3x4x5，得到的积是120，所以120就是5的阶乘

**代码**

# 非递归版本

>>> def factorial(n):

result = n

for i in range(1 ,n):

result \*= i

return result

>>> factorial(5)

120

# 递归版本

>>> def recur(x):

if(x == 1):

return 1

else:

return (x) \* recur(x-1)

>>> recur(5)

120

**测试题**

1. 递归在编程上的形式是如何表现的？

答： 调用函数自身，停止处理程序（一般，参数不断减小）

1. 递归必须满足哪两个基本条件？

答：

* 函数内部调用自身
* 正确的停止条件

1. 按照递归的特性，在编程中有没有不得不使用递归的情况？

答： 快速排序、树的定义

1. 聊聊递归的优缺点

* 递归有危险性，如果停止条件没有设置好，程序会陷入死循环，以至于程序崩溃
* 递归是函数调用自身，首先调用函数需要压栈、弹栈、保存和恢复寄存器等操作，所以当递归次数较大时，会浪费较多内存资源。

**动动手**

1. 使用递归编写一个power()函数模拟内建函数pow()，即power(x,y)为计算并返回x的y次幂的值

答：

>>> def power(m, n):

if(n == 0):

return 1

return m \* power(m, n-1)

>>> power(2, 2)

4

>>> power(2, 10)

1024

>>> power(3, 3)

27

1. 使用递归编写一个函数，利用欧几里得求最大公约数，例如gcd(x,y)返回值为参数x和参数y的最大公约数。

答：

>>> def gcd(x, y):

t = x % y

if(t == 0):

return y

else:

return gcd(y, t)

>>> gcd(4, 6)

2

>>> gcd(33, 99)

33

## 课时23 递归：这帮兔崽子

### 23.1 斐波那契数列

以兔子的繁殖能力来举例，兔子的繁殖能力是惊人的，如下图：

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 月数 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| 兔子对数 | 1 | 1 | 2 | 3 | 5 | 8 | 13 | 21 | 34 | 55 | 89 | 144 |

用函数来表示：

F(n) = 1, 当n=1

F(n) = 1, 当n=2

F(n) = F(n-1) + F(n-2)，当n>2

课间练习

假设我们需要求出经历了20个月后，总共有多少对兔子？（迭代 VS 递归）

答：

# 迭代法

>>> def fibonacci(n):

past = 1

now = 1

result = 2

if( n == 1 or n == 2):

return 1

if( n > 2):

for i in range(1, n-1):

result = past + now

past = now

now = result

return result

else:

return -1

>>> fibonacci(3)

2

>>> fibonacci(7)

13

>>> fibonacci(12)

144

# 递归法

>>> def fibonacci\_rec(n):

if(n == 1 or n == 2):

return 1

if(n > 2):

return fibonacci\_rec(n-1) + fibonacci\_rec(n-2)

else:

return -1

>>> fibonacci\_rec(3)

2

>>> fibonacci\_rec(7)

13

>>> fibonacci\_rec(12)

144

注：当n值很大时，递归法的速度比迭代法慢很多（所以递归并不适用大多数情况）。

## 课时24 递归：汉诺塔

[汉诺塔](https://baike.baidu.com/item/%E6%B1%89%E8%AF%BA%E5%A1%94/3468295?fr=aladdin)：汉诺塔（又称河内塔）问题是源于印度一个古老传说的益智玩具。大梵天创造世界的时候做了三根[金刚石](https://baike.baidu.com/item/%E9%87%91%E5%88%9A%E7%9F%B3/80698" \t "/tmp/wps-amusi/x/_blank)柱子，在一根柱子上从下往上按照大小顺序摞着64片黄金圆盘。大梵天命令婆罗门把圆盘从下面开始按大小顺序重新摆放在另一根柱子上。并且规定，在小圆盘上不能放大圆盘，在三根柱子之间一次只能移动一个圆盘。

[解题思想](https://www.zhihu.com/question/24385418)

我们的目标是x上所有n个盘子移动到z上。至于y盘，我们可以将之看成中转站。

规则要求，小盘子必须在大盘子上。这个过程，必有“移动最大盘子”步骤：n-1个盘子在y中转站上，而我们整将最大盘子从x移动到z上。只有经历“移动最大盘子”步骤，余下的事情才有可能实现。而在此之前，我们要做的就是实现这个步骤。

现在，游戏整个过程以“移动最大盘子”为中央，被分成了兩个步骤。即（前）“将n-1个盘子从x移动到y上”，（中）“移动最大盘子”，（后）“将n-1个盘子从y移动到z上”

这是我们意识到，（前）与（后）操作道理是相似的。不去管那个最大盘子，（前）是以C针为中转站，（后）是以A针为中转站。因此两者所需的移动次数应当是相等的。这意味着我们只要计算出其中一者的移动次数，然而乘以2，在加上“移动最大盘子”的那1次，就是这场游戏的总移动次数了。

用数学语言表达，假设（前）“将N-1个盘子从A针移动到B针”所需次数为Hn-1，总移动次数为Hn，那么可以得出的关系就是：Hn=Hn-1 x 2 + 1.

其实当我们得出这个算式的时候，稍微聪明一点的人已经明白，这就是一个递推公式，可以直接用此公式得出Hn的通解。

**代码**

def hanoi(n, x, y, z):  
 if n == 1:  
 print(x, " --> ", z)  
 else:  
 hanoi(n-1, x, z, y) # 将前n-1个盘从移动到y上  
 print(x, " -->", z) # 将最底下的最后一个盘子从x移动到z上  
 hanoi(n-1, y, x, z) # 将y上的n-1盘子移动到z上  
n = int(input("请输入hanoi层数: "))  
hanoi(n, "x", "y", "z")

注：代码很简洁，主要思想是把参数2作为中转站，参数1是源，参数3是目标。将源移动到目标是我们需要做的事情。每次递归，都需要执行(前)(后)。

**动动手**

1. 使用递归编写一个十进制转换为二进制的函数（要求采用”取2取余”的方式，结果与调用bin()一样返回字符串形式）

答：

# 迭代法

>>> def d2b(x):

if x == 0:

return 0

result = []

strResult = ""

while x != 0:

a = x % 2

if a == 0:

result.append(0)

else:

result.append(1)

x = x // 2

result.reverse()

for i in result:

strResult += str(i)

return strResult

>>> d2b(10)

'1010'

>>> d2b(512)

'1000000000'

>>> d2b(8)

'1000'

>>> d2b(1024)

'10000000000'

# 递归法

def d2b(x):

result = ""

if x:

result = d2b(x//2)

return result + str(x%2)

else:

return result

>>> d2b(10)

'1010'

>>> d2b(512)

'1000000000'

>>> d2b(8)

'1000'

>>> d2b(1024)

'10000000000'

>>> d2b(24)

'11000'

1. 写一个函数get\_digits(n)，将参数n分解出每个位的数字并按顺序存放到列表中。举例：get\_digits(12345)-->[1,2,3,4,5]

答：

>>> result = []

>>> def get\_digits(n):

if n > 0:

result.insert(0, n%10) # insert(0, \*)是给列表的首地址指向的元素赋值

get\_digits(n//10)

>>> get\_digits(123456)

>>> print(result)

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

1. 使用递归的方式求解“回文字符串”

答：

# 使用递归的方式求解“回文字符串”

def is\_palindrome(n, start, end):

if start > end:

return 1

else:

#return is\_palindrome(n, start+1, end-1) if n[start] == n[end] else 0

if n[start] == n[end]:

return is\_palindrome(n, start+1, end-1)

else:

return 0

string = input("请输入一串字符串: ")

length = len(string)-1

if is\_palindrome(string, 0, length):

print("%s 是回文字符串！" % string)

else:

print("%s 不是回文字符串！" % string)

1. 使用递归编程求解如下问题：

有5个人坐在一起，问第5个人多少岁？他说比第4个大2岁。问第4个人岁数，他说比第3个人大2岁。问第3个人，又说比第2个人大2岁。问第2个人，说比第一个人大2岁。最后问第一个人，他说是10岁。请问第5个人多少岁？

答：

def age(n):

if n == 1:

return 10

else:

return 2 + age(n-1)

print("第5个人 %d 岁" % age(5))

# 章节6 字典与集合

## 课时25 字典: 当索引不好用时

* 字典是Python中唯一的**映射**类型
* 字典由键(key)和值(value)组成，即dict = {key1 : value1, key2 : value2, ...}

**代码实例**

# 一般方法

>>> brand = ["耐克", "阿迪达斯", "Vans", "匡威", "Amusi"]

>>> slogan = ["Just do it", "Impossible is noting", "Don't give up", "Fighthing!", "Stay hungray"]

>>> print("Amusi的口号是: ", slogan[brand.index("Amusi")])

Amusi的口号是: Stay hungray

# 字典

>>> dict1 = {"耐克":"Just do it", "阿迪达斯":"Impossible is noting", "Vans":"Don't up", "匡威":"Fighthing!", "Amusi":"Stay hungray"}

>>> print("Amusi的口号是: ", dict1["Amusi"])

Amusi的口号是: Stay hungray

>>> dict2 = {1:"one", 2:"two", 3:"three"}

>>> print(dict2[1])

one

>>> dict3 = {}

>>> dict3

{}

>>> type(dict3) # 查看类型

<class 'dict'>

# 另一种生成字典方法

>>> dict3 = dict((("F", 70), ("A", 65), ("a", 92)))

>>> dict3

{'F': 70, 'A': 65, 'a': 92}

# 另一种生成字典方法

>>> dict4 = dict(Amusi = "阿木寺", Lucy = "静")

>>> dict4

{'Amusi': '阿木寺', 'Lucy': '静'}

# 修改键对应的值

>>> dict4["Amusi"] = "阿木寺better man"

>>> dict4

{'Amusi': '阿木寺better man', 'Lucy': '静'}

# 添加键值

>>> dict4["JackieChen"] = "成龙"

>>> dict4

{'Amusi': '阿木寺better man', 'Lucy': '静', 'JackieChen': '成龙'}

>>> dict4["Apple"] = "苹果"

>>> dict4

{'Amusi': '阿木寺better man', 'Lucy': '静', 'JackieChen': '成龙', 'Apple': '苹果'}

### 测试题

1. 当你听到小伙伴们在讨论”映射”、“哈希”、”散列”或者”关系数组”的时候，事实上他们是在讨论什么？

答：字典

1. 尝试一下将数据(“F”:70, “C”:67, “h”:104, “i”:105, “s”:115 )创建为一个字典并访问键”C”对应的值

答：

>>> dict1 = {"F":70, "C":67, "h":104, "i":105, "s":115}

>>> print(dict1["C"])

67

1. 用方括号(“[]”)括起来的数据我们称为列表(dictionary)，那么用大括号(“{}”)括起来的数据我们叫字典，对么？

答：不对。严格来说，字典是由键(key)和值(value)两部分组成，一定存在映射关系

1. 你如何理解有些东西字典做得到，但”万能”的列表却难以实现？

答： 当索引值不是整型时，列表无法完成映射，而字典却可以做到

1. 下边这些代码，他们都在执行一样的操作吗？你看得出差别吗？

>>> a = dict(one = 1, two = 2, three = 3)

>>> b = {"one":1, "two":2, "three":3}

>>> c = dict(zip(["one", "two", "three"], [1, 2, 3]))

>>> d = dict({"three":3, "one":1, "two":2 })

答： 这些代码是执行一样的操作，都是生成字典。只不过生成字典的方式不一样，有使用”=”，有使用”:”，有使用元组结构的，还有使用字典结构的。这些方法都能生成字典

# 输出结果

>>> a

{'one': 1, 'two': 2, 'three': 3}

>>> b

{'one': 1, 'two': 2, 'three': 3}

>>> c

{'one': 1, 'two': 2, 'three': 3}

>>> d

{'three': 3, 'one': 1, 'two': 2}

1. 如图，你可以推测出打了\*\*\*部分的代码吗？

>>> data = "23, Amusi, Male"

>>> MyDict = {}

>>> (MyDict["Age"], MyDict["Name"], MyDict["Gender"]) = \*\*\*

>>> print("Age:\t" + MyDict["Age"])

Age: 23

>>> print("Name:\t" + MyDict["Name"])

Name: Amusi

>>> print("Gender:\t" + MyDict["Gender"])

Gender: Male

答： \*\*\*应该为data.split(“,”)

### 动动手

1. 尝试利用字典的特性编写一个通讯录程序，功能如图：

答：

def PrintInfo():  
 print("""|--- 欢迎进入通讯录程序 ---|  
 |--- 1:查询联系人资料 ---|  
 |--- 2:插入新的联系人 ---|  
 |--- 3:删除已有联系人 ---|  
 |--- 4:退出通讯录程序 ---|  
 """)  
  
def Process(n):  
 contacts = dict()  
 while 1:  
 if n == 1:  
 nameTemp = input("请输入联系人姓名: ")  
 if nameTemp in contacts: # 注意写法  
 print(nameTemp + ": " + contacts[nameTemp])  
 else:  
 print("您输入的姓名不在通讯录中!")  
 if n == 2:  
 nameTemp = input("请输入联系人姓名: ")  
 if nameTemp in contacts:  
 print("您输入的姓名在通讯录中已经存在-->>", end ="") # 不换行  
 print(nameTemp + ": " + contacts[nameTemp])  
 else:  
 contacts[nameTemp] = input("请输入用户联系电话: ")  
 if n == 3:  
 nameTemp = input("请输入联系人姓名: ")  
 if nameTemp in contacts:  
 del(contacts[nameTemp]) # 注意写法，也可以使用dict.pop()  
 else:  
 print("您输入的联系人不存在...")  
 if n == 4:  
 print("|--- 感谢使用通讯录程序 ---|")  
 break;  
 n = int(input("\n请输入相关的指令代码: "))  
  
  
PrintInfo()  
num = int(input("请输入相关的指令代码: "))  
Process(num)

|--- 欢迎进入通讯录程序 ---|

|--- 1:查询联系人资料 ---|

|--- 2:插入新的联系人 ---|

|--- 3:删除已有联系人 ---|

|--- 4:退出通讯录程序 ---|

请输入相关的指令代码: 2

请输入联系人姓名: Amusi

请输入用户联系电话: 18817277750

请输入相关的指令代码: 2

请输入联系人姓名: Lucy

请输入用户联系电话: 1881777819

请输入相关的指令代码: 1

请输入联系人姓名: Lucy

Lucy: 1881777819

请输入相关的指令代码: 3

请输入联系人姓名: Lucy

请输入相关的指令代码: 1

请输入联系人姓名: Lucy

您输入的姓名不在通讯录中!

请输入相关的指令代码: 4

|--- 感谢使用通讯录程序 ---|

## 课时26 字典: 当索引不好用时2

字典的内建函数

>>> dir(dict) # 查看字典(Dictionary)所有的内建函数

['\_\_class\_\_', '\_\_contains\_\_', '\_\_delattr\_\_', '\_\_delitem\_\_', '\_\_dir\_\_', '\_\_doc\_\_', '\_\_eq\_\_', '\_\_format\_\_', '\_\_ge\_\_', '\_\_getattribute\_\_', '\_\_getitem\_\_', '\_\_gt\_\_', '\_\_hash\_\_', '\_\_init\_\_', '\_\_init\_subclass\_\_', '\_\_iter\_\_', '\_\_le\_\_', '\_\_len\_\_', '\_\_lt\_\_', '\_\_ne\_\_', '\_\_new\_\_', '\_\_reduce\_\_', '\_\_reduce\_ex\_\_', '\_\_repr\_\_', '\_\_setattr\_\_', '\_\_setitem\_\_', '\_\_sizeof\_\_', '\_\_str\_\_', '\_\_subclasshook\_\_', 'clear', 'copy', 'fromkeys', 'get', 'items', 'keys', 'pop', 'popitem', 'setdefault', 'update', 'values']

### 26.1 dict.fromkeys()

字典的[Fromkeys()函数](http://www.runoob.com/python/att-dictionary-fromkeys.html)用于创建一个新字典，以序列seq中元素做字典的键，value为字典所有键对应的初始值。

**函数语法**

dict.fromkeys(seq[, value])

**代码实例**

>>> dict1.fromkeys((1, 2, 3)) # 创建无value字典

{1: None, 2: None, 3: None}

>>> dict1.fromkeys((1, 2, 3), "Amusi")

{1: 'Amusi', 2: 'Amusi', 3: 'Amusi'} # 统一赋一样的值Amusi

>>> seq = ("name", "age", "sex")

>>> dict1 = dict.fromkeys(seq)

>>> dict1

{'name': None, 'age': None, 'sex': None}

>>> dict1 = dict.fromkeys(seq, 10) # 创建有value字典

>>> dict1

{'name': 10, 'age': 10, 'sex': 10}

>>> dict1 = dict.fromkeys(range(36), 0) # 初始化列表

>>> dict1

{0: 0, 1: 0, 2: 0, 3: 0, 4: 0, 5: 0, 6: 0, 7: 0, 8: 0, 9: 0, 10: 0, 11: 0, 12: 0, 13: 0, 14: 0, 15: 0, 16: 0, 17: 0, 18: 0, 19: 0, 20: 0, 21: 0, 22: 0, 23: 0, 24: 0, 25: 0, 26: 0, 27: 0, 28: 0, 29: 0, 30: 0, 31: 0, 32: 0, 33: 0, 34: 0, 35: 0}

### 26.2 dict.keys()

返回字典内所有的键key

**函数语法**

dict.keys()

无参数

返回值是字典内所有的键key

**代码实例**

>>> dict1 = {'name': 10, 'age': 10, 'sex': 10}

>>> for eachKey in dict1.keys():

print(eachKey)

name #全是键值

age

Sex

>>> dict1.keys()

dict\_keys(['name', 'age', 'sex'])

### 26.2 dict.values()

返回字典内所有的值value

**函数语法**

dict.values()

无参数

返回值是字典内所有的值value

**代码实例**

>>> dict1 = {"name":"Amusi", "age":23, "sex":"man"}

>>> for eachKey in dict1.values():

print(eachKey)

10

10

>>> dict1.values() # 输出所有值

dict\_values(['Amusi', 23, 'man'])

### 26.2 dict.items()

返回字典内所有的项item

**函数语法**

dict.items()

无参数

返回值是字典内所有的项item

**代码实例**

dict1 = {"name":"Amusi", "age":23, "sex":"man"}

>>> for eachKey in dict1.items():

print(eachKey)

('name', 10)

('age', 10)

('sex', 10)

>>> dict1.items()

dict\_items([('name', 'Amusi'), ('age', 23), ('sex', 'man')])

### 26.5 dict.clear()

字典(Dictionary)的[clear()函数](http://www.runoob.com/python/att-dictionary-clear.html)用于删除字典内所有元素

**函数语法**

dict.clear()

无参数

无返回值

**代码实例**

>>> dict1

{'name': 10, 'age': 10, 'sex': 10}

>>> dict1.clear()

>>> dict1

{}

### 26.6 dict.copy()

字典(Dictionary)的[copy()函数](http://www.runoob.com/python/att-dictionary-copy.html)返回一个字典的浅拷贝，即指针拷贝

注: 在有指针的情况下，浅拷贝只是增加了一个指针指向已经存在的内存，而深拷贝就是增加一个指针并且申请一个新的内存，使这个增加的指针指向这个新的内存

即在浅拷贝情况下，不同引用指向的是同一块内存，改其中一个引用，那么其他引用也会跟着改变

**函数语法**

dict.copy()

无参数

返回一个字典的浅拷贝（指针拷贝）

**代码实例**

# 创建字典1

>>> dict1 = {"name":"Amusi", "age":23, "sex":"man"}

>>> dict1

{'name': 'Amusi', 'age': 23, 'sex': 'man'}

# 浅拷贝

>>> dict2 = dict1

>>> dict2

{'name': 'Amusi', 'age': 23, 'sex': 'man'}

# 修改字典2某键对应的值

>>> dict2["age"] = 22

>>> dict2

{'name': 'Amusi', 'age': 22, 'sex': 'man'}

>>> dict1

{'name': 'Amusi', 'age': 22, 'sex': 'man'} # 字典1也相应变化

注：若要实现深拷贝，可以使用copy模块的copy.deepcopy函数

注：直接赋值和浅拷贝的区别

* 直接赋值——只是起了别名，但共用同一个指针和同一块内存
* 浅拷贝——不同指针，但指向同一块内存
* 深拷贝——不同指针，指向不同的内存块

>>> a = {"name":"Amusi", "age":23, "sex":"man"}

>>> b = a.copy() # 浅拷贝

>>> c =a # 直接赋值

>>> a

{'name': 'Amusi', 'age': 23, 'sex': 'man'}

>>> b

{'name': 'Amusi', 'age': 23, 'sex': 'man'}

>>> c

{'name': 'Amusi', 'age': 23, 'sex': 'man'}

>>> id(a)

140328819954168

>>> id(b)

140328756188360 # 不同指针

>>> id(c)

140328819954168 # 相同指针

>>> c["height"] = 181

>>> a

{'name': 'Amusi', 'age': 23, 'sex': 'man', 'height': 181}

>>> b

{'name': 'Amusi', 'age': 23, 'sex': 'man'} # 不变

>>> c

{'name': 'Amusi', 'age': 23, 'sex': 'man', 'height': 181}

>>> b["weight"] = 65

>>> a

{'name': 'Amusi', 'age': 23, 'sex': 'man', 'height': 181}

>>> b

{'name': 'Amusi', 'age': 23, 'sex': 'man', 'weight': 65}

>>> c

{'name': 'Amusi', 'age': 23, 'sex': 'man', 'height': 181}

### 26.7 dict.get()

字典的[get()函数](http://www.runoob.com/python/att-dictionary-get.html)返回指定键的值，如果值不在字典中则返回默认值None

**函数语法**

dict.get(key, default = None)

**参数**

* key——字典中要查询的键
* default——如果指定键不存在时，返回该默认值

**返回值**

返回指定键的值，如果值不在字典中则返回默认值None

**代码实例**

>>> dict1 = {"name":"Amusi", "age":23, "sex":"man"}

>>> print(dict1.get("name"))

Amusi

>>> print(dict1.get("height")) # 输出字典中不存在的键值

None

### 26.8 用in代替has\_key方法()

字典的has\_key函数是python2.x中函数，在Python3.x中用in代替has\_key函数。功能都是用于判断键是否存在于字典中，如果键在字典中，则返回true，否则返回false

**函数语法**

key in dict

**参数**

key——要在字典中查找的键

**返回值**

如果键在字典中返回true，否则返回false

**实例代码**

>>> dict1

{'name': 'Amusi', 'age': 23, 'sex': 'man', 'height': 181}

>>> "age" in dict1

True

>>> "name" in dict1

True

>>> "weight" in dict1 # 不存在

False

### 26.9 dict.pop()

字典中的pop()函数是如果键在字典中则删除该键和对应的值，否则有错误

**函数语法**

dict.pop(key)

**参数**

key——要在字典中删除的键

**返回值**

如果键在字典中则删除该键和对应的值，返回键对应的值，反之错误

**代码实例**

>>> dict1

{'name': 'Amusi', 'age': 23, 'sex': 'man', 'height': 181}

>>> dict1.pop("height")

181

>>> dict1

{'name': 'Amusi', 'age': 23, 'sex': 'man'}

>>> dict1.pop("weight") # 删除字典中没有的键

Traceback (most recent call last):

File "<pyshell#55>", line 1, in <module>

dict1.pop("weight")

KeyError: 'weight'

### 26.10 dict.popitem()

字典中的popitem()函数是随机从字典中弹出一个项item

**函数语法**

dict.popitem()

**参数**

无

**返回值**

从字典中随机弹出的项

**代码实例**

>>> dict1

{'name': 'Amusi', 'age': 23, 'sex': 'man'}

>>> dict1.popitem()

('sex', 'man') # 随机弹出的项（并不是刻意从尾部弹出）

>>> dict1

{'name': 'Amusi', 'age': 23}

### 26.11 dict.setdefault()

字典中的[setdefault()函数](http://www.runoob.com/python/att-dictionary-setdefault.html)功能和get方法类似，查找键是否在字典中，如果在则返回键对应的值。但当键不存在于字典中，则会添加键并将值设为默认值

dict.setdefault(key, default=None)

**参数**

* + key——查找
  + default——键不存在时，设置的默认键值

**返回值**

如果字典中包含查询的键，则返回该键对应的值，否则返回为该键设置的值

**代码实例**

>>> dict1 = {'name': 'Amusi', 'age': 23, 'sex': 'man'}

>>> dict1.setdefault("name") # 查询已有的键

'Amusi'

>>> dict1.setdefault("weight") # 查询不存在的键，并赋空值

>>> dict1

{'name': 'Amusi', 'age': 23, 'sex': 'man', 'weight': None}

>>> dict1.setdefault("girlfriend", "Lucy") # 查询不存在的键，并赋初始值

'Lucy'

>>> dict1

{'name': 'Amusi', 'age': 23, 'sex': 'man', 'weight': None, 'girlfriend': 'Lucy'}

### 26.12 dict.update()

字典中的update()函数是把字典dict2的键/值更新到dict1里

**函数语法**

dict.update(dict2)

**参数**

dict2——在指定字典dict的尾部添加子字典dict2

**返回值**

无返回值

**代码实例**

>>> dict1

{'name': 'Amusi', 'age': 23, 'sex': 'man', 'weight': None, 'girlfriend': 'Lucy'}

>>> dict1.pop("weight") # 弹出一对键/值

>>> dict1

{'name': 'Amusi', 'age': 23, 'sex': 'man', 'girlfriend': 'Lucy'}

>>> dict2 = {"weight":65, "degree":"master"}

>>> dict1.update(dict2) # 在dict1尾部添加子字典

>>> dict1

{'name': 'Amusi', 'age': 23, 'sex': 'man', 'girlfriend': 'Lucy', 'weight': 65, 'degree': 'master'}

>>> dict3 = {"age":24, "weight":70}

>>> dict1.update(dict3) # 更新已有的键值

>>> dict1

{'name': 'Amusi', 'age': 24, 'sex': 'man', 'girlfriend': 'Lucy', 'weight': 70, 'degree': 'master'}

### 练习题

1. Python的字典是否支持一键(key)多值(value)？

答： 不支持。字典是一对一映射的关系，而键是唯一的，因此值也就是唯一的

>>> dict1 = {"name":"Amusi"}

>>> dict1["age"] = 23

>>> dict1

{'name': 'Amusi', 'age': 23}

>>> dict1["name"] = "Lucy"

>>> dict1

{'name': 'Lucy', 'age': 23}

>>> dict1 = {"name":"Amusi", "name":"Lucy"}

>>> dict1

{'name': 'Lucy'}

1. 在字典中，如果试图为一个不存在的键(key)赋值会怎么样？

答： 会自动创建新的一对键/值

>>> dict1 = {"name":"Amusi"}

>>> dict1

{'name': 'Amusi'}

>>> dict1["age"] = 23

>>> dict1

{'name': 'Amusi', 'age': 23}

1. 成员资格操作符(in和not in)可以检查一个元素是否存在序列中，当然也可以用来检查一个键(key)是否存在字典中，那么请问哪种检查效率更高些？为什么？

答： 检查一个键(key)是否存在字典中 比 检查一个元素是否存在序列中更高效

因为前者是用哈希算法实现，时间复杂度为o(1)

1. Python对键(key)和值(value)有没有类型限制

答： 必须是可哈希(Hash)的对象，但对一般的变量类型(str, int, float)都支持

>>> dict1 = {"name":"Amusi", "age":23, 1:"Dream", 1.231:"Random"}

>>> dict1

{'name': 'Amusi', 'age': 23, 1: 'Dream', 1.231: 'Random'}

>>> print(dict1["name"])

Amusi

>>> print(dict1[1])

Dream

1. 请目测下列代码执行后，dict1的内容是什么?

dict1.fromkeys((1, 2, 3), ("one", "two", "three"))

dict1.fromkeys((1, 3), "数字")

答： 此题考察fromkeys函数用法，是创建新的字典

>>> dict1.fromkeys((1, 2, 3), ("one", "two", "three"))

{1: ('one', 'two', 'three'), 2: ('one', 'two', 'three'), 3: ('one', 'two', 'three')}

>>> dict1.fromkeys((1, 3), "数字")

{1: '数字', 3: '数字'}

1. 如果你需要将字典dict1 = {1:”one”, 2:”two”, 3:”three”}拷贝到dict2，如何做？

答： 使用字典的copy()函数

>>> dict1 = {1:"one", 2:"two", 3:"three"}

>>> dict1

{1: 'one', 2: 'two', 3: 'three'}

>>> dict2 = dict1.copy()

>>> dict2

{1: 'one', 2: 'two', 3: 'three'}

## 课时27 集合: 在我的世界里，你就是唯一！

[集合(set)函数](http://www.runoob.com/python/python-func-set.html)创建一个无序不重复的元素集，可进行关系测试，删除重复数据，还可计算交集、差集、并集等。

**函数语法**

class set([iterable])

**参数**

iterable——可迭代对象

**返回值**

返回新的集合对象

**代码实例**

# 字典

>>> num = {}

>>> type(num)

<class 'dict'>

# 集合

>>> num = {1, 2, 3}

>>> type(num)

<class 'set'>

# 集合操作

# 创建集合set

>>> x = set("Amusi")

>>> y = set("Lucy")

>>> x, y

({'A', 'm', 'i', 's', 'u'}, {'L', 'u', 'y', 'c'})

>>> x & y # 交集

{'u'}

>>> x | y # 并集

{'s', 'A', 'm', 'i', 'u', 'L', 'y', 'c'}

>>> x - y # 差集

{'s', 'A', 'i', 'm'}

# 集合是不重复的

>>> num2 = {1, 2, 2, 3, 2, 1, 9, 10, 9, 8, 100}

>>> num2

{1, 2, 3, 100, 8, 9, 10}

# 集合是无序的，所以不能用索引访问

>>> num2[6]

Traceback (most recent call last):

File "<pyshell#19>", line 1, in <module>

num2[6]

TypeError: 'set' object does not support indexing

# 集合set是无序的，并不会将元素排序

>>> num1 = {9, 8, 7, 6, 6, 5, 4, 9, 3, 8, 2, 4, 1}

>>> num1

{1, 2, 3, 4, 5, 6, 7, 8, 9}

>>> num1 = set([1, 3, 8, -2, 99, 98, 77, 1, 5, 3, 77, 12])

>>> num1

{1, 98, 3, 99, 5, 8, 12, 77, -2}

>>> num1 = {1, 3, 8, -2, 99, 98, 77, 1, 5, 3, 77, 12}

>>> num1

{1, 98, 3, 99, 5, 8, 12, 77, -2}

**课堂练习**

1. 要求去掉列表中重复的元素[0, 1, 2, 3, 4, 5, 5, 3, 1]

# 常规方法

# 可

>>> num1 = [0, 1, 2, 3, 4, 5, 5, 3, 1]

>>> temp = []

>>> for each in num1:

if each not in temp:

temp.append(each)

>>> temp

[0, 1, 2, 3, 4, 5]

# set方法

>>> num1 = [0, 1, 2, 3, 4, 5, 5, 3, 1]

>>> num1 = set(num1)

>>> num1

{0, 1, 2, 3, 4, 5}

>>> num1 = list(num1)

>>> num1

[0, 1, 2, 3, 4, 5]

1. 如何访问集合中的值

# 可以使用for把集合中的数据一个个读取出来

# 可以通过in和not判断一个元素是否在集合中已经存在

>>> num1 = {1, 2, 3, 4, 6}

>>> num1

{1, 2, 3, 4, 6}

>>> 1 in num

True

>>> 9 in num

False

>>> 9 not in num

True

>>> num1.add(9) # 向set添加不存在的元素

>>> num1

{1, 2, 3, 4, 6, 9}

>>> num1.add(1) # 向set添加已存在的元素(无变化)

>>> num1

{1, 2, 3, 4, 6, 9}

>>> num1.remove(4) # 删除set元素

>>> num1

{1, 2, 3, 6, 9}

**frozenset不可修改集合**

**函数语法**

class set([iterable])

**参数**

iterable——可迭代对象

**返回值**

返回新的不可修改的集合对象

**代码实例**

**#** frozenset

>>> set1 = frozenset({1, 2, 3, 4, 5, 6})

>>> set1

frozenset({1, 2, 3, 4, 5, 6})

# set

>>> set2 = set({1, 2, 3, 4, 5, 6})

>>> set2

{1, 2, 3, 4, 5, 6}

>>> set1.add(9) # 不可修改

Traceback (most recent call last):

File "<pyshell#4>", line 1, in <module>

set1.add(9)

AttributeError: 'frozenset' object has no attribute 'add'

>>> set2.add(9) # 可修改

>>> set2

{1, 2, 3, 4, 5, 6, 9}

### 练习题

1. 请问集合的唯一作用是什么？

答: 集合的作用就是确保元素的唯一性

1. 如果你希望创建的集合是不变的，应该怎么做？

答: 创建frozenset不可变集合

1. 请问如何确定一个集合里面有多少个元素？

答: 使用len函数

>>> set1 = set({1, 2, 3, 4, 5, 6})

>>> len(set1)

6

>>> set1 = set({1, 2, "A", 4, 5, "m", 6, "L", "Amusi"})

>>> len(set1)

9

1. 请目测以下代码会打印什么内容？

>>> num\_set = set([1, 2, 3, 4, 5])

>>> num\_set[0]

答：

>>> num\_set = set([1, 2, 3, 4, 5])

>>> num\_set

{1, 2, 3, 4, 5}

>>> num\_set[0] # set是无序的，所以不支持索引访问

Traceback (most recent call last):

File "<pyshell#25>", line 1, in <module>

num\_set[0]

TypeError: 'set' object does not support indexing

1. 请问set1 = {[1, 2]}和set1 = set{[1, 2]}执行的结果一样吗？

答： 不一样，前者是错误的代码，因为列表不是可哈希的类型，而后者是创建集合(set)

>>> set1 = {[1, 2]}

Traceback (most recent call last):

File "<pyshell#0>", line 1, in <module>

set1 = {[1, 2]}

TypeError: unhashable type: 'list'

>>> set1 = set([1, 2])

>>> set1

{1, 2}

1. 打开IDLE，输入set1 = {1, 1.0}，你发现了什么？

答： 创建一个集合(set)，且里面只有一个元素。因为set是无序且不重复的

>>> set1 = {1, 1.0}

>>> set1

{1}

>>> type(set1)

<class 'set'>

1. 请问如何给集合(set)添加和删除元素？

答:

add()添加元素

remove()删除元素

>>> set1 = set([1, 2, 3, 4, 5, 6])

>>> set1.add(9)

>>> set1.remove(4)

>>> set1

{1, 2, 3, 5, 6, 9}

# 章节7 永久存储

## 课时28 文件: 因为懂你，所以永恒

### 28.1 open()打开文件

Python open()函数用于打开一个文件，创建一个file对象。可用help(open)查看

函数语法

open(name[, mode[, buffering]])

参数说明

* name: 一个包含了你要访问的文件名称的字典串值
* mode: 打开文件的模式: 只读、写入、追加等，详细可见下表
* buffering: 如果buffering的值被设为0，就不会有寄存。如果buffering的值取1，访问文件时会寄存行。如果将buffering的值设为大于1的整数，这就表明寄存器的缓冲大小。

表28.1 文件打开模式

|  |  |
| --- | --- |
| 打开模式 | 执行操作 |
| ‘r’ | 以只读方式打开文件（默认） |
| ‘w’ | 以写入的方式打开文件，会覆盖已存在的文件 |
| ‘x’ | 如果文件已经存在，使用此模式打开将引发异常 |
| ‘a’ | 以写入模式打开，如果文件存在，则在末尾追加写入 |
| ‘b’ | 以二进制模式打开文件 |
| ‘t’ | 以文本模式打开（默认） |
| ‘+’ | 可读写模式（可添加到其它模式中使用） |
| ‘U’ | 通用换行符支持 |

表28.2 文件对象方法

|  |  |
| --- | --- |
| 打开模式 | 执行操作 |
| f.close() | 关闭文件 |
| f.read([size=-1]) | 从文件读取size个字符，当未给定size或给定负数时，读取剩余的所有字符，然后作为字符串返回 |
| f.readline([size=-1]) | 从文件中读取并返回一行（包括行结束符），如果size有定义，则返回size个字符 |
| f.write(str) | 将字符串str写入文件 |
| f.writeline(seq) | 向文件写入字符串序列seq，seq应该是一个返回字符串的可迭代对象 |
| f.seek(offset,from) | 在文件中移动文件指针，从from（0代表文件起始位置，1代表当前位置，2代表文件末尾）偏移offese个字节 |
| f.tell() | 返回当前在文件的位置 |
| f.truncate([size=file.tell()]) | 截取文件size个字节，默认是截取到文件指针当前位置 |

**代码实例**

首先创建record.txt文件，可以在里面写任意内容，如：

每一天

每一夜

每一晚

都值得

我们为之奋斗！

加油，Amusi

record.txt文件被我保存在”/home/amusi/Projects/Python/零基础入门学python/课时28 文件: 因为懂你，所以永恒”文件夹下

# 代码

>>> f = open("/home/amusi/Projects/Python/零基础入门学python/课时28 文件: 因为懂你，所以永恒/record.txt") # 注意是绝对路径

**# 返回文件指针的当前位置**

>>> f.tell()

0

**# 因为打开模式不对，所以不能写入**

>>> f.write("Fighting!")

Traceback (most recent call last):

File "<pyshell#10>", line 1, in <module>

f.write("Fighting!")

io.UnsupportedOperation: not writable

# 读取文件中前3个字符

>>> f.read(3)

'每一天'

**# 文件指针移动，再读取3个字符**

>>> f.read(3)

'\n每一'

**# 返回文件指针的当前位置**

>>> f.tell()

16

>>> f = open("/home/amusi/Projects/Python/零基础入门学python/课时28 文件: 因为懂你，所以永恒/record.txt")

**# 文件指针从0开始偏移10位**

>>> f.seek(10 ,0)

10

**# 从文件中按行读取**

>>> f.readline()

'每一夜\n'

**# 列表对文件操作**

>>> list(f)

['每一晚\n', '都值得\n', '我们为之奋斗！\n', '加油，Amusi']

**# 逐行打印文件内容**

>>> f = open("/home/amusi/Projects/Python/零基础入门学python/课时28 文件: 因为懂你，所以永恒/record.txt")

>>> f.seek(0, 0) # 重定位

0

>>> for each\_line in f:

print(each\_line)

每一天

每一夜

每一晚

都值得

我们为之奋斗！

加油，Amusi

**# 文件的写入**

>>> f = open("/home/amusi/Projects/Python/零基础入门学python/课时28 文件: 因为懂你，所以永恒/record.txt", "w")

>>> f.write("Amusi love Lucy!")

16

>>> f.close()

>>> f = open("/home/amusi/Projects/Python/零基础入门学python/课时28 文件: 因为懂你，所以永恒/record.txt")

>>> f.readline()

'Amusi love Lucy!'

### 练习题

1. 下边只有一种方式不能打开文件，请问是哪一种？为什么？

答： B选项。Python可以接受斜线”/”，也可以接受反斜线”\”，但反斜线必须出现两次才有效，即”\\”等价于”/”

1. 打开一个文件我们使用open()函数，通过设置文件的打开模式，决定打开的文件具有哪些性质，请问默认的打开模式是什么呢？

答： 默认是’rt’，只读的文本模式

1. 请问>>>open(“E\\Test.bin”, “xb”)是以什么样的模式打开文件的？

答： ‘xb”，可读写的二进制文件模式

1. 尽管Python有所谓的“垃圾回收机制”，但对于打开了的文件，在不需要用到的时候我们仍然需要使用f.close()将文件对象“关闭”，这是为什么呢？

答： Python拥有垃圾回收机制，会在文件对象的引用计数降为0的时候自动关闭文件，所以在Python编程里，如果忘记关闭文件，不会造成危险的内存泄漏。

但应该养成好习惯，打开一个文件后，应在完成写入后关闭文件。因为Python虽然会缓存数据，但当出现意外，如断电，那么缓存的数据并不会写入文件中。

1. 如何将一个文件对象（f）中的数据存放到列表中？

答： 可直接使用list

>>> list1 = list(f)

>>> list1

['Amusi love Lucy!']

1. 如何迭代打印出文件对象（f）中的每一行数据？

答：

for each\_line in f:

print(each\_line)

1. 文件对象的内置方法f.read([size=-1])作用是读取文件对象内容，size对象是可选的，那如果设置了size=10，例如f.read(10)，将返回什么内容呢？

答： 返回从当前文件指针指向的元素，并偏移10个长度，这段文件指针所指向的元素值

1. 如果获得文件对象（f）当前文件指针的位置？

答： 使用f.seek()函数

## 课时29 文件: 一个任务

**任务：将文件（record.txt）中的数据进行分割并按照以下规律保存起来：**

* Amusi的对话单独保存为boy\_\*.txt的文件（去掉“Amusi:”）
* Lucy的对话单独保存为girl\_\*.txt的文件（去掉Lucy:）
* 文件总共有三段对话，分别保存为boy\_1.txt，girl\_1.txt，boy\_2.txt，girl\_2.txt，boy\_3.txt，girl\_3.txt共6个文件（提示：文件中不同的对话间已经使用“==========”分割）

1. 新建record.txt，里面写入

Amusi:干嘛呢？

Lucy:做教案

Amusi:吃饭了么？

Lucy:恩，刚吃过，吃的饺子，你呢？

Amusi:我呀，吃得面条

Lucy:嘿嘿，很赞

===============================

Amusi:下班了么

Lucy:恩，刚下班了

Amusi:走到哪啦？

Lucy:南京路，准备坐地铁啦，你在干嘛呢？

Amusi:我呀，还在加班工作呢

===============================

Lucy:你看那个狗狗，好萌啊！

Amusi:哈哈，感觉像是哈士奇

Lucy:应该不是，我看像牧羊犬

Amusi:昨天买的哈密瓜吃完了么？

Lucy:恩，吃完啦

Amusi:走，那个有个水果店，去买点水果

===============================

**代码实例——入门（无函数）**

f = open("record.txt") # 按相对路径打开文件

boy = []

girl = []

count = 1

for each\_line in f:

if each\_line[:6] != "======":

# 我们这里使用split进行字符串分割操作

(role, line\_spoken) = each\_line.split(":", 1)

if role == "Amusi":

boy.append(line\_spoken)

if role == "Lucy":

girl.append(line\_spoken)

else:

# 创建文件名称

file\_name\_boy = "boy\_"+ str(count) + ".txt"

file\_name\_girl = "girl\_" + str(count) + ".txt"

# 打开文件

boy\_file = open(file\_name\_boy, "w")

girl\_file = open(file\_name\_girl, "w")

# 按行写入

boy\_file.writelines(boy)

girl\_file.writelines(girl)

# 关闭文件

boy\_file.close()

girl\_file.close()

# 重新初始化

boy = []

girl = []

count += 1

f.close

**代码实例1——进阶（两个函数解决）**

def save\_file(boy, girl, count):

# 创建文件名称

file\_name\_boy = "boy\_"+ str(count) + ".txt"

file\_name\_girl = "girl\_" + str(count) + ".txt"

# 打开文件

boy\_file = open(file\_name\_boy, "w")

girl\_file = open(file\_name\_girl, "w")

# 按行写入

boy\_file.writelines(boy)

girl\_file.writelines(girl)

# 关闭文件

boy\_file.close()

girl\_file.close()

def split\_file(file\_name):

f = open(file\_name) # 按相对路径打开文件

boy = []

girl = []

count = 1

for each\_line in f:

if each\_line[:6] != "======":

# 我们这里使用split进行字符串分割操作

(role, line\_spoken) = each\_line.split(":", 1)

if role == "Amusi":

boy.append(line\_spoken)

if role == "Lucy":

girl.append(line\_spoken)

else:

save\_file(boy, girl, count)

# 重新初始化

boy = []

girl = []

count += 1

f.close

split\_file("record.txt")

结果：创建6个文件，如下

**boy\_1.txt**

干嘛呢？

吃饭了么？

我呀，吃得面条

**girl\_1.txt**

做教案

恩，刚吃过，吃的饺子，你呢？

嘿嘿，很赞

**boy\_2.txt**

下班了么

走到哪啦？

我呀，还在加班工作呢

**girl\_2.txt**

恩，刚下班了

南京路，准备坐地铁啦，你在干嘛呢？

**boy\_3.txt**

哈哈，感觉像是哈士奇

昨天买的哈密瓜吃完了么？

走，那个有个水果店，去买点水果

**girl\_3.txt**

你看那个狗狗，好萌啊！

应该不是，我看像牧羊犬

恩，吃完啦

### 动动手

1. 编写一个程序，接受用户的输入并保存为新的文件：

答： 注意input()是遇到回车符即结束输入，所以可以写成死循环来接收输入

def create\_file(fileName):  
 f = open(fileName, "w") # 按相对路径打开文件  
 print("请输入内容[单独输入\":w\"保存退出]:\n")  
 while True:  
 writeLine = input()  
 if writeLine != "w:":  
 f.write("%s\n" % writeLine)  
 else:  
 break  
 f.close()  
  
fileName = input("请输入文件名: ")  
create\_file(fileName)

1. 编写一个程序，比较用户输入的两个文件，如果不同，显示出所有不同处的行号与第一个不同字符的位置：

答：

def compare\_file(file\_name1, file\_name2):  
 f1 = open(file\_name1) # 按相对路径打开文件  
 f2 = open(file\_name2) # 按相对路径打开文件  
 count = 0 # 计算不一样的数量  
 differ = [] # 统计不一样的行数  
 for line1 in f1:  
 line2 = f2.readline()  
 count += 1  
 if line1 != line2:  
 differ.append(count)  
 f1.close()  
 f2.close()  
  
 if len(differ) == 0:  
 print("两个文件完全一样")  
 else:  
 print("两个文件共有[%d]处不同:" % len(differ))  
 for each in differ:  
 print("第 %d 行不一样" % each)  
file\_name1 = input("请输入需要比较的第一个文件名: ")  
file\_name2 = input("请输入需要比较的第二个文件名: ")  
  
  
compare\_file(file\_name1, file\_name2)

1. 编写一个程序，当用户输入文件名和行数（N）后，将该文件的前N行内容打印到屏幕上

答：

def print\_file\_lines(file\_name1, N):  
 if N <= 0:  
 print("请输入大于1的行数!")  
 return  
 f = open(file\_name1) # 按相对路径打开文件  
 count = 0 # 计算不一样的数量  
 differ = [] # 统计不一样的行数  
 for line in f:  
 count += 1  
 if count <= N:  
 print(line)  
 f.close()  
  
  
file\_name = input("请输入文件名: ")  
N = int(input("请输入需要打印的行数: "))  
  
print\_file\_lines(file\_name, N)

1. 编写一个程序，实现”全部替换”功能。如将record.txt中的Lucy全部替换成Trista

**record.txt**

Amusi:干嘛呢？

Lucy:做教案

Amusi:吃饭了么？

Lucy:恩，刚吃过，吃的饺子，你呢？

Amusi:我呀，吃得面条

Lucy:嘿嘿，很赞

===============================

Amusi:下班了么

Lucy:恩，刚下班了

Amusi:走到哪啦？

Lucy:南京路，准备坐地铁啦，你在干嘛呢？

Amusi:我呀，还在加班工作呢

===============================

Lucy:你看那个狗狗，好萌啊！

Amusi:哈哈，感觉像是哈士奇

Lucy:应该不是，我看像牧羊犬

Amusi:昨天买的哈密瓜吃完了么？

Lucy:恩，吃完啦

Amusi:走，那个有个水果店，去买点水果

===============================

答：

def Allreplace(file\_name, old\_word, new\_word):  
 f = open(file\_name)  
 content = []  
 count = 0  
  
 for each in f:  
 if old\_word in each:  
 count += each.count(old\_word) # 计算每行出现的数量  
 each2 = each.replace(old\_word,new\_word) # 行替换后的内容  
 content.append(each2) # 替换后总的内容  
 f.close()  
 decide = input("\n文件 %s 中共有%s个[%s]\n您确定要把所有的[%s]替换成[%s]吗？\n[YES/NO]:" \  
 % (file\_name, count, old\_word, old\_word, new\_word))  
  
 if decide in ["YES", "Yes", "yes"]:  
 f\_write = open(file\_name, "w")  
 f\_write.writelines(content)  
 f\_write.close()  
 print("替换成功！")  
file\_name = input("请输入文件名: ")  
old\_word = input("请输入需要替换的单词或字符: ")  
new\_word = input("请输入新的单词或字符: ")  
  
Allreplace(file\_name, old\_word, new\_word)

## 课时30 文件系统: 介绍一个高大上的东西

### 30.1 模块简介

在前面几个章节中，我们都是用Python解释器IDLE或Pycharm来编程，但如果解释器退出后再进入，则之前定义的所有方法和变量都消失了。

为此Python提供一个方法，把这些定义存在在文件中，为一些脚本或交互式的解释器实例使用，这个文件被称为[模块（module）](http://www.runoob.com/python3/python3-module.html)。

[模块](https://www.liaoxuefeng.com/wiki/001374738125095c955c1e6d8bb493182103fac9270762a000/0013868200171577d6385bb5b4f4875bee9cbf0f0fa29c5000)是一个包含所有你定义的函数和变量的文件，其后缀名为.py。模块可以被别的程序引入，以[使用该模块中的函数等功能。](http://blog.csdn.net/hanzihan123/article/details/41898643)

[注： Python标准库（经典模块）](http://blog.csdn.net/hanzihan123/article/details/41898643)

**代码实例**

# 未导入模块

>>> serect = random.ranint(1, 10)

Traceback (most recent call last):

File "<pyshell#0>", line 1, in <module>

serect = random.ranint(1, 10)

NameError: name 'random' is not defined

# import导入random模块

>>> import random

>>> secret = random.randint(1, 10)

>>> secret

1

### 30.2 os模块

表30.2.1 os模块中关于文件/目录常用的函数使用方法

|  |  |
| --- | --- |
| 函数名 | 使用方法 |
| getcwd() | 返回当前的工作目录 |
| chdir(path) | 改变工作目录 |
| listdir(path=”.”) | 列举指定目录中的文件名（“.”表示当前目录，”..”表示上一级目录） |
| mkdir(path) | 创建单层目录，如该目录已存在则抛出异常 |
| makedirs(path) | 递归创建多层目录，如该目录已存在则抛出异常，注意：”E:\\a\\b”和”E:\\a\\c”并不冲突 |
| remove(path) | 删除文件 |
| rmdir(path) | 删除单层目录，如该目录非空则抛出异常 |
| removedirs(path) | 递归删除目录，从子目录到父目录逐层尝试删除，遇到目录非空则抛出异常 |
| rename(old, new) | 将文件old重名为new |
| system(command) | 运行系统的shell命令 |
| walk(top) | 遍历top路径下所有的子目录，返回一个三元组：(路径, [包含目录],[包含文件]) |
| 以下是支持路径操作中常用到的一些定义，支持所有平台 | |
| os.curdir | 指代当前目录（“.”） |
| os.pardir | 指代上一级目录（”..“） |
| os.sep | 输出操作系统特定的路径分隔符（Win下为”\\”，Linux下为”/”） |
| os.linesep | 当前平台使用的行终止符（Win下为”\r\n”，Linux下为”\n”） |
| os.name | 指代当前使用的操作系统（包括：”posix”, “nt”, “mac”, “ce”） |

表30.2.2 os模块中关于路径常用的函数使用方法

注：要使下述函数，需要调用os.path.\*\*\*

|  |  |
| --- | --- |
| 函数名 | 使用方法 |
| basename(path) | 去掉目录路径，单独返回文件名 |
| dirname(path) | 去掉文件名，单独返回目录路径 |
| join(path1[path2[,...]]) | 将path1, path2, ...各部分组成合成一个路径名 |
| split(path) | 分割文件名与路径，返回（f\_path,f\_name）元组。如果完全使用目录，它也会将最后一个目录作为文件名分离，且不会判断文件或者目录是否存在 |
| splitext(path) | 分离文件名与扩展名，返回（f\_name,f\_extension）元组 |
| getsize(file) | 返回指定文件的尺寸，单位是字节 |
| getatime(file) | 返回指定文件最近的访问时间（浮点型整数，可用time模块的gmtime()或localtime函数换算） |
| getctime(file) | 返回指定文件的创建时间（浮点型整数，可用time模块的gmtime()或localtime函数换算） |
| getmtime(file) | 返回指定文件最新的修改时间（浮点型整数，可用time模块的gmtime()或localtime函数换算） |
| 以下函数返回True或False | |
| exists(path) | 判断指定路径（目录或文件）是否存在 |
| isabs(path) | 判断指定路径是否为绝对路径 |
| isdir(path) | 判断指定路径是否存在且是一个目录 |
| isfile(path) | 判断指定路径是否存在且是一个文件 |
| islink(path) | 判断指定路径是否存在且是一个符号链接 |
| ismount(path) | 判断指定路径是否存在且是一个挂载点 |
| samefile(path1, path2) | 判断path1和path2两个路径是否指向同一个文件 |

**课堂编程实战**

# 导入os模块

>>> import os

# 获得当前工作目录

>>> os.getcwd()

'/home/amusi'

# 改变工作目录

>>> os.chdir("/home/amusi/Projects")

>>> os.getcwd()

'/home/amusi/Projects'

# 列出指定目录中的文件名

>>> os.listdir(os.getcwd())

['NanRui', 'Qt', 'Python', 'Jhead', 'C++', 'CUDA', 'Images', 'OpenCV\_Python', 'tensorflow', 'OpenCV', 'Stitch']

# 创建单层目录

>>> os.mkdir("PythonTest")

# 创建多级目录

>>> os.makedirs("PythonTest/Test1")

>>> 删除路径

>>> os.rmdir("PythonTest/Test1")

>>> os.rmdir("PythonTest")

# 重命名文件名

>>> os.mkdir("Test")

>>> os.rename("Test", "TestNew")

>>> os.rmdir("TestNew")

# 指定当前目录（“.”）

>>> os.curdir

'.'

>>> os.listdir(os.curdir)

['NanRui', 'Qt', 'Python', 'Jhead', 'C++', 'CUDA', 'Images', 'OpenCV\_Python', 'tensorflow', 'OpenCV', 'Stitch']

# 指定上一级目录（“..”）

>>> os.pardir

'..'

# 路径分割符（“/”）

>>> os.sep

'/'

# 换行符（“\n”）

>>> os.linesep

'\n'

# 查看系统环境（“posix”表示linux）

>>> os.name

'posix'

# 去掉目录路径，单独返回文件名

>>> os.path.basename("/home/amusi/test.txt")

'test.txt'

# 去掉文件名，单独返回路径

>>> os.path.dirname("/home/amusi/test.txt")

'/home/amusi'

# 将path1、path2...组成一个路径名

>>> os.path.join("A", "B", "C", "D")

'A/B/C/D'

# 分离文件名和路径

>>> os.path.split("/home/amusi/test.txt")

('/home/amusi', 'test.txt')

# 分离文件名与与扩展名

>>> os.path.splitext("home/amusi/test.txt")

('home/amusi/test', '.txt')

# 返回指定文件的大小(byte)

>>> os.path.getsize("/home/amusi/test.txt")

13

# 返回指定文件最近的访问时间

>>> os.path.getatime("/home/amusi/test.txt")

1509811376.5588443

# 利用time模块转换时间

>>> import time

>>> time.gmtime(os.path.getatime("/home/amusi/test.txt"))

time.struct\_time(tm\_year=2017, tm\_mon=11, tm\_mday=4, tm\_hour=16, tm\_min=2, tm\_sec=56, tm\_wday=5, tm\_yday=308, tm\_isdst=0)

>>> time.localtime(os.path.getatime("/home/amusi/test.txt"))

time.struct\_time(tm\_year=2017, tm\_mon=11, tm\_mday=5, tm\_hour=0, tm\_min=2, tm\_sec=56, tm\_wday=6, tm\_yday=309, tm\_isdst=0)

# 返回指定文件创建的时间

>>> time.localtime(os.path.getctime("/home/amusi/test.txt"))

time.struct\_time(tm\_year=2017, tm\_mon=11, tm\_mday=5, tm\_hour=0, tm\_min=2, tm\_sec=56, tm\_wday=6, tm\_yday=309, tm\_isdst=0)

# 判断路径/目录/文件是否存在

>>> os.path.exists("/home/amusi/test.txt")

True

# 判断指定路径是否存在是一个目录

>>> os.path.isdir("/home/amusi/test.txt")

False

# 判断指定路径是否存在是一个文件

>>> os.path.isfile("/home/amusi/test.txt")

True

# 判断指定路径是否存在是一个挂载点

>>> os.path.ismount("/home")

True

>>> os.path.ismount("/home/amusi/")

False

### 动动手

1. 编写一个程序，统计当前目录下每个文件类型的文件数

答： 使用os模块，使用os.walk内建函数，使用split函数

import os  
def count\_files(file\_cur\_dir):  
 files = []  
 extens = {}  
 for root, dirs, files in os.walk(file\_cur\_dir):  
 print(root) # 当前目录  
 print(dirs) # 当前路径下的所有子目录  
 print(files) # 当前路径下所有非目录子文件  
 # 将文件名和后缀分开  
 for each in files:  
 # 常规方法  
 # f\_name, f\_extension = each.split(".", 1)  
 # 使用os.path.split函数  
 f\_name, f\_extension = os.path.splitext(each)  
 if f\_extension not in extens:  
 extens[f\_extension] = 0  
 extens[f\_extension] = extens[f\_extension] + 1  
 # 输出结果  
 for extens\_name, nums in extens.items():  
 #print("该文件夹下共有类型为[" + extens\_name +"]的文件" + str(nums) + "个")  
 print("该文件夹下共有类型为[%s]的文件 %d 个" % (extens\_name, nums))  
  
file\_cur\_dir = os.getcwd()  
count\_files(file\_cur\_dir)

# 输出结果

该文件夹下共有类型为[txt]的文件 3 个

该文件夹下共有类型为[docx]的文件 2 个

该文件夹下共有类型为[png]的文件 1 个

该文件夹下共有类型为[py]的文件 1 个

该文件夹下共有类型为[doc]的文件 2 个

1. 编写一个程序，计算当前文件夹下所有文件的大小

答：

import os  
def count\_files(file\_cur\_dir):  
 files = []  
 extens = {}  
 for root, dirs, files in os.walk(file\_cur\_dir):  
 print(root) # 当前目录  
 print(dirs) # 当前路径下的所有子目录  
 print(files) # 当前路径下所有非目录子文件  
 #  
 for each\_file in files:  
 f\_size = os.path.getsize(each\_file)  
 print("%s [%dBytes]" % (each\_file, f\_size))  
  
file\_cur\_dir = os.getcwd()  
count\_files(file\_cur\_dir)

# 输出结果

text4.txt [530Bytes]

text1.docx [644Bytes]

text2.txt [395Bytes]

text8.png [644Bytes]

text6.docx [285Bytes]

review000.py [1064Bytes]

text3.doc [644Bytes]

review001.py [592Bytes]

text7.doc [0Bytes]

text5.txt [644Bytes]

1. 编写一个程序，用户输入文件名以及开始搜索的路径，搜索该文件是否存在。如遇到文件夹，则进入文件夹继续搜索

答：

import os  
def judge\_exit(file\_name, file\_path):  
 os.chdir(file\_path) # 改变工作目录  
 flag\_exit = 0 # 文件存在标志  
 # 判断每层目录的同级文件  
 for each\_file in os.listdir(os.curdir):  
 if each\_file == file\_name:  
 flag\_exit = 1  
 print(os.getcwd() + os.sep + each\_file) # 使用os.seq使程序更标准  
 if os.path.isdir(each\_file):  
 judge\_exit(file\_name, each\_file) # 递归调用  
 os.chdir(os.pardir) # 递归调用后切记返回上一层目录  
  
 if flag\_exit == 0:  
 print("很抱歉在[%s]下没有找到[%s]" % (file\_path, file\_name))  
  
file\_name = input("请输入您需要查询的文件名: ")  
file\_path = input("请输入您需要搜索的路径: ")  
judge\_exit(file\_name, file\_path)

# 输出结果

请输入您需要查询的文件名: demo.txt

请输入您需要搜索的路径: /home/test

/home/test/test2/demo.txt

/home/test/demo.txt

/home/test/test1/demo.txt

1. 编写一个程序，用户输入开始搜索的路径，查找该路径下（包含子文件夹内）所有的视频格式文件（要求查找mp4,rmvb,avi的格式即可），并把创建的一个文件（vedioList.txt）存放所有找到的文件的路径

答：

import os  
def search\_file(file\_path, target):  
 os.chdir(file\_path)  
  
 for each\_file in os.listdir(os.curdir):  
 ext = os.path.splitext(each\_file)[1]  
 if ext in target:  
 video\_list.append(os.getcwd() + os.sep + each\_file + os.linesep)  
  
 if os.path.isdir(each\_file):  
 search\_file(each\_file, target) # 递归调用  
 os.chdir(os.pardir) # 递归调用后切记返回上一层目录  
  
  
file\_path = input("请输入您需要搜索的路径: ")  
program\_dir = os.getcwd()  
  
target = [".mp4", ".avi", ".rmvb"]  
video\_list = []  
search\_file(file\_path, target)  
  
f = open(program\_dir + os.sep + "videoList.txt", "w")  
f.writelines(video\_list)  
f.close

1. 编写一个程序，用户输入关键字，查找当前文件夹内（如果当前文件夹内包含文件夹，则进入文件夹继续搜索）所有含有该关键字的文本文件（.txt后缀），要求显示该文件所在的路径以及关键字在文件中的具体位置（第几行第几个字符）

答：

import os # 加载模块  
  
# 输出位置  
def print\_pos(key\_dict):  
 #  
 keys = key\_dict.keys()  
 keys = sorted(keys) #由于字典是无序的，所以这里对行数进行排序  
 for each\_key in keys:  
 print("关键字出现在第 %s 行, 第 %s 个位置" % (each\_key, str(key\_dict[each\_key])))  
  
# key在每行对应的位置  
def pos\_in\_line(line, key):  
 pos = []  
 begin = line.find(key)  
 while begin != -1:  
 pos.append(begin + 1) # 代码的索引+1 = 用户直观的索引  
 begin = line.find(key, begin + 1) # 从下一个位置继续查找  
 return pos  
  
# 读取当前txt文件进行查询  
def search\_in\_file(file\_name, key):  
 f = open(file\_name)  
 count = 0 # 记录行数  
 key\_dict = dict()  
  
 # 遍历文件中的每一行  
 for each\_line in f:  
 count += 1  
 if key in each\_line:  
 pos = pos\_in\_line(each\_line, key) # key在每行对应的位置  
 key\_dict[count] = pos  
 f.close()  
 return key\_dict  
# 查询函数  
def search\_files(key, detail):  
 # 获得当前路径及子目录下所有文件名  
 all\_files = os.walk(os.getcwd())  
 txt\_files = []  
  
 # 遍历所有层目录下的所有文件  
 for i in all\_files:  
 # 遍历当前层目录下的所有文件  
 for each\_file in i[2]:  
 # 根据文件后缀判断是否为文本文件  
 if os.path.splitext(each\_file)[1] == ".txt":  
 # 添加文件的绝对路径  
 each\_file = os.path.join(i[0], each\_file)  
 txt\_files.append(each\_file)  
  
 # 对所有txt文件进行查询  
 for each\_txt\_file in txt\_files:  
 # 读取当前txt文件进行查询  
 key\_dict = search\_in\_file(each\_txt\_file, key)  
 if key\_dict:  
 print("=============================")  
 print("在文件[%s]中找到关键字[%s]" % (each\_txt\_file, key))  
 if detail in ["YES", "Yes", "yes"]:  
 print\_pos(key\_dict)  
  
key = input("请将脚本(.py)放在待查找的文件夹中，请输入待查询的关键字: ")  
detail = input("请问是否需要打印关键字[%s]在文件中的具体位置(YES/NO): " % key)  
search\_files(key, detail) # 查询

## 课时31 永久存储: 腌制一缸美味的泡菜pickle

[Python::pickle知识点](https://docs.python.org/3.6/library/pickle.html)

The [pickle](https://docs.python.org/3.6/library/pickle.html" \l "module-pickle" \o "pickle: Convert Python objects to streams of bytes and back.) module implements binary protocols for serializing and de-serializing a Python object structure.

在机器学习中，我们常常需要把训练好的模型存储起来，这样在进行决策时直接将模型读出，而不需要重新训练模型，这样就大大节约了时间。Python提供的pickle模块就很好地解决了这个问题，它可以序列化对象并保存到磁盘中，并在需要的时候读取出来，任何对象都可以执行序列化操作。

### 31.1 pickle.dump()

**函数语法**

pickle.dump(obj, file, [,protocol])

**参数**

* obj: 想要序列化的对象
* file: 已打开的文件名变量
* [,protocol]: 序列化所使用的协议

**返回值**

无

**函数功能**

将obj对象序列化存入已经打开的file中

**代码实例**

# 导入pickle模块

>>> import pickle

>>> my\_list = [123, 456, "Amusi", ["Lucy"]]

# 以写入二进制“wb”方式打开文件

>>> pickle\_file = open("my\_list.pkl", "wb")

# 使用pickle.dump

>>> pickle.dump(my\_list, pickle\_file)

>>> pickle\_file.close()

### 31.2 pickle.load()

**函数语法**

pickle.load(file)

**参数**

* file: 已打开的文件名变量

**返回值**

pickle对象

**函数功能**

将file中的对象序列化读出

**代码实例**

# 导入pickle模块

>>> import pickle

# 以只读二进制“rb”方式打开文件

>>> pickle\_file = open("my\_list.pkl", "rb")

# pickle.load读取

>>> my\_list2 = pickle.load(pickle\_file)

>>> print(my\_list2)

[123, 456, 'Amusi', ['Lucy']]

### 31.3 pickle.dumps()

**函数语法**

pickle.dumps(obj[, protocol])

**参数**

* obj: 想要序列化为str的obj对象
* protocal: 序列化所使用的协议

**返回值**

string序列化对象

**函数功能**

将obj对象序列化为string形式，而不是存入文件中

**代码实例**

>>> import pickle

>>> data\_list = [[1, 1, "yes"], [1, 1, "yes"], [1, 0, "no"], [0, 1, "no"], [0, 1, "no"]]

>>> p = pickle.dumps(data\_list)

>>> print(pickle.loads(p))

[[1, 1, 'yes'], [1, 1, 'yes'], [1, 0, 'no'], [0, 1, 'no'], [0, 1, 'no']]

### 31.4 pickle.loads()

**函数语法**

pickle.loads(string)

**参数**

* string: string序列变量名

**返回值**

string序列化前的对象

**函数功能**

从string中读取序列化前的obj对象

**代码实例**

>>> import pickle

>>> data\_list = [[1, 1, "yes"], [1, 1, "yes"], [1, 0, "no"], [0, 1, "no"], [0, 1, "no"]]

>>> p = pickle.dumps(data\_list)

>>> print(pickle.loads(p))

[[1, 1, 'yes'], [1, 1, 'yes'], [1, 0, 'no'], [0, 1, 'no'], [0, 1, 'no']]

### 练习题

1. pickle的本质是什么？

答: 将数据对象pickle成二进制（binary）文件，存储在磁盘上，也可以放到数据库中

1. 使用pickle的什么方法存储数据？

答: picle.dump(obj, file, [, protocal])

注：记得使用”wb”方式打开文件

1. 使用pickle的什么方法读取数据？

答: pickle.load(file)

1. 使用pickle能不能保存为”\*.txt”类型的文件

答: 可以，但打开后会出现乱码。因为pickel保存的文件是二进制文件，而”\*.txt”文件保存的是字符串文件，所以会显示乱码。

### 动动手

1. 编写一个程序，要求使用pickle模块将record.txt里的对话pickle成不同文件

# Summary: 编写一个程序，要求使用pickle模块将record.txt里的对话pickle成不同文件  
# Author: Fangjie Chen  
# Date: 2017-11-06  
  
import pickle  
  
def save\_file(boy, girl, count):  
 # 创建文件名称  
 file\_name\_boy = "boy\_"+ str(count) + ".pkl"  
 file\_name\_girl = "girl\_" + str(count) + ".pkl"  
  
 # 打开文件  
 boy\_file = open(file\_name\_boy, "wb")  
 girl\_file = open(file\_name\_girl, "wb")  
  
 # 使用pickle.dump按行写入  
 pickle.dump(boy, boy\_file)  
 pickle.dump(girl, girl\_file)  
  
 # 关闭文件  
 boy\_file.close()  
 girl\_file.close()  
def split\_file(file\_name):  
 f = open(file\_name) # 按相对路径打开文件  
  
 boy = []  
 girl = []  
 count = 1  
  
 for each\_line in f:  
 if each\_line[:6] != "======":  
 # 我们这里使用split进行字符串分割操作  
 (role, line\_spoken) = each\_line.split(":", 1)  
 if role == "Amusi":  
 boy.append(line\_spoken)  
 if role == "Lucy":  
 girl.append(line\_spoken)  
 else:  
 save\_file(boy, girl, count)  
  
 # 重新初始化  
 boy = []  
 girl = []  
 count += 1  
 f.close  
  
split\_file("record.txt")

# 章节8 异常处理

## 课时32 异常处理: 你不可能总是对的

参考[Python 异常处理](http://www.runoob.com/python/python-exceptions.html)

参考[Python 标准异常总结](http://bbs.fishc.com/forum.php?mod=viewthread&tid=45814&extra=page%3D1%26filter%3Dtypeid%26typeid%3D403)

**什么是异常？**

异常即是一个事件，该事件会在程序执行过程中发生，影响了程序的正常执行。

一般情况下，在Python无法正常处理程序时就会发生一个异常。

异常是Python对象，表示一个错误。

当Python脚本发生异常时我们需要捕获处理它，否则程序会终止执行。

**实例代码——AssertError**

>>> my\_list = ["Amusi"]

>>> assert len(my\_list) > 0

>>> my\_list.pop()

'Amusi'

>>> assert len(my\_list) > 0

Traceback (most recent call last):

File "<pyshell#3>", line 1, in <module>

assert len(my\_list) > 0

AssertionError

**实例代码——IndexError**

>>> my\_list = [1, 2, 3]

>>> my\_list[0]

1

>>> my\_list[3]

Traceback (most recent call last):

File "<pyshell#6>", line 1, in <module>

my\_list[3]

IndexError: list index out of range

**实例代码——SyntaxError**

>>> print "I love Lucy!"

SyntaxError: Missing parentheses in call to 'print'

**实例代码——TypeError**

>>> sums = 1 + "1"

Traceback (most recent call last):

File "<pyshell#8>", line 1, in <module>

sums = 1 + "1"

TypeError: unsupported operand type(s) for +: 'int' and 'str'

**实例代码——ZeroDivisionError**

>>> a = 9 / 0

Traceback (most recent call last):

File "<pyshell#9>", line 1, in <module>

a = 9 / 0

ZeroDivisionError: division by zero

表31.1 Python标准异常总结

|  |  |
| --- | --- |
| BaseException | 所有异常的基类 |
| SystemExit | 解释器请求退出 |
| KeyboardInterrupt | 用户中断执行(通常是输入ctrl + c) |
| Exception | 常规错误的基类 |
| StopIteration | 迭代器没有更多的值 |
| GeneratorExit | 生成器(generator)发生异常来通知退出 |
| StandardError | 所有的内建标准异常的基类 |
| ArithmeticError | 所有数值计算错误的基类 |
| FloatingPointError | 浮点计算错误 |
| OverflowError | 数值运算超出最大限制 |
| ZeroDivisionError | 除(或取模)零 (所有数据类型) |
| AssertionError | 断言语句（assert）失败 |
| AttributeError | 对象没有这个属性 |
| EOFError | 没有内建输入,到达EOF 标记 |
| EnvironmentError | 操作系统错误的基类 |
| IOError | 输入/输出操作失败 |
| OSError | 操作系统错误 |
| WindowsError | 系统调用失败 |
| ImportError | 导入模块/对象失败 |
| LookupError | 无效数据查询的基类 |
| IndexError | 序列中没有此索引(index) |
| KeyError | 映射中没有这个键 |
| MemoryError | 内存溢出错误(对于Python 解释器不是致命的) |
| NameError | 未声明/初始化对象 (没有属性) |
| UnboundLocalError | 访问未初始化的本地变量 |
| ReferenceError | 弱引用(Weak reference)试图访问已经垃圾回收了的对象 |
| RuntimeError | 一般的运行时错误 |
| NotImplementedError | 尚未实现的方法 |
| SyntaxError | Python 语法错误 |
| IndentationError | 缩进错误 |
| TabError | Tab 和空格混用 |
| SystemError | 一般的解释器系统错误 |
| TypeError | 对类型无效的操作 |
| ValueError | 传入无效的参数 |
| UnicodeError | Unicode 相关的错误 |
| UnicodeDecodeError | Unicode 解码时的错误 |
| UnicodeEncodeError | Unicode 编码时错误 |
| UnicodeTranslateError | Unicode 转换时错误 |
| Warning | 警告的基类 |
| DeprecationWarning | 关于被弃用的特征的警告 |
| FutureWarning | 关于构造将来语义会有改变的警告 |
| OverflowWarning | 旧的关于自动提升为长整型(long)的警告 |
| PendingDeprecationWarning | 关于特性将会被废弃的警告 |
| RuntimeWarning | 可疑的运行时行为(runtime behavior)的警告 |
| SyntaxWarning | 可疑的语法的警告 |
| UserWarning | 用户代码生成的警告 |

### 测试题

1. 结合你自身编程经验，总结下异常处理机制的重要性？

答: 方便开发者查看代码错误，增强程序的健壮性和用户体验

1. 请问以下代码是否会产生异常，如果会的话，请写出异常的名称：

>>> my\_list = [1, 2, 3, 4,,]

答: 语法错误，即提示SyntaxError: invalid syntax

1. 请问以下代码是否会产生异常，如果会的话，请写出异常的名称：

>>> my\_list = [1, 2, 3, 4, 5]

>>> print(my\_list[len(my\_list)])

答: 索引错误，即提示IndexError: list index out of range。因为索引值超出列表的个数，会找不到对应的元素

1. 请问以下代码是否会产生异常，如果会的话，请写出异常的名称：

>>> my\_list = [3, 5, 1, 4, 2]

>>> my\_list.sorted()

答: 属性错误，即提示AttributeError: 'list' object has no attribute 'sorted'。因为对于list，没有sorted方法

1. 请问以下代码是否会产生异常，如果会的话，请写出异常的名称：

>>> my\_dict = {"host":"http://amusi.com", "port":"80"}

>>> print(my\_dict["server"])

答: 键错误，即提示KeyError: 'server'。因为在my\_dict中没有”server”这个键

1. 请问以下代码是否会产生异常，如果会的话，请写出异常的名称：

>>> f(x=1, 2)

答: 语法错误，即提示SyntaxError: positional argument follows keyword argument

1. 请问以下代码是否会产生异常，如果会的话，请写出异常的名称：

>>> f = open("test.txt", "wb")

>>> f.write("Amusi love Lucy!\n")

>>> f.close()

答: 类型错误，即提示TypeError: a bytes-like object is required, not 'str'。因为上述代码是使用”wb”方式打开文件，即用二进制的方式写入数据，此时”Amusi love Lucy!\n”是字符串类型，所以出现类型错误。

1. 请问以下代码是否会产生异常，如果会的话，请写出异常的名称：

>>> def my\_fun1():

x = 5

def my\_fun2():

x \*= x

return x

return my\_fun2()

>>> my\_fun1()

答: 访问未初始化的局部变量错误，即UnboundLocalError: local variable 'x' referenced before assignment。但my\_fun1()调用my\_fun2()时，my\_fun2()中的x是局部变量，且没有初值，所以会出现错误。（my\_fun1函数与my\_fun2函数中都有x，但没有直接关联）

## 课时33 异常处理: 你不可能总是对的2

参考[Python 异常处理](http://www.runoob.com/python/python-exceptions.html)

### 33.1 try-except语句

**语法**

try:

检测范围

except Exception[as reason]:

出现异常（Exception）后的处理代码

**参数**

* Exception是异常的类型（如TypeError）

**代码实例**

* 新建test.py，写入下述代码

f = open("try\_except.txt")

print(f.read())

f.close()

* 因为当前目录下不存在try\_except.txt文件，所以会打开失败，弹出下述错误

RESTART: /home/amusi/Projects/Python/零基础入门学python/课时33 异常处理: 你不可能总是对的2/test.py

Traceback (most

Traceback (most recent call last):

File "/home/amusi/Projects/Python/零基础入门学python/课时33 异常处理: 你不可能总是对的2/test.py", line 1, in <module>

f = open("try\_except.txt")

FileNotFoundError: [Errno 2] No such file or directory: 'try\_except.txt'

**1. 捕获一个异常**

* 此时利用try-except结构体捕获异常，修改成下述代码

try:

f = open("try\_except.txt")

print(f.read())

f.close()

except OSError:

print("文件打开失败!")

* 再次运行test.py

RESTART: /home/amusi/Projects/Python/零基础入门学python/课时33 异常处理: 你不可能总是对的2/test.py

文件打开失败!

**2. 捕获一个异常并提示异常信息**

* 再次修改代码，在try-except结构体中加入as reason，可提示具体错误信息

try:

f = open("try\_except.txt")

print(f.read())

f.close()

except OSError as reason:

print("文件打开失败!\n错误的原因是: " + str(reason))

* 再次运行test.py

RESTART: /home/amusi/Projects/Python/零基础入门学python/课时33 异常处理: 你不可能总是对的2/test.py

文件打开失败!

错误的原因是: [Errno 2] No such file or directory: 'try\_except.txt'

**3. 捕获两个异常**

* 再次修改代码，加入TypeError异常，并捕获

try:

sum = 1 + "1"

f = open("try\_except.txt")

print(f.read())

f.close()

except OSError as reason:

print("文件打开失败!\n错误的原因是: " + str(reason))

except TypeError as reason:

print("类型出错啦!\n错误的原因是: " + str(reason))

* 再次运行test.py

RESTART: /home/amusi/Projects/Python/零基础入门学python/课时33 异常处理: 你不可能总是对的2/test.py

类型出错啦!

错误的原因是: unsupported operand type(s) for +: 'int' and 'str'

**4. 未能捕获没有设置的异常**

* 再次修改代码，加入ValueError异常，但except不捕获

try:

nums = int("abc")

sum = 1 + "1"

f = open("try\_except.txt")

print(f.read())

f.close()

except OSError as reason:

print("文件打开失败!\n错误的原因是: " + str(reason))

except TypeError as reason:

print("类型出错啦!\n错误的原因是: " + str(reason))

* 再次运行test.py（可知except不能捕获错误，但系统会自动提示ValueError）

RESTART: /home/amusi/Projects/Python/零基础入门学python/课时33 异常处理: 你不可能总是对的2/test.py

Traceback (most recent call last):

File "/home/amusi/Projects/Python/零基础入门学python/课时33 异常处理: 你不可能总是对的2/test.py", line 2, in <module>

nums = int("abc")

ValueError: invalid literal for int() with base 10: 'abc'

1. **捕获多个异常**

* 再次修改代码，将所要要捕获的Error类型写成元组的形式

try:

num = int("abc")

sum = 1 + "1"

f = open("try\_except.txt")

print(f.read())

f.close()

except (OSError, TypeError, ValueError):

print("出错啦!")

* 再次运行test.py

RESTART: /home/amusi/Projects/Python/零基础入门学python/课时33 异常处理: 你不可能总是对的2/test.py

出错啦!

**注：不推荐下述捕获的结构体类型**

try:

\*\*

except:

\*\*

### 33.2 try-finally语句

try-finally是确保一定要执行的代码，如f.close()

**语法**

try:

检测范围

except Exception[as reason]:

出现异常（Exception）后的处理代码

finally:

无论如何都会被执行的代码

**参数**

* Exception是异常的类型（如TypeError）

**代码实例**

* 新建test.py，写入下述代码

try:

f = open("try\_except.txt", "w")

print(f.write("我存在啦！"))

sums = 1 + "1"

f.close()

except (OSError, TypeError, ValueError):

print("出错啦!")

由于在sums = 1 + “1”会出现TypeError异常，所以f.close()被略过，此时新建的try\_except.txt文件中没有“我存在啦”的内容。

* 加入finally:

try:

f = open("try\_except.txt", "w")

print(f.write("我存在啦！"))

sums = 1 + "1"

except (OSError, TypeError, ValueError):

print("出错啦!")

finally:

f.close()

### 33.3 raise触发异常

我们可以使用raise语句来自定义引发异常

**语法**

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

**参数**

* Exception是异常的类型（如TypeError），可选
* 最后一个参数是可选的（在实践中很少使用），如果存在，是跟踪异常对象

**代码实例**

>>> raise TypeError

Traceback (most recent call last):

File "<pyshell#0>", line 1, in <module>

raise TypeError

TypeError

>>> raise ZeroDivisionError("除数为0的异常")

Traceback (most recent call last):

File "<pyshell#1>", line 1, in <module>

raise ZeroDivisionError("除数为0的异常")

ZeroDivisionError: 除数为0的异常

**练习题**

1. 我们使用什么方法来处理程序中出现的异常？

答: try-except, try-finally语句

1. 一个try语句可以和多个except语句搭配吗？为什么？

答: 可以

1. 你知道如何统一处理多类异常吗？

答: 将需要处理的异常写成元组的形式，如except (IndexError, TypeError):

1. except后边如果不带任何异常类，Python会捕获所有（try语句块内）的异常并统一处理，但并不推荐这样做，你知道为什么吗？

答: 因为它会隐藏所有程序员未想到且未做好处理的异常错误，例如当用户想输入ctrl+c命令试图终止正在运行的程序，但此时会被解析为KeyboardInterrupt异常。

1. 如果异常发生在成功打开文件后，Python跳到except语句执行，并没有执行关闭文件的命名（用户写入文件的数据可能没有保存起来），因此我们需要确保无论如何（就算出了异常退出）文件也要关闭，我们应该怎么做呢？

答: 使用try-finally语句。语句块如下述代码所示：

finally:

关闭文件的语句

1. 请恢复以下代码中\*\*\*的内容，使得程序执行后可以按要求输出

try:

for i in range(3):

for j in range(3):

\*\*\*

\*\*\*

print(i, j)

except KeyboardInterrupt:

print("退出啦!")

# 运行代码，输出

0 0

0 1

0 2

1 0

1 1

1 2

退出啦！

答: 分析输出结果，可知本应在输出2 0的时候，却触发KeyboardInterrupt异常，所以在代码中，应该做了某种判断，符合某种条件下，触发KeyboardInterrupt异常。

try:

for i in range(3):

for j in range(3):

if i == 2:

raise KeyboardInterrupt

print(i, j)

except KeyboardInterrupt:

print("退出啦!")

**动动手**

1. 尝试一个新的函数int\_input()，当用户输入整数的时候正常返回，否则提示出错并要求重新输入。

答:

def int\_input(input\_str=""):  
 while True:  
 try:  
 int(input(input\_str))  
 break  
 except ValueError:  
 print("类型出错，请重新输入!")  
  
int\_input("请输入整数: ")

1. 把文件关闭放在finally语句块中执行还是会出现问题，像下边这个代码，当前文件夹中并不存在”My\_File.txt”这个文件，那么程序执行起来会发生什么事情呢？你有办法解决这个问题么？

try:

f = open("My\_File.txt") # 当前文件夹下并不存在"My\_File.txt"文件

print(f.read())

except OSError as reason:

print("出错啦: " + str(reason))

finally:

f.close()

**运行上述代码，弹出下述消息：**

Traceback (most recent call last):

File "/home/amusi/Projects/Github/零基础入门学python/课时33 异常处理: 你不可能总是对的2/review002.py", line 15, in <module>

f.close()

**NameError: name 'f' is not defined**

出错啦: [Errno 2] No such file or directory: 'My\_File.txt'

答: 因为当文件打开失败后，f是无效的，所以在finally代码块中试图打开无效的f会失败。

因此我们可以在f.close之前，先判断f是否打开有效

try:  
 f = open("My\_File.txt") # 当前文件夹下并不存在"My\_File.txt"文件  
 print(f.read())  
except OSError as reason:  
 print("出错啦: " + str(reason))  
finally:  
 if f in locals(): # 如果文件对象变量存在当前局部变量符号表的话，说明文件打开成功  
 f.close()

## 课时34 丰富的else语句及简洁的with语句

### 34.1 丰富的else语句

* 要么怎样，要么不怎样

**代码实例**

def choose(flag):

if flag == True:

print("True")

else:

print("False")

choose(True)

* 干完了能怎样，干不完就别想怎样

**代码实例**

def showMaxFactor(num):

count = num // 2

while count > 1:

if num % count == 0:

print("%d最大约数是%d" % (num, count))

break

count -= 1

else:

print("%d是素数!" % num)

num = int(input("请输入一个数: "))

showMaxFactor(num)

* 没有问题，那么就干吧

**代码实例**

try:

int("123")

except ValueError as reason:

print("出错啦: " + str(reason))

else:

print("没有任何异常")

### 34.2 简洁的with语句

[Python的with语句知识点](http://python.jobbole.com/82494/)

**代码实例**

try:

f = open("data.txt", w)

for each\_line in f:

print(each\_line)

except OSError as reason:

print("出错啦: " + str(reason))

finally:

f.close()

上述代码存在bug，因为当”data.txt”文件打开失败后，f是无效的，所以运行f.close()会出错。上一课时的课后练习题中，有通过判断f是否在局部变量表中来确定文件打开状态。此处再介绍一种更为简洁的方法“上下文管理器with语句”。

try:

with open("data.txt", "w") as f:

for each\_line in f:

print(each\_line)

except OSError as reason:

print("出错啦: " + str(reason))

### 测试题

1. 在Python中，else语句能跟哪些语句进行搭配？

答: if-else、while-else、try-except-else等

1. 请问以下例子中，循环中的break语句会跳过else语句吗？

def showMaxFactor(num):

count = num // 2

while count > 1:

if num % count == 0:

print("%d最大约数是%d" % (num, count))

break

count -= 1

else:

print("%d是素数!" % num)

num = int(input("请输入一个数: "))

showMaxFactor(num)

答: 会。在执行break时，代表程序已经进入while中，所以不会再进入else中。

1. 请目测以下代码会打印什么内容？

答:

‘ABC’

‘GHI’

‘JKL’

1. 使用什么语句可以使你不必担心文件打开后却忘了关闭的尴尬？

答: 上下文管理器with语句，如with open(“demo.txt”) as f:

1. 使用with语句固然方便，但如果出现异常的话，文件还会自动正常关闭吗？

答: 会。因为with语句会自动处理文件的打开和关闭，如果中途出现异常，会执行清理代码，然后确保文件正常关闭

1. 换一种形式写出下面的伪代码

with A() as a:

with B() as b:

suite

答: with语句处理多个项目时，可以用逗号隔开写成一条语句

with A() as a, B() as b:

Suite

### 动动手

1. 使用with语句改写以下代码，让Python去关心文件的打开和关闭吧

源代码

def file\_compare(file1, file2):

f1 = open(file1)

f2 = open(file2)

count = 0 #　统计行数

differ = [] # 统计不一样的数量

for line1 in f1:

line2 = f2.readline()

count += 1

if line1 != line2:

differ.append(count)

f1.close()

f2.close()

return differ

file1 = input("请输入需要比较的头一个文件名: ")

file2 = input("请输入需要比较的另一个文件名: ")

differ = file\_compare(file1, file2)

if len(differ) == 0:

print("两个文件完全一样!")

else:

print("两个文件共有[%d]处不同" % len(differ))

for each in differ:

print("第 %d 行不一样" % each)

答:用上下文管理器with语句替换

def file\_compare(file1, file2):

with open(file1) as f1, open(file2) as f2:

count = 0 #　统计行数

differ = [] # 统计不一样的数量

for line1 in f1:

line2 = f2.readline()

count += 1

if line1 != line2:

differ.append(count)

return differ

file1 = input("请输入需要比较的头一个文件名: ")

file2 = input("请输入需要比较的另一个文件名: ")

differ = file\_compare(file1, file2)

if len(differ) == 0:

print("两个文件完全一样!")

else:

print("两个文件共有[%d]处不同" % len(differ))

for each in differ:

print("第 %d 行不一样" % each)

# 章节9 图形用户界面入门

## 课时35 图形用户界面入门: EasyGui

* EasyGui官网

<http://easygui.sourceforge.net/>

* EasyGui GitHub

<https://github.com/robertlugg/easygui>

* EasyGui Tutorial

<http://easygui.sourceforge.net/tutorial.html>

* EasyGui中文学习文档

<http://bbs.fishc.com/forum.php?mod=viewthread&tid=46069&extra=page%3D1%26filter%3Dtypeid%26typeid%3D403>

* Ubuntu下使用EasyGui教程

<http://blog.csdn.net/marksinoberg/article/details/50948900>

**安装方法**

pip install easygui

### 35.1 导入EasyGui

**# 方法一**

>>> import easygui

>>> easygui.msgbox("你好，Amusi")

**# 方法二**

>>> from easygui import \*

>>> msgbox("你好，Amusi")

**# 方式三（推荐）**

>>> import easygui as eg

>>> eg.msgbox("你好，Amusi")

### 35.2 使用EasyGui

下面用EasyGui实现一个简单的可交互界面小程序

**代码实例**

# 导入EasyGui模块

import easygui as eg

import sys

while True:

eg.msgbox("Hi，欢迎进入第一个界面小程序^\_^")

msg = "请问你希望打开哪个功能选项呢？"

title = "EasyGuo小程序"

choices = ["Python入门", "Python进阶", "Python实战", "Python－AI"]

# 获得选择的结果

choice = eg.choicebox(msg, title, choices)

# 弹出对话框，显示选择的结果

eg.msgbox("你的选择是: "+ str(choice), "结果")

msg = "你希望重新开始小程序么？"

title = "请选择"

# 显示一个继续、取消对话框

if eg.ccbox(msg, title):

pass # 用户选择继续

else:

sys.exit(0) # 用户选择取消

注：建议不要在IDLE上运行EasyGui程序

**动动手**

1. 猜数字小游戏，用户输入1~10之间的数字，若与随机生成的数值相等，则退出程序，反之，继续循环

答:

import easygui as eg  
import random  
  
# 初始化  
eg.msgbox("欢迎进入猜数字小游戏")  
secret = random.randint(1,10)  
title = "猜数字小游戏"  
msg = "不妨猜一下数字（1~10）:"  
guess = eg.integerbox(msg, title, lowerbound=1, upperbound=10)  
  
while True:  
 if guess == secret:  
 eg.msgbox("好厉害猜对了！")  
 break  
 else:  
 if guess > secret:  
 eg.msgbox("您输入的数值[偏大]")  
 else:  
 eg.msgbox("您输入的数值[偏小]")  
 guess = eg.integerbox(msg, title, lowerbound=1, upperbound=10)  
  
print("游戏结束！")

# 章节10 类和对象

## 课时36 类和对象: 给大家介绍对象

Python是一门[面向对象](http://www.runoob.com/python/python-object.html)的编程语言。面向对象编程是一种编程方式，此编程方式的落地需要使用“类”和“对象”来实现。所以，面向对象编程其实就是对“类”和“对象”的使用。

类是一个模板，包含了属性和方法；对象是类的实例化，如属性赋予不同的实值。打个比方，类是图纸，而对象就是房屋。



面向对象（oriented object, OO）的特征

* 封装
* 继承
* 多态

**封装代码实例**

新建class\_object.py文件，写入下述代码

# Python中的类名约定以大写字母开头  
class Turtle:  
 # 属性  
 color = "green"  
 weight = 20  
 height = 8  
 mouth = "big"  
  
 # 方法  
 def climb(self):  
 print("我正在努力地向前爬...")  
  
 def run(self):  
 print("我正在努力地向前跑...")  
  
 def fly(self):  
 print("我正在努力地向前飞...")  
  
 def sleep(self):  
 print("我正在睡觉...")

使用IDLE运行class\_object.py文件，可以根据下述代码简单了解类和对象的用法。

>>> tt = Turtle() # 类Turtle的实例化——对象tt

>>> tt.climb() # tt调用方法

我正在努力地向前爬...

>>> tt.run()

我正在努力地向前跑...

>>> tt.sleep()

我正在睡觉...

>>> tt.weight # tt调用属性

20

>>> tt.mouth

'big'

**继承代码实例**

# 若继承某类，只需要在定义类后的括号里写入需要继承的类名

>>> class MyList(list): # 类MyList继承了list类

pass

>>> list2 = MyList() # 类的实例化——对象

>>> list2.append(6) # 此时list2具有list的方法

>>> list2

[6]

>>> list2.append(10)

>>> list2.append(1)

>>> list2.append(199)

>>> list2

[6, 10, 1, 199]

>>> list2.sort()

>>> list2

[1, 6, 10, 199]

**多态代码实例**

# 定义两个类，类里有同名的方法

>>> class A:

def fun(self):

print("A")

>>> class B:

def fun(self):

print("B")

>>> a = A()

>>> b = B()

>>> a.fun()

A

>>> b.fun()

B

### 练习题

1. 对象中的属性和方法，在实际编程中是什么？

答:

* 对象中的属性——变量
* 对象中的方法——函数

1. 类和对象是什么关系？

答: 类是包含属性和方法的封装；对象是类的实例（比如图纸和房子的关系）

类和对象的关系就是抽象和具体的关系。

1. 如果我们定义一个猫类，那么“猫类”可实例化的对象有哪些？

答: 蓝猫、波斯猫、白猫等

1. 类的定义有时候不那么拟物，有时候会抽象一点，例如我们定义一个矩形类，那么可以添加哪些属性和方法呢？

答:

属性：高度、宽度、长宽比

方法：求面积、求周长

1. 类的属性定义应该尽可能的抽象，还是尽可能的具体？

答: 应该尽可能的抽象，这样才符合面向对象的思想

1. 请用一句话描述面向对象的几个特征

答:

* 封装——对外部隐藏对象的工作细节
* 继承——子类自动共享父类的属性和方法
* 多态——可以对不同类调用相同的方法，产生不同的结果

6. 函数和方法有什么区别？

答:

函数和变量是程序层面的术语，而方法和属性是设计层面的术语。在OO中，我们称之为方法和属性，实质上即为函数和变量。

另一种解释，用对象调用的叫方法，直接调用函数名的叫函数。因为方法是类里的函数，所以不能直接显示调用，只能通过对象来间接调用。

### 动动手

1. 按照以下提示，尝试定义一个Person类并生成类实例对象

属性: 姓名（默认姓名为”Amusi”）

方法: 打印姓名

提示: 方法中对属性的引用形式需要加上self，如self.name

答:

# 定义类

>>> class Person:

name = "Amusi"

def print(self):

print(self.name)

# 实例化对象

>>> person = Person()

>>> person.print()

Amusi

1. 按照以下提示，尝试定义一个Rect类并生成类实例对象

属性: 长和宽

方法: 设置长和宽->setRect(self)，获得长和宽->getRect(self)，获得面积->getArea(self)

提示: 方法中对属性的引用形式需加上self，如self.width

答:

# 定义Rect类

>>> class Rect:

length = 0

width = 0

def setRect(self):

print("请输入矩形的长和宽...")

self.length = float(input("长: "))

self.width = float(input("宽: "))

def getRect(self):

print("这个矩形的长是: %f, 宽是: %f" % (self.length, self.width))

def getArea(self):

print("这个矩形的面积是: %f" % (self.length \* self.width))

# 类的实例化对象

>>> rect = Rect()

>>> rect.setRect()

请输入矩形的长和宽...

长: 10

宽: 20

>>> rect.getRect()

这个矩形的长是: 10.000000, 宽是: 20.000000

>>> rect.getArea()

这个矩形的面积是: 200.000000

## 课时37 类和对象: 面向对象编程

当类的实例化对象调用方法时，其实是将对象作为参数传递给self，来调用方法。

### 37.1 self

**代码实例**

>>> class Ball:

def setName(self, name):

self.name = name

def kick(self):

print("我叫%s，有人踢我" % self.name)

# self作用——使得不同对象调用相同的属性和方法是独一无二的

>>> a = Ball()

>>> a.setName("球A")

>>> b = Ball()

>>> b.setName("球B")

>>> c = Ball()

>>> c.setName("球C")

>>> c.kick()

我叫球C，有人踢我

>>> a.kick()

我叫球A，有人踢我

### 37.2 \_\_init\_\_

初始化方法，可初始化属性值，相当于C++的构造函数

**语法**

\_\_init\_\_(self, param1, param2,...,paramN)

**代码实例**

>>> class Ball:

def \_\_init\_\_(self, name):

self.name = name

def kick(self):

print("我叫%s，有人踢我" % self.name)

# 传入参数

>>> b = Ball("Master")

>>> b.kick()

我叫Master，有人踢我

# 没有传入参数

>>> c = Ball()

Traceback (most recent call last):

File "<pyshell#60>", line 1, in <module>

c = Ball()

TypeError: \_\_init\_\_() missing 1 required positional argument: 'name'

### 37.3 公有和私有

* 公有——对象可直接访问
* 私有——对象不可直接访问

注: 由于Python中有name mangling（名字改变、重整）机制，所以在Python中定义私有变量只需要在变量名或函数名前加上”\_\_”两个下划线，那么这个函数或变量就会变成私有的了。

**公有代码实例**

>>> class Person:

name = "Amusi"

>>> p = Person()

>>> p.name

'Amusi'

**私有代码实例**

>>> class Person:

\_\_name = "Amusi"

>>> p = Person()

# 不可直接访问

>>> p.name

Traceback (most recent call last):

File "<pyshell#69>", line 1, in <module>

p.name

AttributeError: 'Person' object has no attribute 'name'

# 不可直接访问

>>> p.\_\_name

Traceback (most recent call last):

File "<pyshell#70>", line 1, in <module>

p.\_\_name

AttributeError: 'Person' object has no attribute '\_\_name'

# 但还是因为name mangling机制，所以根据这种方法还是可以访问”私有”成员

>>> p.\_Person\_\_name

'Amusi'

### 练习题

1. 以下代码体现了面向对象的什么特征？

>>> "Amusi".count("m")

1

>>> [1, 1, 2, 3, 5, 8].count(1)

2

>>> (0, 2, 4, 8, 12, 18).count(1)

0

答: 多态

>>> "Amusi".count("m") # 对字符串调用count方法

>>> [1, 1, 2, 3, 5, 8].count(1) # 对列表调用count方法

>>> (0, 2, 4, 8, 12, 18).count(1) # 对元组调用count方法

1. 当程序员不想把代码写几次，他们发现函数解决了这个问题。当程序有一个类时，又想建立新的非常相近的类，我们应该怎么做呢？

答: 继承

1. self参数的作用?

答: 区分对象（相当于C++中的this指针），你可以认为self是实例对象的唯一标志

1. 如果我们不希望对象的属性或方法被外部直接引用，我们应该怎么做？

答: 在需要处理的属性或方法前加上”\_\_”双下划线，即可屏蔽

1. 类在实例化后，哪个方法会被自动调用？

答: \_\_init\_\_方法会在类实例化时自动调用，我们成为魔法方法（在C++中，我们可以认为是构造函数）。我们可以重写这个方法，对对象定制初始化方案。

1. 请解释下面代码错误的原因？

>>> class MyClass:

name = "Amusi"

def myFun(self):

print("Hello FishC!")

>>> MyClass.name

'Amusi'

>>> MyClass.myFun()

Traceback (most recent call last):

File "<pyshell#9>", line 1, in <module>

MyClass.myFun()

答:

首先要明确类、类对象、实例对象这三个不同的名词。

类指的是类定义，如MyClass

类对象是可以直接调用属性，如MyClass.name

实例对象是类的不同实例化对象，如a = MyClass()、b = MyClass()，类为了区分不同的实例化对象，使用self来限定来区分。

这里，MyClass.myFun()出错因为方法有参数self，这表示使用该方法是需要实例化的对象来调用。

### 动动手

1. 请按照以下要求定义一个游乐园门票的类，并尝试计算2个成人和1个小孩的平日票价

* 平日票价100元
* 周末票价为平时的120%
* 儿童半票

答: 面向对象的难点在于思维的转换

>>> class Ticket():

def \_\_init\_\_(self, weekend=False, child=False):

self.exp = 100

if weekend:

self.inc = 1.2

else:

self.inc = 1

if child:

self.discount = 0.5

else:

self.discount = 1

def calcPrice(self, num):

return self.exp \* self.inc \* self.discount \* num

>>> adult = Ticket()

>>> child = Ticket(child = True)

>>> print("2个成人 + 1个小孩平日票价为: %.2f"% (adult.calcPrice(2) + child.calcPrice(1)))

2个成人 + 1个小孩平日票价为: 250.00

1. 游戏编程: 按以下要求定义一个乌龟类和鱼类并尝试编写游戏

* 假设游戏场景为范围(x,y)，其中0<=x<=10，0<=y<=10
* 游戏生成1只乌龟和10条鱼
* 它们的移动方向均随机
* 乌龟的最大移动能力是2（它可以随机选择1还是2移动），鱼儿的最大移动能力是1
* 当移动到场景边缘，自动向反方向移动
* 乌龟初始化体力为100（上限）
* 乌龟每移动一次，体力消耗1
* 当乌龟和鱼坐标重叠，乌龟吃掉鱼，乌龟体力增加20
* 鱼暂不计算体力
* 当乌龟体力值为0（挂掉）或者鱼儿的数量为0时，游戏结束

答:

import random as r  
legal\_x = [0, 10]  
legal\_y = [0, 10]  
  
class Turtle:  
 def \_\_init\_\_(self):  
 # 初始体力  
 self.power = 100  
  
 # 初始位置随机  
 self.x = r.randint(legal\_x[0], legal\_y[1])  
 self.y = r.randint(legal\_x[0], legal\_x[1])  
  
 def move(self):  
 # 随机计算方向并移动到新的位置(x,y)  
 new\_x = self.x + r.choice([1, 2, -1, -2])  
 new\_y = self.y + r.choice([1, 2, -1, -2])  
 # 检查移动后是否超过场景ｘ轴边界  
 if new\_x < legal\_x[0]:  
 self.x = legal\_x[0] - (new\_x - legal\_x[0])  
 elif new\_x > legal\_x[1]:  
 self.x = legal\_x[1] - (new\_x - legal\_x[1])  
 else:  
 self.x = new\_x  
 # 检查移动后是否超过场景y轴边界  
 if new\_y < legal\_y[0]:  
 self.y = legal\_y[0] - (new\_y - legal\_y[0])  
 elif new\_y > legal\_y[1]:  
 self.y = legal\_y[1] - (new\_y - legal\_y[1])  
 else:  
 self.y = new\_y  
  
 # 体力消耗  
 self.power -= 1  
 # 返回移动后的新位置  
 return (self.x, self.y)  
  
 def eat(self):  
 self.power += 20  
 if self.power > 100:  
 self.power = 100  
  
class Fish:  
 def \_\_init\_\_(self):  
  
 # 初始位置随机  
 self.x = r.randint(legal\_x[0], legal\_y[1])  
 self.y = r.randint(legal\_x[0], legal\_x[1])  
  
 def move(self):  
 # 随机计算方向并移动到新的位置(x,y)  
 new\_x = self.x + r.choice([1, -1])  
 new\_y = self.y + r.choice([1, -1])  
 # 检查移动后是否超过场景ｘ轴边界  
 if new\_x < legal\_x[0]:  
 self.x = legal\_x[0] - (new\_x - legal\_x[0])  
 elif new\_x > legal\_x[1]:  
 self.x = legal\_x[1] - (new\_x - legal\_x[1])  
 else:  
 self.x = new\_x  
 # 检查移动后是否超过场景y轴边界  
 if new\_y < legal\_y[0]:  
 self.y = legal\_y[0] - (new\_y - legal\_y[0])  
 elif new\_y > legal\_y[1]:  
 self.y = legal\_y[1] - (new\_y - legal\_y[1])  
 else:  
 self.y = new\_y  
  
 # 返回移动后的新位置  
 return (self.x, self.y)  
  
# 1个乌龟  
turtle = Turtle()  
fish = []  
# 10个鱼  
for i in range(10):  
 new\_fish = Fish()  
 fish.append(new\_fish)  
  
while True:  
 if not len(fish):  
 print("鱼儿都被吃完了，游戏结束!")  
 break  
  
 if not turtle.power:  
 print("乌龟体力耗尽，挂掉了!")  
 break  
  
 pos = turtle.move()  
  
 # 在迭代器中删除列表元素是非常危险的，经常会出现意向不到的问题，因为迭代器是直接引用列表的数据进行引用  
 # 这里我们把列表拷贝给迭代器，然后对原列表进行删除操作就不会有问题了  
 for each\_fish in fish[:]:  
 if each\_fish.move() == pos:  
 # 鱼儿被吃掉了  
 turtle.eat()  
 fish.remove(each\_fish)  
 print("有一条鱼被吃掉了...")

### 课时38 类和对象: 继承

面向对象的编程带来的主要好处之一是代码的重用，实现这种重用的方法之一是通过继承机制。继承完全可以理解成类之间的类型和子类型关系。

继承的关系可以理解为基类和派生类，或者父类和子类。

需要注意的地方：继承语法 class 派生类名（基类名）：//... 基类名写在括号里，基本类是在类定义的时候，在元组之中指明的。

### 38.1 继承知识点

**语法**

派生类的声明，与它们的基类类似。但需要添加括号，且括号里写入需要继承的基类列表，如下所示：

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

'Optional class documentation string'

class\_suite

**代码实例**

>>> class Parent:

def hello(self):

print("父类")

>>> class Child(Parent):

pass

>>> p = Parent()

>>> p.hello()

父类

>>> c = Child()

# 继承调用父类的方法

>>> c.hello()

父类

注: 如果子类中定义与父类中同名的方法或属性，则会自动覆盖父类对应的方法或属性

>>> class Child(Parent):

def hello(self):

print("子类")

>>> c = Child()

>>> c.hello()

子类

>>> p.hello()

父类

**代码实例**

import random as r  
  
class Fish:  
 def \_\_init\_\_(self):  
 self.x = r.randint(0, 10)  
 self.y = r.randint(0, 10)  
  
 def move(self):  
 self.x -= 1  
 print("我的位置: (%d, %d)" % (self.x, self.y))  
  
class Goldfish(Fish):  
 pass  
  
class Carp(Fish):  
 pass  
  
class Salmon(Fish):  
 pass  
  
class Shark(Fish):  
 def \_\_init\_\_(self):  
 self.hungry = True  
  
 def eat(self):  
 if self.hungry == True:  
 print("有吃的啦！")  
 self.hungry = False  
 else:  
 print("太撑了，吃不下了")

**运行上述代码**

>>> fish = Fish()

>>> fish.move()

我的位置: (-1, 1)

>>> goldfish = Goldfish()

>>> goldfish.move()

我的位置: (1, 8)

>>> goldfish.move()

我的位置: (0, 8)

>>> shark = Shark()

>>> shark.eat()

有吃的啦！

>>> shark.eat()

太撑了，吃不下了

# 调用父类的方法

>>> shark.move()

Traceback (most recent call last):

File "<pyshell#9>", line 1, in <module>

shark.move()

File "/home/amusi/Projects/Github/零基础入门学python/课时038 类和对象: 继承/fish.py", line 12, in move

self.x -= 1

AttributeError: 'Shark' object has no attribute 'x'

注: 由于在Shark类中重写了\_\_init\_\_方法，所以在Shark中没有x、y属性，所以再去调用父类的属性时，程序会出错。解决方法有两种，1调用未绑定的父方法，2supper函数

### 38.2　调用未绑定的父方法

import random as r  
  
class Fish:  
 def \_\_init\_\_(self):  
 self.x = r.randint(0, 10)  
 self.y = r.randint(0, 10)  
  
 def move(self):  
 self.x -= 1  
 print("我的位置: (%d, %d)" % (self.x, self.y))  
  
class Shark(Fish):  
 def \_\_init\_\_(self):  
 Fish.\_\_init\_\_(self)  
 self.hungry = True  
  
 def eat(self):  
 if self.hungry == True:  
 print("有吃的啦！")  
 self.hungry = False  
 else:  
 print("太撑了，吃不下了")

**运行代码**

>>> shark = Shark()

>>> shark.eat()

有吃的啦！

>>> shark.move()

我的位置: (2, 4)

>>> shark.move()

我的位置: (1, 4)

### 38.3 使用super函数

**语法**

supper().父类的方法

import random as r  
  
class Fish:  
 def \_\_init\_\_(self):  
 self.x = r.randint(0, 10)  
 self.y = r.randint(0, 10)  
  
 def move(self):  
 self.x -= 1  
 print("我的位置: (%d, %d)" % (self.x, self.y))  
  
class Goldfish(Fish):  
 pass  
  
class Carp(Fish):  
 pass  
  
class Salmon(Fish):  
 pass  
  
class Shark(Fish):  
 def \_\_init\_\_(self):  
 # Fish.\_\_init\_\_(self)  
 super().\_\_init\_\_()  
 self.hungry = True  
  
 def eat(self):  
 if self.hungry == True:  
 print("有吃的啦！")  
 self.hungry = False  
 else:  
 print("太撑了，吃不下了")

**运行代码**

>>> shark = Shark()

>>> shark.eat()

有吃的啦！

>>> shark.move()

我的位置: (2, 4)

>>> shark.move()

我的位置: (1, 4)

### 38.4 多重继承

**语法**

Class DerivedClassName(Base1, Base2, Base3,..., BaseN):

...

**代码实例**

>>> class Base1:

def run1(self):

print("跑得第一名!")

>>> class Base2:

def run2(self):

print("跑得第二名!")

# 多重继承

>>> class C(Base1, Base2):

pass

>>> c = C()

>>> c.run1()

跑得第一名!

>>> c.run2()

跑得第二名!

>>> c.run1()

跑得第一名!

### 练习题

1. 继承机制给程序员带来最明显的好处是？

答: 不用重复写相同的属性或方法

1. 如果按以下方式重写\_\_init\_\_，结果会怎么样？

>>> class MyClass:

def \_\_init\_\_(self):

return "Amusi love Lucy!"

答: 程序出现错误，异常退出，具体如下，可知\_\_init\_\_方法只能返回None

>>> myclass = MyClass()

Traceback (most recent call last):

File "<pyshell#23>", line 1, in <module>

myclass = MyClass()

TypeError: \_\_init\_\_() should return None, not 'str'

1. 当子类定义了相同的属性和方法，Python是否会自动删除父类的同名属性和方法？

答: 不会删除，但会覆盖，即子类调用同名的属性或函数时，是调用子类自身。此时，父类的同名属性或函数还在，但子类“看不见”。

#　父类

>>> class Parent:

name = "Parent"

def run(self):

print("Parent")

# 子类

>>> class Child:

name = "Child"

def run(self):

print("Child")

>>> c = Child()

>>> c.name

'Child'

>>> c.run()

Child

1. 假设已经有鸟类的定义，现在我要定义企鹅类继承鸟类，但我们都知道企鹅不会飞，我们应该如何屏蔽父类（鸟类）中飞的方法？

答: 重写同名函数，但结构体中不处理，即pass

>>> class Bird:

def fly(self):

print("I can fly!")

>>> class Penguin:

def fly(self):

pass

>>> p = Penguin()

>>> p.fly()

# 无返回值

>>>

1. super函数有什么”超级”的地方？

答: super函数超级之处在于你不需要明确给出基类的名称，它会自动帮你找到所有基类以及对应的方法。由于不需要给出基类的名称，这意味着如果你需要改变继承关系，只要改变class()中的基类名称即可。而不必大量去修改所有被继承的方法。

1. 多重继承使用不当会导致重复调用（也称钻石继承、菱形继承）的问题，请分析以下代码在实际编程中有可能导致什么问题？

class A():

def \_\_init\_\_(self):

print("进入A...")

print("离开A...")

class B(A):

def \_\_init\_\_(self):

print("进入B...")

A.\_\_init\_\_(self)

print("离开B...")

class C(A):

def \_\_init\_\_(self):

print("进入C...")

A.\_\_init\_\_(self)

print("离开C...")

class D(B, C):

def \_\_init\_\_(self):

print("进入D...")

B.\_\_init\_\_(self)

C.\_\_init\_\_(self)

print("离开D...")

# 运行后

>>> d = D()

进入D...

进入B...

进入A...

离开A...

进入A...

离开A...

离开B...

离开D...

答: 多重继容易导致重复调用问题。对于D类，虽然继承了不同的B和C类，但实质上重复继承了A。如Ｄ实例化对象d，会重叠调用A的\_\_init\_\_方法。所以编程中，一定要尽量避免多重继承。

1. 如何解决上一题中出现的问题？

答: 使用super函数。

**Review001.py**

class A():

def \_\_init\_\_(self):

print("进入A...")

print("离开A...")

class B(A):

def \_\_init\_\_(self):

print("进入B...")

A.\_\_init\_\_(self)

print("离开B...")

class C(A):

def \_\_init\_\_(self):

print("进入C...")

A.\_\_init\_\_(self)

print("离开C...")

class D(B, C):

def \_\_init\_\_(self):

print("进入D...")

# B.\_\_init\_\_(self)

# C.\_\_init\_\_(self)

# 使用super()函数解决多重继承的问题

super().\_\_init\_\_()

print("离开D...")

**# 运行上述代码**

>>> d = D()

进入D...

进入B...

进入A...

离开A...

离开B...

离开D...

### 动动手

1. 定义一个点（Point）类和直线（Line）类，使用getLen方法可以获得直线的长度

提示:

* 设点A(X1,Y1)、点B(X2,Y2)，则两点构成的直线长度|AB|=sqrt((X1-X2)\*(X1-X2)+(Y1-Y2)\*(Y1-Y2)
* Python中计算开根号可使用math模块中的sqrt函数
* 直线需要两点构成，因此初始化时需要两个点(Point)对象作为参数

答:

**Review002.py**

import math

class Point:

def \_\_init\_\_(self, x=0, y=0):

self.x = x

self.y = y

def getX(self):

return self.x

def getY(self):

return self.y

class Line(Point):

def \_\_init\_\_(self, p1, p2):

self.x = p1.x - p2.x

self.y = p1.y - p2.y

self.len = 0

# 计算直线长度

def getLen(self):

self.len = math.sqrt(self.x \* self.x + self.y \* self.y)

return self.len

**# 运行上述代码**

>>> p1 = Point(10, 15)

>>> p2 = Point(6, 20)

>>> l = Line(p1, p2)

>>> l.getLen()

6.4031242374328485

1. 已经学习了这么多的Python知识，请尝试用所学到的知识写一个Python软件作品（可以是游戏、应用软件或脚本）

答:

未完成

### 课时39 类和对象: 拾遗

### 39.1 组合

组合是一个类有中有其它类的属性和方法。（继承是父类和子类的关系）

**代码实例**

class Turtle:

def \_\_init\_\_(self, x):

self.num = x

class Fish:

def \_\_init\_\_(self, x):

self.num = x

class Pool:

def \_\_init\_\_(self, x, y):

# 组合

self.turtle = Turtle(x)

self.fish = Fish(y)

def print\_num(self):

print("水池里总共有乌龟 %d 只, 小鱼 %d 条!" % (self.turtle.num, self.fish.num))

# 运行上述代码

>>> pool = Pool(1, 10)

>>> pool.print\_num()

水池里总共有乌龟 1 只, 小鱼 10 条!

### 39.2 类、类对象和实例化对象

* 类是一种抽象化的描述，可用class表示。
* 类对象是类class的定义，如class People: \*\*\*
* 实例化对象是类的实例，如people = People()
* **代码实例——**实例化对象属性会将类对象属性覆盖

# 类class和类对象C

>>> class C:

count = 0

# 实例化对象a, b, c

>>> a = C()

>>> b = C()

>>> c = C()

>>> a.count

0

>>> b.count

0

>>> c.count

0

# 只改变c的属性值，并不会对a、b产生影响

>>> c.count += 10

>>> c.count

10

>>> a.count

0

>>> b.count

0

# 类对象属性值

>>> C.count

0

# 修改类对象属性值，发现对a、b产生影响，而c没有变化

>>> C.count += 100

>>> a.count

100

>>> b.count

100

>>> c.count

10

**C**

**C**

**a**

**b**

**c**

**类定义**

**类对象**

**实例化对象**

图39.1 类、类对象和实例化对象

* **代码实例——**实例化对象新创建的属性会覆盖同名方法

>>> class C:

def x(self):

print("X-man!")

>>> c = C()

>>> c.x()

X-man!

# 实例化对象新创建属性x（与已有的方法同名）

>>> c.x = 1

>>> c.x

1

>>> c.x()

Traceback (most recent call last):

File "<pyshell#27>", line 1, in <module>

c.x()

TypeError: 'int' object is not callable

### 39.3 Python绑定

Python严格要求方法需要有实例才能被绑定，这种限制其实就是Python所谓的绑定概念. 这里绑定可以加入self来约束。

无绑定（无self）—>类对象可调用

有绑定（有self）—>实例化对象可调用

注:

一般情况下，推荐使用self绑定.

**代码实例——未绑定**

>>> class BB:

def printBB():

print("Fighting!")

# 类对象调用未绑定方法——可行

>>> BB.printBB()

Fighting!

>>> bb = BB()

# 实例化对象调用未绑定方法——不可行

>>> bb.printBB()

Traceback (most recent call last):

File "<pyshell#34>", line 1, in <module>

bb.printBB()

TypeError: printBB() takes 0 positional arguments but 1 was given

**代码实例——有绑定**

>>> class BB:

def printBB(self):

print("Fighting!")

# 类对象调用有绑定——不可行

>>> BB.printBB()

Traceback (most recent call last):

File "<pyshell#37>", line 1, in <module>

BB.printBB()

TypeError: printBB() missing 1 required positional argument: 'self'

# 实例化对象调用有绑定——可行

>>> bb = BB()

>>> bb.printBB()

Fighting!

**代码实例**

>>> class CC:

def setXY(self, x, y):

self.x = x

self.y = y

def printXY(self):

print(self.x, self.y)

# 实例化对象dd

>>> dd = CC()

# 查看dd属性（暂无）

>>> dd.\_\_dict\_\_

{}

# 查看CC属性

>>> CC.\_\_dict\_\_

mappingproxy({'\_\_module\_\_': '\_\_main\_\_', 'setXY': <function CC.setXY at 0x7f49aa730268>, 'printXY': <function CC.printXY at 0x7f49aa7302f0>, '\_\_dict\_\_': <attribute '\_\_dict\_\_' of 'CC' objects>, '\_\_weakref\_\_': <attribute '\_\_weakref\_\_' of 'CC' objects>, '\_\_doc\_\_': None})

>>> dd.setXY(8, 9)

# 再次查看dd属性，此时多了x、y和对应值

>>> dd.\_\_dict\_\_

{'x': 8, 'y': 9}

# CC属性不变

>>> CC.\_\_dict\_\_

mappingproxy({'\_\_module\_\_': '\_\_main\_\_', 'setXY': <function CC.setXY at 0x7f49aa730268>, 'printXY': <function CC.printXY at 0x7f49aa7302f0>, '\_\_dict\_\_': <attribute '\_\_dict\_\_' of 'CC' objects>, '\_\_weakref\_\_': <attribute '\_\_weakref\_\_' of 'CC' objects>, '\_\_doc\_\_': None})

# 删除CC类对象，不能再用CC实例化对象了

>>> del CC

>>> dc = CC()

Traceback (most recent call last):

File "<pyshell#55>", line 1, in <module>

dc = CC()

NameError: name 'CC' is not defined

# 但dd方法仍然可用（因为类的属性和方法是静态的，生命周期是整个程序）

>>> dd.printXY()

8 9

### 课时40 类和对象: 一些相关的BIF

### 40.1 issubclass()

[issubclass()](http://www.runoob.com/python/python-func-issubclass.html)方法用于判断函数class是否为类型参数classinfo的子类

**语法**

issubclass(class, classinfo)

**参数**

* + class——类
  + classinfo——类

**返回值**

如果class是classinfo的子类则返回True，否则返回False

**代码实例**

>>> class A:

pass

# B继承A

>>> class B(A):

pass

>>> issubclass(B, A)

True

# 类是自身的子类

>>> issubclass(B, B)

True

>>> issubclass(A, B)

False

# 任何类都默认继承object类

>>> issubclass(B, object)

True

>>> issubclass(A, object)

True

### 40.2 isinstance()

[isinstance()方法](http://www.runoob.com/python/python-func-isinstance.html)来判断一个对象是否是一个已知的类型，类似于type()

isinstance()与type()区别

* + type()不会认为一个子类是一种父类类型，不考虑继承关系
  + ininstance()会认为子类是一种父类类型，考虑继承关系

**语法**

isinstance(object, classinfo)

**参数**

* object——实例化对象
* classinfo——可以是直接或间接类名、基本类型或者它们组成的元组

**返回值**

如果对象的类型１（object）与参数2的类型（classinfo）相同，则返回True，否则返回False。

注：如果第１个参数不是对象，则永远返回False；如果第2个参数不是由类或类对象或组成的元组，则会抛出一个TypeError异常

**代码实例**

>>> class A:

pass

# B继承A

>>> class B(A):

pass

>>> b1 = B()

>>> isinstance(b1, B)

True

# isinstance()对父类有效

>>> isinstance(b1, A)

True

>>> isinstance(b1, (A, B))

True

>>> type(b1) == B

True

# type()对父类无效

>>> type(b1) == A

False

### 40.3 hasattr()

[hasattr()方法](http://www.runoob.com/python/python-func-hasattr.html)用于判断对象是否包含对应的属性

**语法**

hasattr(object, name)

**参数**

* + object——对象
  + name——字符串类型的属性名

**返回值**

如果对象有该属性则返回True，否则返回False

**代码实例**

>>> class C:

def \_\_init\_\_(self, x=0):

self.x = x

>>> c1 = C()

# 类型名一定要是字符串

>>> hasattr(c1, x)

Traceback (most recent call last):

File "<pyshell#29>", line 1, in <module>

hasattr(c1, x)

NameError: name 'x' is not defined

>>> hasattr(c1, "x")

True

### 40.4 getattr()

[getattr()方法](http://www.runoob.com/python/python-func-getattr.html)用于返回一个对象属性值（实值）

**语法**

getattr(object, name[, default])

**参数**

* object——对象
* name——字符串，对象属性
* default——默认返回值，如果不提供参数，在没有对应属性时，将触发AttributeError

**返回值**

返回对象属性值

**代码实例**

>>> class A:

bar = 1

>>> a = A()

# 获得存在的属性值bar

>>> getattr(a, "bar")

1

# 获得不存在的属性值bar2

>>> getattr(a, "bar2")

Traceback (most recent call last):

File "<pyshell#36>", line 1, in <module>

getattr(a, "bar2")

AttributeError: 'A' object has no attribute 'bar2'

# 属性bar2不存在，但设置了默认值3

>>> getattr(a, "bar2", 3)

3

### 40.5 setattr()

[setattr()方法](http://www.runoob.com/python/python-func-getattr.html)对应于方法getattr，用于设置属性值，该属性必须存在

**语法**

setattr(object, name, value)

**参数**

* + object——对象
  + name——字符串，对象属性
  + value——属性值

**返回值**

无

**代码实例**

>>> class A:

bar = 1

>>> a = A()

# 获得属性值

>>> getattr(a, "bar")

1

# 给存在的属性赋值

>>> setattr(a, "bar", 9)

>>> a.bar

9

# 给不存在的属性赋值！！！居然成功了

>>> setattr(a, "bar2", 8)

>>> a.bar2

8

### 40.6 delattr()

[delattr()方法](http://www.runoob.com/python/python-func-delattr.html)用于删除属性

**语法**

delattr(classname, name)

**参数**

* + classname——类名
  + name——字符串，必须是对象的属性

**返回值**

无

**代码实例**

>>> class Point:

def \_\_init\_\_(self, x=8, y=88):

self.x = x

self.y = y

>>> p = Point()

>>> p.x

8

>>> p.y

88

# 删除属性x

>>> delattr(p, 'x')

>>> p.x

Traceback (most recent call last):

File "<pyshell#69>", line 1, in <module>

p.x

AttributeError: 'Point' object has no attribute 'x'

>>> p.y

88

>>> delattr(p, "y")

# 重复删除属性y

>>> delattr(p, "y")

Traceback (most recent call last):

File "<pyshell#72>", line 1, in <module>

delattr(p, "y")

AttributeError: y

# 删除不存在的属性z

>>> delattr(p, "z")

Traceback (most recent call last):

File "<pyshell#73>", line 1, in <module>

delattr(p, "z")

AttributeError: z

### 40.7 property()

[property()方法](http://www.runoob.com/python/python-func-property.html)的作用是在新式类中返回属性值

简单地说，设置定义好的属性

**语法**

property([fget[, fset[, fdel[, doc]]])

**参数**

* + fget——获得属性值的函数
  + name——字符串，必须是对象的属性
  + fdel——删除属性值函数
  + doc——属性描述信息

**返回值**

返回新式类属性

**代码实例**

>>> class C:

def \_\_init\_\_(self, size=10):

self.size = size

# 获得属性

def getSize(self):

return self.size

# 设置属性——建立关联

def setSize(self, value):

self.size = value

# 删除属性

def delSize(self):

del self.size

x = property(getSize, setSize, delSize)

>>> c1 = C()

>>> c1.getSize()

10

# property作用

>>> c1.x

10

>>> c1.x = 18

>>> c1.size

18

>>> c1.getSize()

18

>>> del c1.x

>>> c1.size

Traceback (most recent call last):

File "<pyshell#97>", line 1, in <module>

c1.size

AttributeError: 'C' object has no attribute 'size'

练习题

动动手

# 章节11 魔法方法

## 课时41 魔法方法: 构造和析构

魔法方法总是被双下划线包围，例如\_\_init\_\_

魔法方法是面向对象的Python的一切

### 41.1 \_\_init\_\_()

**实例化对象的初始化，\_\_init\_\_()是一个实例化方法，第一个参数是self**。当类被调用时，实例化的第一步是创建实例对象。一旦对象创建了，Python检查是否实习了\_\_init\_\_()方法。程序员可以自己定义或者不定义\_\_init\_\_()方法（其实是重载），一旦定义了，会覆盖它的默认行为。简单来说，把创建实例的调用当成是对构造器的调用。

**语法**

\_\_init(self[, ...])

**参数**

self: self

[...]: 传入的参数名

**返回值**

None

注: 重载\_\_init\_\_()方法时，不需要返回值，否则会出现“TypeError”异常出错

**代码实例**

>>> class Rectangle:

# 重载\_\_init\_\_()方法，设定有默认参数width和height

def \_\_init\_\_(self, width=0, height=0):

self.width = width

self.height = height

def getPeri(self):

return (self.width + self.height)\*2

def getArea(self):

return self.width \* self.height

>>> rect = Rectangle(3, 4)

>>> rect.getPeri()

14

>>> rect.getArea()

12

### 41.2 \_\_new\_\_()

对象的创建，\_\_new\_\_()试是一个静态方法，第一个参数cls。（此时对象还没有创建，因此self还不存在，故先创建再初始化，即\_\_new\_\_()方法要早于\_\_init\_\_()被调用）

注: 一般不会重写/重载\_\_new\_\_()方法

**语法**

\_\_new\_\_(cls[, ...])

**参数**

cls——类

**返回值**

cls的实例化对象object

代码实例

# CapStr继承str类

>>> class CapStr(str):

# 重载\_\_new\_\_()方法

def \_\_new\_\_(cls, string):

string = string.upper()

return str.\_\_new\_\_(cls, string)

>>> a = CapStr("Amusi")

>>> a

'AMUSI'

### 41.3 \_\_del\_\_()

销毁/删除实例化对象

**语法**

\_\_del\_\_(self)

**参数**

self

**返回值**

**注:**

del x != x.\_\_del\_\_()

del x不会主动调用\_\_del\_\_方法，只有引用计数==0时，\_\_del\_\_()才会被执行，并且定义了\_\_del\_\_()的实例无法被Python的循环垃圾收集器手机，所以尽量不要自定义\_\_del\_\_()。一般情况下，\_\_del\_\_()不会破坏垃圾处理器。

**代码实例**

>>> class C:

# 重写\_\_init\_\_()初始化方法

def \_\_init\_\_(self):

print("我是\_\_init\_\_()方法")

# 重写\_\_del\_\_()析构方法

def \_\_del\_\_(self):

print("我是\_\_del\_\_()方法")

>>> c1 = C()

我是\_\_init\_\_()方法

>>> c2 = c1

>>> c3 = c2

>>> del c3

>>> del c2

>>> del c1

我是\_\_del\_\_()方法

**小节**

\_\_init\_\_()和\_\_new\_\_()方法可以看作是Python的构造器

\_\_del\_\_()可以看作是Python的析构器

练习题

动动手

## 课时42 魔法方法: 算数运算

### 42.1 工厂函数

**工厂函数**——内建函数(BIF)为类对象的函数.当调用它们时，实际上创建了一个类实例

表42.1 常见的工厂函数

|  |  |
| --- | --- |
| **工厂函数** | **属性** |
| int() | 生成整型 |
| float() | 生成浮点型 |
| bool() | 生成布尔型 |
| str() | 生成字符串 |
| list() | 生成列表 |
| tuple() | 生成元组 |
| dict() | 生成字典 |
| set() | 生成可变集合 |
| frozenset() | 生成不可变集合 |
| type() | 查看类型 |
| complex() | 创建复数 |
| unicode() | 创建unicode字符串 |
| basestring() | 判断一个对象是否为str或者unicode的实例 |
| object() | 空类 |
| classmethod() | 声明一个类方法 |
| staticmethod() | 声明一个静态方法 |
| super() | 指定类的父类 |
| property() | 在新式类中返回属性值 |

表42.2 算数运算

|  |  |
| --- | --- |
| \_\_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) | 定义整数乘法的行为 // |
| \_\_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) | 定位当被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 | |

**代码实例**

>>> type(len)

<class 'builtin\_function\_or\_method'>

>>> type(dir)

<class 'builtin\_function\_or\_method'>

>>> type(int)

<class 'type'>

>>> type(list)

<class 'type'>

>>> class C:

pass

>>> type(C)

<class 'type'>

>>> a = int("123")

>>> b = int("456")

>>> a + b

579

# 修改类的魔法方法

>>> class New\_int(int):

def \_\_add\_\_(self, other):

return int.\_\_sub\_\_(self, other)

def \_\_sub\_\_(self, other):

return int.\_\_add\_\_(self, other)

>>> a = New\_int(3)

>>> b = New\_int(9)

>>> a + b

-6

>>> a - b

12

>>> 10 + 5

15

# 错误使用代码——递归问题

>>> class Try\_int(int):

def \_\_add\_\_(self, other):

return self + other

def \_\_sub\_\_(self, other):

return self - other

>>> a = Try\_int(8)

>>> b = Try\_int(9)

>>> a + b

Traceback (most recent call last):

File "<pyshell#33>", line 1, in <module>

a + b

File "<pyshell#30>", line 3, in \_\_add\_\_

return self + other

File "<pyshell#30>", line 3, in \_\_add\_\_

return self + other

File "<pyshell#30>", line 3, in \_\_add\_\_

return self + other

[Previous line repeated 327 more times]

RecursionError: maximum recursion depth exceeded while calling a Python object

>>> a - b

Traceback (most recent call last):

File "<pyshell#34>", line 1, in <module>

a - b

File "<pyshell#30>", line 5, in \_\_sub\_\_

return self - other

File "<pyshell#30>", line 5, in \_\_sub\_\_

return self - other

File "<pyshell#30>", line 5, in \_\_sub\_\_

return self - other

[Previous line repeated 327 more times]

RecursionError: maximum recursion depth exceeded while calling a Python object

# 解决问题

>>> class Try\_int(int):

def \_\_add\_\_(self, other):

return int(self) + int(other)

def \_\_sub\_\_(self, other):

return int(self) - int(other)

>>> a = Try\_int(8)

>>> b = Try\_int(9)

>>> a + b

17

>>>

### 42.2 新式类和经典类（旧式类）的区别

* 新式类都是从object继承，经典类不需要
* 新式类的基类搜索顺序（Method Resolution Order，MRO）算法采用C3算法广度优先搜索，而旧式类MRO算法是采用深度优先搜索
* 新式类相同父类只执行一次构造函数，经典类重复执行多次
* Python2.x默认都是经典类，只有显式继承了object才是新式类
* Python3.x默认都是新式类，经典类则已被移除
* 新式类更符合OOP思想，统一了Python中的类型机制

**代码实例**

class A():

def \_\_init\_\_(self):

pass

def save(self):

print("This is from A")

class B(A):

def \_\_init\_\_(self):

pass

class C(A):

def \_\_init\_\_(self):

pass

def save(self):

print("This is from C")

class D(B,C):

def \_\_init\_\_(self):

pass

# 运行上述代码

fun = D()

fun.save()

经典类的答案： This is from A

新式类的答案： This is from C

练习题

动动手

### 课时43 魔法方法: 算法运算2

[魔法方法详解](http://bbs.fishc.com/thread-48793-1-1.html)

**代码实例——重载int的add方法**

>>> class int(int):

# 重写

def \_\_add\_\_(self, other):

return int.\_\_sub\_\_(self, other)

>>> a = int("5")

>>> a

5

>>> b = int("3")

>>> a + b

2

**代码实例——注意重写方法参数的顺序**

>>> class Nint(int):

def \_\_rsub\_\_(self, other):

return int.\_\_sub\_\_(other, self)

>>> a = Nint(9)

>>> 1 - a

-8

>>> class Nint(int):

def \_\_rsub\_\_(self, other):

return int.\_\_sub\_\_(self, other)

>>> a = Nint(9)

>>> 1 - a

8

## 课时44 魔法方法: 简单定制

**基本要求**

* 定制一个计时器的类
* start和stop方法代表启动计时和停止计时
* 假设计时器对象t1，print(t1)和直接调用t1均显示结果
* 当计时器未启动或已经停止计时，调用stop方法会给予温馨的提示
* 两个计时器对象可以进行相加：t1+t2
* 只能使用提供的有限资源完成

**资源**

* 使用time模块的localtime方法获取时间
  + 扩展阅读：[time模块详解](http://bbs.fishc.com/thread-51326-1-1.html)（时间获取和转换）
* time.localtime返回struct\_time的时间格式
* 表现你的类：\_\_str\_\_和\_\_repr\_\_

**代码实例——\_\_str\_\_和\_\_repr\_\_**

>>> class A():

def \_\_str\_\_(self):

return "Amusi"

>>> a = A()

>>> print(a)

Amusi

>>> a

<\_\_main\_\_.A object at 0x7f4c435db128>

>>> class B():

def \_\_repr\_\_(self):

return "Amusi"

>>> b = B()

>>> b

Amusi

>>> print(b)

Amusi

**代码实例**

* 首先创建一个MyTimer.py文件，写入：

# 导入time模块

import time as t

class MyTimer():

def \_\_init\_\_(self):

self.unit = ["年", "月", "日", "时", "分", "秒"]

self.prompt = "未开始计时！"

self.lasted = []

self.begin = 0

self.end = 0

def \_\_str\_\_(self):

return self.prompt

\_\_repr\_\_ = \_\_str\_\_

def \_\_add\_\_(self, other):

prompt = "总共运行了"

result = []

for index in range(6):

result.append(self.lasted[index] + other.lasted[index])

if result[index]:

prompt += (str(result[index]) + self.unit[index])

return prompt

# 开始计时

def start(self):

self.begin = t.localtime()

self.prompt = "提示: 先调用stop()停止计时!"

print("计时开始...")

# 停止计时

def stop(self):

self.end = t.localtime()

if not self.begin:

print("提示: 请先调用start()进行计时！")

else:

self.\_\_calc()

print("计时结束...")

# 内部方法，计算运算时间

def \_\_calc(self):

self.lasted = []

self.prompt = "总共运行了"

for index in range(6):

self.lasted.append(self.end[index] - self.begin[index])

if self.lasted[index]:

self.prompt += (str(self.lasted[index]) + self.unit[index])

# 为下一轮计时初始化变量

self.begin = 0

self.end = 0

* 然后运行上述MyTimer.py文件

>>> t1 = MyTimer()

>>> t1

未开始计时！

>>> t1.stop()

提示: 请先调用start()进行计时！

>>> t1.start()

计时开始...

>>> t1

提示: 先调用stop()停止计时!

>>> t1.stop()

计时结束...

>>> t1

总共运行了8秒

>>> t2 = MyTimer()

>>> t2.start()

计时开始...

>>> t2.stop()

计时结束...

>>> t2

总共运行了4秒

>>> t1 + t2

'总共运行了12秒'

**进阶定制**

* 如果开始计时的时间是（2022年2月22日16:30:30），停止时间是（2025年1月23日15:30:30），那么我们用停止时间减去开始时间的计算方式就会出现负数（3年-1月1天-1小时），你应该对此做一些转换.
* 现在的计算机速度都非常块，而且我们这个程序最小的计算单位只有秒，精度远远不够.

## 课时45 魔法方法: 属性访问

一般的，实例化对象可以通过”.”操作符访问属性值；也可以通过class的相关方法访问属性值.

下述将介绍\_\_getattr\_\_()、\_\_getattribute\_\_()、\_\_setattr\_\_()和\_\_delattr\_\_魔法方法，来实现属性访问.

**代码实例——getattr()方法**

>>> class C:

def \_\_init\_\_(self):

self.x = "X-man"

>>> c = C()

# 对象通过”.”访问属性

>>> c.x

'X-man'

# 使用class方法访问属性

>>> getattr(c, "x")

'X-man'

>>> getattr(c, "y", "没有这个属性")

'没有这个属性'

**代码实例——property()方法**

>>> class C:

def \_\_init\_\_(self, size = 10):

self.size = size

def getSize(self):

return self.size

def setSize(self, value):

self.size = value

def delSize(self):

del self.size

x = property(getSize, setSize, delSize)

>>> c = C()

>>> c.size

10

>>> c.getSize()

10

>>> c.x = 1

>>> c.x

1

>>> c.size

1

>>> c.getSize()

1

>>> del c.x

>>> c.size

Traceback (most recent call last):

File "<pyshell#28>", line 1, in <module>

c.size

AttributeError: 'C' object has no attribute 'size'

### 45.1 \_\_getattribute\_\_()

\_\_getattribute\_\_(self, name)定义当该类的属性被访问的行为

### 45.2 \_\_getattr\_\_()

\_\_getattr\_\_(self, name)定义当用户试图获取一个不存在的属性时的行为

### 45.3 \_\_setattr\_\_()

\_\_setattr\_\_(self, name, value)定义当一个属性被设置时的行为

### 45.4 \_\_delattr\_\_()

\_\_delattr\_\_(self, name)定义当一个属性被删除时的行为

注：在类中若重写上述方法，则覆盖object父类中的同名方法，所以要小心使用

\_\_getattribute\_\_()方法总在\_\_getattr\_\_()方法前被调用

**代码实例**

>>> class C:

def \_\_getattribute\_\_(self, name):

print("getattribute")

return super().\_\_getattribute\_\_(name)

def \_\_getattr\_\_(self, name):

print("getattr")

def \_\_setattr\_\_(self, name, value):

print("setattr")

super().\_\_setattr\_\_(name, value)

def \_\_delattr\_\_(self, name):

print("delattr")

super().\_\_delattr\_\_(name)

>>> c = C()

>>> c.x

getattribute

getattr

>>> c.x = 9

setattr

>>> c.x

getattribute

9

>>> del c.x

delattr

### 45.5课堂练习

* 写一个矩形类，默认有宽和高两个属性；
* 如果为一个名叫square的属性赋值，那么说明这是一个正方形，值就是正方形的边长，此时宽和高都应该等于边长.

**代码实例**

* 新建Rectangle.py，写入下述代码

class Rectangle:

def \_\_init\_\_(self, width=0, height=0):

self.width = width

self.height = height

# set

def \_\_setattr\_\_(self, name, value):

if name == "square":

self.width = value

self.height = value

else:

super().\_\_setattr\_\_(name, value)

def getArea(self):

return (self.width \* self.height)

* 运行上述代码

>>> r1 = Rectangle(3, 4)

>>> r1.getArea()

12

# 使用\_\_setattr\_\_()方法

>>> r1.square = 10

>>> r1.width

10

>>> r1.height

10

>>> r1.getArea()

100

### 练习题

### 动动手

## 课时46 魔法方法: 描述符（Property的原理）

描述符就是将某种特殊类型的类的实例指派给另一个类的属性

* \_\_get\_\_(self, instance, owner)
  + 用于访问属性，它返回属性的值
* \_\_set\_\_(self, instance, value)
  + 将在属性分配操作中调用，不返回任何内容
* \_\_delete\_\_(self, instance)
  + 控制删除操作，不返回任何内容

**代码实例——简单了解上述三个函数**

>>> class MyDescriptor:

def \_\_get\_\_(self, instance, owner):

print("getting...", self, instance, owner)

def \_\_set\_\_(self, instance, value):

print("setting...", self, instance, value)

def \_\_delete\_\_(self, instance):

print("deleting...", self, instance)

>>> class Test:

x = MyDescriptor()

# 实例化

>>> test = Test()

>>> test.x

getting... <\_\_main\_\_.MyDescriptor object at 0x7f5d3749c860> <\_\_main\_\_.Test object at 0x7f5d3749cf98> <class '\_\_main\_\_.Test'>

>>> test

<\_\_main\_\_.Test object at 0x7f5d3749cf98>

>>> test.x = "X-Man"

setting... <\_\_main\_\_.MyDescriptor object at 0x7f5d3749c860> <\_\_main\_\_.Test object at 0x7f5d3749cf98> X-Man

**代码实例——重写property类**

>>> class MyProperty:

def \_\_init\_\_(self, fget = None, fset = None, fdel = None):

self.fget = fget

self.fset = fset

self.fdel = fdel

def \_\_get\_\_(self, instance, owner):

return self.fget(instance)

def \_\_set\_\_(self, instance, value):

return self.fset(instance, value)

def \_\_delete\_\_(self, instance):

self.fdel(instance)

>>> class C:

def \_\_init\_\_(self):

self.\_x = None

def getx(self):

return self.\_x

def setx(self, value):

self.\_x = value

def delx(self):

del self.\_x

# 自定义的MyProperty相当于property的作用

x = MyProperty(getx, setx, delx)

>>> c = C()

>>> c.x = "X-Man"

>>> c.\_x

'X-Man'

>>> del c.x

>>> c.\_x

Traceback (most recent call last):

File "<pyshell#68>", line 1, in <module>

c.\_x

AttributeError: 'C' object has no attribute '\_x'

**代码实例——小应用**

* 先定义一个温度类，然后定义两个描述符类用于描述摄氏度和华氏度两个属性
* 要求两个属性会自动进行转换，也就是你可以给摄氏度这个属性赋值，然后打印的华氏度属性是自动转换后的结果
* 华氏度 = 摄氏度\*1.8 + 32

**# 新建Temperature.py**

# 摄氏度

class Celsius:

def \_\_init\_\_(self, value = 26.0):

self.value = float(value)

def \_\_get\_\_(self, instance, owner):

return self.value

def \_\_set\_\_(self, instance, value):

self.value = float(value)

# 华氏度

class Fahrenheit:

def \_\_get\_\_(self, instance, owner):

return instance.cel\*1.8 + 32

def \_\_set\_\_(self, instance, value):

instance.cel = (float(value)-32) / 1.8

# 温度

class Temperature:

cel = Celsius()

fah = Fahrenheit()

**# 运行上述代码**

>>> temp = Temperature()

>>> temp.cel

26.0

>>> temp.cel = 30

>>> temp.fah

86.0

>>> temp.fah = 100

>>> temp.cel

37.77777777777778

课时47魔法方法: 定制序列

**协议（Protocols）**与其它编程语言中的接口很相似，它规定你哪些方法必须要定义。然而，在Python中的协议就不那么正式。事实上，在Python中，协议更像是一种指南。

**[容器类型的协议](http://bbs.fishc.com/thread-48793-1-1.html)**

* + 不可变容器，只需要定义\_\_len\_\_()和\_\_getitem\_\_()方法
  + 可变容器，除了需要定义\_\_len\_\_()和\_\_getitem\_\_()方法之外，还需要定义\_\_setitem\_\_()和\_\_delitem\_\_()方法

**代码实例——小应用**

* 编写一个不可改变的自定义列表，要求记录列表中每个元素被访问的次数

**新建CountList.py**

class CountList:

# 初始化——\*args参数是可变数量的

def \_\_init\_\_(self, \*args):

# 列表推导式

self.values = [x for x in args]

# 创建一个新列表

self.count = {}.fromkeys(range(len(self.values)), 0)

def \_\_len\_\_(self):

return len(self, values)

def \_\_getitem\_\_(self, key):

self.count[key] += 1

return self.values[key]

**运行上述代码**

>>> c1 = CountList(1, 3, 5, 7, 9)

>>> c2 = CountList(2, 4, 6, 8, 10)

>>> c1[1]

3

>>> c2[1]

4

>>> c1[1] + c2[1]

7

>>> c1.count

{0: 0, 1: 2, 2: 0, 3: 0, 4: 0}

>>> c2[2]

6

# 不可变列表

>>> c2[2] = 100

Traceback (most recent call last):

File "<pyshell#13>", line 1, in <module>

c2[2] = 100

TypeError: 'CountList' object does not support item assignment

>>> c2.count

{0: 0, 1: 2, 2: 1, 3: 0, 4: 0}

### 练习题

### 动动手

课时48魔法方法: 迭代器

[迭代器（Iterator）](https://www.liaoxuefeng.com/wiki/0014316089557264a6b348958f449949df42a6d3a2e542c000/00143178254193589df9c612d2449618ea460e7a672a366000)

### 48.1 迭代器简单实例

**代码实例——字符串是一个迭代器**

>>> for i in "Amusi":

print(i)

A

m

u

s

i

**代码实例——列表是一个迭代器**

>>> list1 = ["Amusi", "Lucy", "Nancy"]

>>> for each in list1:

print(each)

Amusi

Lucy

Nancy

**代码实例——元组是一个迭代器**

>>> tuple1 = ("Amusi", "Jackie", "Jie")

>>> for each in tuple1:

print(each)

Amusi

Jackie

Jie

**代码实例——字典是一个迭代器**

>>> dict1 = {"Amusi":"Chen", "Lucy":"Luo", "Jackie":"Jie"}

>>> for each in dict1:

print("%s -> %s" % (each, dict1[each]))

Amusi -> Chen

Lucy -> Luo

Jackie -> Jie

### 48.2 iter()和next()

* iter()——获得当前迭代器的指针位置
* next()——迭代器指针指向下一个位置

**代码实例——iter()和next()的简单用法**

>>> str1 = "Amusi"

>>> it = iter(str1)

>>> next(it)

'A'

>>> next(it)

'm'

>>> next(it)

'u'

>>> next(it)

's'

>>> next(it)

'i'

>>> next(it)

Traceback (most recent call last):

File "<pyshell#45>", line 1, in <module>

next(it)

StopIteration

**代码实例——iter()和next()还原for循环**

>>> str1 = "Amusi"

>>> it = iter(str1)

>>> while(True):

try:

each = next(it)

except StopIteration:

break

print(each)

A

m

u

s

i

### 48.2 \_\_iter\_\_()和\_\_next\_\_()

**代码实例１——斐波那契数列**

>>> class Fibs:

def \_\_init\_\_(self):

self.a = 0

self.b = 1

def \_\_iter\_\_(self):

return self

def \_\_next\_\_(self):

self.a, self.b = self.b , self.a + self.b

return self.a

>>> fibs = Fibs()

>>> for each in fibs:

if each < 20:

print(each)

else:

break

1

1

2

3

5

8

13

**代码实例１——有限的斐波那契数列**

>>> class Fibs:

def \_\_init\_\_(self, n = 10):

self.a = 0

self.b = 1

self.n = n

def \_\_iter\_\_(self):

return self

def \_\_next\_\_(self):

self.a, self.b = self.b , self.a + self.b

if self.a > self.n:

raise StopIteration

return self.a

>>> fibs = Fibs()

>>> for each in fibs:

print(each)

1

1

2

3

5

8

### 练习题

### 动动手

课时49乱入: 生成器

[生成器（generator）](https://www.liaoxuefeng.com/wiki/0014316089557264a6b348958f449949df42a6d3a2e542c000/0014317799226173f45ce40636141b6abc8424e12b5fb27000)：一边循环一边计算的机制。

如果一个函数定义中包含yield关键字，那么这个函数就不再是一个普通函数，而是一个generator。yield相当于return的作用，会返回一个值，但不会退出当前函数。

生成器符合协同程序的要求。什么是协同程序？协同程序就是可以运行的独立函数调用，函数可以暂停或者挂起，并在需要的时候从程序离开的地方继续或者重写开始。

**代码实例——yield简单实例**

>>> def myGen():

print("生成器被执行")

yield 1

yield 2

>>> MyG = myGen()

>>> next(MyG)

生成器被执行

1

>>> next(MyG)

2

>>> next(MyG)

Traceback (most recent call last):

File "<pyshell#91>", line 1, in <module>

next(MyG)

StopIteration

>>> for i in myGen():

print(i)

生成器被执行

1

2

**代码实例——斐波那契数列**

>>> def lib():

a = 0

b = 1

while True:

a,b = b, a + b

yield a

>>> for each in lib():

if each > 100:

break

print(each, end=" ")

1 1 2 3 5 8 13 21 34 55 89

**代码实例——列表与元组生成式的不同**

# 列表生成式

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

>>> a

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

# 元组生成式

>>> a = (i for i in range(10))

>>> a

<generator object <genexpr> at 0x7f9bf106b3b8>

为什么两种近似的代码，输出的结果差别这么大？由元组生成式的返回值”generator”可知，此时a是一个生成器，所以可以利用生成器的特性，利用next对它进行操作。

# 利用生成器的特性，对元组生成式操作

>>> next(a)

0

>>> next(a)

1

>>> for each in a:

print(each)

2

3

4

5

6

7

8

9

# 元组作为函数的输入

>>> sum((i for i in range(100)))

4950

>>> sum(i for i in range(100))

4950

### 练习题

### 动动手

课时50模块: 模块就是程序

* 容器－>数据的封装
* 函数－>语句的封装
* 类　－>方法和属性的封装
* [模块](http://www.runoob.com/python/python-modules.html)－>程序（如.py文件）

注：导入模型时，一定要注意命令空间的问题（一般命名空间就是模块的名称）

**代码实例——模块**

* 新建hello.py，写入下述代码

def Hi():

print("Hi, Amusi!")

* 在同级目录下打开idle/idle3，输入Hi()，会出现错误

>>> import hello

>>> Hi()

Traceback (most recent call last):

File "<pyshell#1>", line 1, in <module>

Hi()

NameError: name 'Hi' is not defined

* 这里是命名空间的问题，所以可以在函数前直接加上命名空间的前缀

>>> hello.Hi()

Hi, Amusi!

50.1 **[导入模型的３种方法](https://www.cnblogs.com/allenblogs/archive/2011/11/15/2055149.html)**

1. **import modname**

用import语句导入模块，就在当前的名称空间（namespace）建立一个到该模块的引用。这种引用必须使用全称，也就是说，当使用在被导入模块中定义的函数时，必须包含模块的名字。所以不能只使用 funcname，而应该使用 modname.funcname

1. **from modname import funcname**

**from modname import fa, fb, fc**

**from modname import \***

与第1种方法的区别：funcname 被直接导入到本地名字空间去了，所以它可以直接使用，而不需要加上模块名的限定   
\* 表示，该模块的所有公共对象(public objects)都被导入到 当前的名称空间，也就是任何只要不是以”\_”开始的东西都会被导入。   
modname没有被定义，所以modname.funcname这种方式不起作用。并且，如果funcname如果已经被定义，它会被新版本（该导入模块中的版本）所替代。如果funcname被改成指向其他对象，modname不能不会觉察到。   
建议:

1)如果你要经常访问模块的属性和方法，且不想一遍又一遍地敲入模块名，使用 from module import   
2)如果你想要有选择地导入某些属性和方法，而不想要其它的，使用 from module import   
3)如果模块包含的属性和方法与你的某个模块同名，你必须使用import module来避免名字冲突   
4)尽量少用 from module import \* ，因为判定一个特殊的函数或属性是从哪来的有些困难，并且会造成调试和重构都更困难。

1. **（推荐）import modname as new\_modname**

这种方法与第１种方法类似，但命名空间的名称不再是modname，而是自定义名称new\_modname，一般该名称是缩写。如tensorflow改成tf

**代码实例——对导入模块的3种方法测试**

* 新建TemperatureConversion.py模块文件，写入

# 摄氏度转华氏度

def c2f(cel):

fah = cel \* 1.8 + 32

return fah

# 华氏度转摄氏度

def f2c(fah):

cel = (fah - 32) / 1.8

return cel

* [方法1]新建calc1.py调用文件，写入

import TemperatureConversion

print("32摄氏度 = %2.f华氏度" % (TemperatureConversion.c2f(32)))

print("99华氏度 = %2.f摄氏度" % (TemperatureConversion.f2c(99)))

* [方法2]新建calc2.py调用文件，写入

from TemperatureConversion import c2f, f2c

print("32摄氏度 = %2.f华氏度" % (c2f(32)))

print("99华氏度 = %2.f摄氏度" % (f2c(99)))

# 或者导入所有函数

from TemperatureConversion import \*

print("32摄氏度 = %2.f华氏度" % (c2f(32)))

print("99华氏度 = %2.f摄氏度" % (f2c(99)))

* [方法3]新建calc3.py调用文件，写入

import TemperatureConversion as tc

print("32摄氏度 = %2.f华氏度" % (tc.c2f(32)))

print("99华氏度 = %2.f摄氏度" % (tc.f2c(99)))

### 练习题

### 动动手

# 附件

**学习资源**

* [Python100例](http://www.runoob.com/python/python-100-examples.html)