Python

Python

- 基本数据类型
  - 数据转换
- Python 推导式
  - 列表推导式
  - 字典推导式
  - 集合推导式
  - 元组推导式
- Python不常使用的运算符
  - Python成员运算符
  - Python身份运算符
- Python数字
  - 数学函数
  - 随机数函数
  - 三角函数
  - 数学常量
- Python字符串
  - Python 访问字符串中的值
  - Python 字符串格式化
  - Python 的字符串内建函数
- Python列表
  - 列表更新
  - 列表元素删除
  - Python列表脚本操作符
  - Python列表函数&方法
- Python元组
  - 修改元组
  - 删除元组
  - 关于元组是不可变的
- Python字典
  - 创建空字典
  - 访问字典里的值
  - 修改字典
  - 删除字典元素
  - 字典键的特性
  - 字典内置函数&方法
- Python集合
  - 集合的交并等操作
  - 添加元素
  - 移除元素
  - 集合内置方法
- Python函数
  - 必需参数
  - 关键字参数
  - 默认参数
  - 匿名函数
    - 语法
- Python数据结构
  - 堆栈
  - 队列
  - 遍历技巧
- Python文件方法

open() 方法
file 对象
Python OS模块
Python 错误和异常
异常处理
try/except
try-finally 语句
抛出异常
Python面向对象
Python命名空间/作用域
global 和 nonlocal关键字
Python排序
Python urllib
urllib.request
模拟头部信息
urllib.parse
一个爬虫范例
Python JSON 数据解析

基本数据类型

Python3 中有六个标准的数据类型：

- Number（数字）
- String（字符串）
- List（列表）
- Tuple（元组）
- Set（集合）
- Dictionary（字典）

Python3 的六个标准数据类型中：

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

内置的 **type()** 函数可以用来查询变量所指的对象类型，此外还可以用 **isinstance** 来判断（isinstance 的第一个参数为一个实例对象，第二个参数为数据类型）。

isinstance 和 type 的区别在于：

- **type()不会认为子类是一种父类类型。**
- **isinstance()会认为子类是一种父类类型。**

注意：Python3 中，bool 是 int 的子类，True 和 False 可以和数字相加，**True==1、False==0** 会返回 True，但可以通过 **is来判断类型。**

数据转换

函数	描述
int(x [,base])	将x转换为一个整数
float(x)	将x转换到一个浮点数
complex(real [,imag])	创建一个复数
str(x)	将对象 x 转换为字符串
repr(x)	将对象 x 转换为表达式字符串
eval(str)	用来计算在字符串中的有效Python表达式,并返回一个对象
tuple(s)	将序列 s 转换为一个元组
list(s)	将序列 s 转换为一个列表
set(s)	转换为可变集合
dict(d)	创建一个字典。**d 必须是一个 (key, value)元组序列。**
frozenset(s)	转换为不可变集合
chr(x)	将一个整数转换为一个字符
ord(x)	**将一个字符转换为它的整数值**
hex(x)	将一个整数转换为一个十六进制字符串
oct(x)	将一个整数转换为一个八进制字符串

Python 推导式

Python 推导式是一种**独特的数据处理方式**，可以从一个数据序列构建另一个新的数据序列的结构体。

Python 支持各种数据结构的推导式：

- 列表(list)推导式
- 字典(dict)推导式
- 集合(set)推导式
- 元组(tuple)推导式

列表推导式

- 列表推导式格式为：

```
[表达式 for 变量 in 列表]
[out_exp_res for out_exp in input_list]

或者

[表达式 for 变量 in 列表 if 条件]
[out_exp_res for out_exp in input_list if condition]
```

- out_exp_res：**列表生成元素表达式，可以是有返回值的函数。**
- for out_exp in input_list：迭代 input_list 将 out_exp 传入到 out_exp_res 表达式中。
- if condition：**条件语句，可以过滤列表中不符合条件的值。**

> 过滤掉长度小于或等于3的字符串列表，并将剩下的转换成大写字母：

```
names = ['Bob','Tom','alice','Jerry','Wendy','Smith']
new_names = [name.upper()for name in names if len(name)>3]
print(new_names)
>>> ['ALICE', 'JERRY', 'WENDY', 'SMITH']
```

字典推导式

- 字典推导基本格式：

```
{ key_expr: value_expr for value in collection }

或

{ key_expr: value_expr for value in collection if condition }
```

> 使用字符串及其长度创建字典：

```
listdemo = ['Google','Runoob', 'Taobao']
# 将列表中各字符串值为键，各字符串的长度为值，组成键值对
newdict = {key:len(key) for key in listdemo}
newdict
>>> {'Google': 6, 'Runoob': 6, 'Taobao': 6}
```

集合推导式

- 集合推导式基本格式(和**列表推导式基本一致**)，，但是注意！！！，其**返回的集合的元素具有唯一性**：

```
{ expression for item in Sequence }
或
{ expression for item in Sequence if conditional }
```

> 判断不是 abc 的字母并输出：

```
a = {x for x in 'abracadabra' if x not in 'abc'}
a
>>> {'d', 'r'}
type(a)
>>> <class 'set'>
```

元组推导式

- 元组推导式基本格式：

```
(expression for item in Sequence )
或
(expression for item in Sequence if conditional )
```

元组推导式和列表推导式的用法也完全相同，只是元组推导式是用 **()** 圆括号将各部分括起来，而列表推导式用的是中括号 **[]**，另外**元组推导式返回的结果是一个生成器对象。**

> 生成一个包含数字 1~9 的元组：

```
a = (x for x in range(1,10))
a
>>> <generator object <genexpr> at 0x7faf6ee20a50>  # 返回的是生成器对象

tuple(a)       # 使用 tuple() 函数，可以直接将生成器对象转换成元组
>>> (1, 2, 3, 4, 5, 6, 7, 8, 9)
```

Python不常使用的运算符

Python成员运算符

除了以上的一些运算符之外，Python还支持成员运算符，测试实例中包含了一系列的成员，包括字符串，列表或元组。

运算符	描述	实例
in	如果在指定的序列中找到值返回 True，否则返回 False。	x 在 y 序列中 , 如果 x 在 y 序列中返回 True。
not in	如果在指定的序列中没有找到值返回 True，否则返回 False。	x 不在 y 序列中 , 如果 x 不在 y 序列中返回 True。

Python身份运算符

身份运算符用于比较两个对象的**存储单元**

运算符	描述	实例
is	is 是判断两个标识符是不是引用自一个对象	**x is y**, 类似 **id(x) == id(y)** , 如果引用的是同一个对象则返回 True，否则返回 False
is not	is not 是判断两个标识符是不是引用自不同对象	**x is not y** ， 类似 **id(a) != id(b)**。如果引用的不是同一个对象则返回结果 True，否则返回 False。

注： id() 函数用于获取对象内存地址。

Python数字

数学函数

函数	返回值 (描述)
abs(x)	返回数字的绝对值，如abs(-10) 返回 10
ceil(x)	返回数字的上入整数，如math.ceil(4.1) 返回
exp(x)	返回e的x次幂(ex),如math.exp(1) 返回2.718281828459045
fabs(x)	返回数字的绝对值，如math.fabs(-10) 返回10.0
floor(x)	返回数字的下舍整数，如math.floor(4.9)返回 4
log(x)	如math.log(math.e)返回1.0,math.log(100,10)返回2.0
log10(x)	返回以10为基数的x的对数，如math.log10(100)返回 2.0
max(x1, x2,...)	返回给定参数的最大值，参数可以为序列。
min(x1, x2,...)	返回给定参数的最小值，参数可以为序列。
modf(x)	返回x的整数部分与小数部分，两部分的数值符号与x相同，整数部分以浮点型表示。
pow(x, y)	x**y 运算后的值。
round(x [,n])	返回浮点数 x 的四舍五入值，如给出 n 值，则代表舍入到小数点后的位数。**其实准确的说是保留值将保留到离上一位更近的一端。**
sqrt(x)	返回数字x的平方根。
cmp(x, y)	如果 x < y 返回 -1, 如果 x == y 返回 0, 如果 x > y 返回 1。 **Python 3 已废弃，使用 (x>y)-(x<y) 替换。**

随机数函数

随机数可以用于数学，游戏，安全等领域中，还经常被嵌入到算法中，用以提高算法效率，并提高程序的安全性。

Python包含以下常用随机数函数：

函数	描述
choice(seq)	从序列的元素中随机挑选一个元素，比如random.choice(range(10))，从0到9中随机挑选一个整数。
randrange ([start,] stop [,step])	从指定范围内，按指定基数递增的集合中获取一个随机数，基数默认值为1
random()	随机生成下一个实数，它在**[0,1)范围内。**
seed([x])	改变随机数生成器的种子seed。如果你不了解其原理，你不必特别去设定seed，Python会帮你选择seed。
shuffle(lst)	**将序列的所有元素随机排序**
uniform(x, y)	随机生成下一个实数，它在[x,y]范围内。

三角函数

Python包括以下三角函数：

函数	描述
acos(x)	返回x的反余弦弧度值。
asin(x)	返回x的反正弦弧度值。
atan(x)	返回x的反正切弧度值。
atan2(y, x)	返回给定的 X 及 Y 坐标值的反正切值。
cos(x)	返回x的弧度的余弦值。
hypot(x, y)	**返回欧几里德范数 sqrt(x*x + y*y)。**
sin(x)	返回的x弧度的正弦值。
tan(x)	返回x弧度的正切值。
degrees(x)	**将弧度转换为角度,如degrees(math.pi/2)， 返回90.0**
radians(x)	**将角度转换为弧度**

数学常量

常量	描述
pi	数学常量 pi（圆周率，一般以π来表示）
e	数学常量 e，e即自然常数（自然常数）。

Python字符串

Python 访问字符串中的值

Python 访问子字符串，可以使用方括号 **[]** 来截取字符串，字符串的截取的语法格式如下：

```
变量[头下标:尾下标:步长]
```

索引值以 **0** 为开始值，**-1** 为从末尾的开始位置。

Python 字符串格式化

Python 支持格式化字符串的输出 。尽管这样可能会用到非常复杂的表达式，但最基本的用法是将一个值插入到一个有字符串格式符 %s 的字符串中。

- python字符串格式化符号:

符 号	描述
%c	格式化字符及其ASCII码
%s	格式化字符串
%d	格式化整数
%u	格式化无符号整型
%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	格式化浮点数字，可指定小数点后的精度
%e	用科学计数法格式化浮点数
%E	作用同%e，用科学计数法格式化浮点数
%g	%f和%e的简写
%G	%f 和 %E 的简写
%p	用十六进制数格式化变量的地址

- 格式化操作符辅助指令:

符号	功能
*	定义宽度或者小数点精度
-	用做左对齐
+	在正数前面显示加号(+)
	在正数前面显示空格
#	在八进制数前面显示零('0')，在十六进制前面显示'0x'或者'0X'(取决于用的是'x'还是'X')
0	显示的数字前面填充'0'而不是默认的空格
%	'%%'输出一个单一的'%'
(var)	映射变量(字典参数)
m.n.	m 是显示的最小总宽度,n 是小数点后的位数(如果可用的话)

Python 的字符串内建函数

- Python 的字符串常用内建函数如下：

序号	方法及描述
1	capitalize() 将字符串的第一个字符转换为大写
2	center(width, fillchar)返回一个指定的宽度 width 居中的字符串，fillchar 为填充的字符，默认为空格。
3	count(str, beg= 0,end=len(string)) 返回 str 在 string 里面出现的次数，如果 beg 或者 end 指定则返回指定范围内 str 出现的次数
4	bytes.decode(encoding="utf-8", errors="strict") Python3 中没有 decode 方法，但我们可以使用 bytes 对象的 decode() 方法来解码给定的 bytes 对象，这个 bytes 对象可以由 str.encode() 来编码返回。
5	encode(encoding='UTF-8',errors='strict') 以 encoding 指定的编码格式编码字符串，如果出错默认报一个ValueError 的异常，除非 errors 指定的是'ignore'或者'replace'
6	endswith(suffix, beg=0, end=len(string)) 检查字符串是否以 obj 结束，如果beg 或者 end 指定则检查指定的范围内是否以 obj 结束，如果是，返回 True,否则返回 False.
7	expandtabs(tabsize=8) 把字符串 string 中的 tab 符号转为空格，tab 符号默认的空格数是 8 。
8	find(str, beg=0, end=len(string)) 检测 str 是否包含在字符串中，如果指定范围 beg 和 end ，则检查是否包含在指定范围内，如果包含返回开始的索引值，否则返回-1
9	index(str, beg=0, end=len(string)) 跟find()方法一样，只不过如果str不在字符串中会报一个异常。
10	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
11	isalpha() 如果字符串至少有一个字符并且所有字符都是字母或中文字则返回 True, 否则返回 False
12	isdigit() 如果字符串只包含数字则返回 True 否则返回 False..
13	islower() 如果字符串中包含至少一个区分大小写的字符，并且所有这些(区分大小写的)字符都是小写，则返回 True，否则返回 False
14	isnumeric() 如果字符串中只包含数字字符，则返回 True，否则返回 False
15	isspace() 如果字符串中只包含空白，则返回 True，否则返回 False.
16	istitle() 如果字符串是标题化的(见 title())则返回 True，否则返回 False
17	isupper() 如果字符串中包含至少一个区分大小写的字符，并且所有这些(区分大小写的)字符都是大写，则返回 True，否则返回 False
18	join(seq) **以指定字符串作为分隔符，将 seq 中所有的元素(的字符串表示)合并为一个新的字符串**
19	len(string) **返回字符串长度**
20	ljust(width[, fillchar]) 返回一个原字符串左对齐,并使用 fillchar 填充至长度 width 的新字符串，fillchar 默认为空格。
21	lower() 转换字符串中所有大写字符为小写.

序号	方法及描述
22	lstrip() 截掉字符串左边的空格或指定字符。
23	maketrans() 创建字符映射的转换表，对于接受两个参数的最简单的调用方式，第一个参数是字符串，表示需要转换的字符，第二个参数也是字符串表示转换的目标。
24	max(str) 返回字符串 str 中最大的字母。
25	min(str) 返回字符串 str 中最小的字母。
26	replace(old, new [, max]) 把 将字符串中的 old 替换成 new,如果 max 指定，则替换不超过 max 次。
27	rfind(str, beg=0,end=len(string)) 类似于 find()函数，不过是从右边开始查找.
28	rindex(str, beg=0, end=len(string)) 类似于 index()，不过是从右边开始.
29	rjust(width,[, fillchar]) 返回一个原字符串右对齐,并使用fillchar(默认空格）填充至长度 width 的新字符串
30	rstrip() 删除字符串末尾的空格或指定字符。
31	split(str="", num=string.count(str)) **以 str 为分隔符截取字符串，如果 num 有指定值，则仅截取 num+1 个子字符串**
32	splitlines([keepends]) 按照行('\r', '\r\n', \n')分隔，返回一个包含各行作为元素的列表，如果参数 keepends 为 False，不包含换行符，如果为 True，则保留换行符。
33	startswith(substr, beg=0,end=len(string)) 检查字符串是否是以指定子字符串 substr 开头，是则返回 True，否则返回 False。如果beg 和 end 指定值，则在指定范围内检查。
34	strip([chars]) 在字符串上执行 lstrip()和 rstrip()
35	swapcase() 将字符串中大写转换为小写，小写转换为大写
36	title() 返回"标题化"的字符串,就是说所有单词都是以大写开始，其余字母均为小写(见 istitle())
37	translate(table, deletechars="") 根据 str 给出的表(包含 256 个字符)转换 string 的字符, 要过滤掉的字符放到 deletechars 参数中
38	upper() 转换字符串中的小写字母为大写
39	zfill (width) 返回长度为 width 的字符串，原字符串右对齐，前面填充0
40	isdecimal() 检查字符串是否只包含十进制字符，如果是返回 true，否则返回 false。

Python列表

列表都可以进行的操作包括索引，切片，加，乘，检查成员。

此外，Python 已经**内置确定序列的长度以及确定最大和最小的元素的方法。**

列表是最常用的 Python 数据类型，它可以作为一个方括号内的逗号分隔值出现。

列表的数据项不需要具有相同的类型

列表更新

```
List.append()
List.extend(列表)
List.insert(位置索引, )
```

列表元素删除

```
List.remove()
List.pop() 或 List.pop(位置索引)
del + 列表[位置索引]
```

Python列表脚本操作符

列表对 + 和 * 的操作符与字符串相似。+ 号用于组合列表，* 号用于重复列表。

如下所示:

Python 表达式	结果	描述
len([1, 2, 3])	3	长度
[1, 2, 3] + [4, 5, 6]	[1, 2, 3, 4, 5, 6]	组合
['Hi!'] * 4	['Hi!', 'Hi!', 'Hi!', 'Hi!']	重复
3 in [1, 2, 3]	True	元素是否存在于列表中
for x in [1, 2, 3]: print(x, end=" ")	1 2 3	迭代

Python列表函数&方法

- Python包含以下函数:

序号	函数
1	len(list) 列表元素个数
2	max(list) 返回列表元素最大值
3	min(list) 返回列表元素最小值
4	list(seq) 将元组转换为列表

- Python包含以下方法:

序号	方法
1	list.append(obj) 在列表末尾添加新的对象
2	list.count(obj) 统计某个元素在列表中出现的次数
3	list.extend(seq) 在列表末尾一次性追加另一个序列中的多个值（用新列表扩展原来的列表）
4	list.index(obj) 从列表中找出某个值第一个匹配项的索引位置
5	list.insert(index, obj) 将对象插入列表
6	list.pop([index=-1]) 移除列表中的一个元素（默认最后一个元素），并且返回该元素的值
7	list.remove(obj) 移除列表中某个值的第一个匹配项
8	list.reverse() 反向列表中元素
9	list.sort(key=None, reverse=False) 对原列表进行排序
10	list.clear() 清空列表
11	list.copy() 复制列表

Python元组

元组中只包含一个元素时，需要在元素后面添加逗号**,**，否则括号会被当作运算符使用：

修改元组

元组中的元素值是不允许修改的，但我们可以对元组进行切片连接组合

删除元组

元组中的元素值是不允许删除的，但我们可以使用del语句来删除整个元组

关于元组是不可变的

所谓元组的不可变指的是**元组所指向的内存中的内容不可变**，重新赋值的元组 tup，绑**定到新的对象，可以看作是一种伪改变**

Python字典

字典是另一种可变容器模型，且可存储任意类型对象。

字典的每个键值 **key=>value** 对用冒号 **:** 分割，每个对之间用逗号(,)分割，整个字典包括在花括号 **{}** 中 ,格式如下所示：

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

创建空字典

- 使用大括号 **{ }** 创建空字典
- 使用内建函数 **dict()** 重建字典

访问字典里的值

把相应的键放入到方括号中

修改字典

向字典添加新内容的方法是增加新的键/值对

删除字典元素

能删单一的元素也能清空字典，清空只需一项操作。

显式删除一个字典用del命令

```
del tinydict['Name'] # 删除键 'Name'
tinydict.clear()     # 清空字典
del tinydict         # 删除字典
```

字典键的特性

字典值可以是**任何的 python 对象**，既可以是标准的对象，也可以是用户定义的，但键不行。

两个重要的点需要记住：

1. 不允许同一个键出现两次。创建时如果**同一个键被赋值两次，后一个值会被记住**
2. **键必须不可变，所以可以用数字，字符串或元组充当**，而用列表就不行

字典内置函数&方法

- Python字典包含了以下内置函数：

序号	函数及描述	实例
1	len(dict) 计算字典元素个数，即键的总数。	`>>> tinydict = {'Name': 'Runoob', 'Age': 7, 'Class': 'First'} >>> len(tinydict) 3`
2	str(dict) 输出字典，可以打印的字符串表示。	`>>> tinydict = {'Name': 'Runoob', 'Age': 7, 'Class': 'First'} >>> str(tinydict) "{'Name': 'Runoob', 'Class': 'First', 'Age': 7}"`
3	type(variable) 返回输入的变量类型，如果变量是字典就返回字典类型。	`>>> tinydict = {'Name': 'Runoob', 'Age': 7, 'Class': 'First'} >>> type(tinydict) <class 'dict'>`

- Python字典包含了以下内置方法：

序号	函数及描述
1	radiansdict.clear() 删除字典内所有元素
2	radiansdict.copy() 返回一个字典的浅复制
3	radiansdict.fromkeys() 创建一个新字典，以序列seq中元素做字典的键，val为字典所有键对应的初始值
4	radiansdict.get(key, default=None) 返回指定键的值，如果键不在字典中返回 default 设置的默认值
5	key in dict 如果键在字典dict里返回true，否则返回false
6	radiansdict.items() 以列表返回一个视图对象
7	radiansdict.keys() 返回一个视图对象
8	radiansdict.setdefault(key, default=None) 和get()类似, 但如果键不存在于字典中，将会添加键并将值设为default
9	radiansdict.update(dict2) 把字典dict2的键/值对更新到dict里
10	radiansdict.values() 返回一个视图对象
11	pop(key[,default]) 删除字典给定键 key 所对应的值，返回值为被删除的值。key值必须给出。 否则，返回default值。
12	popitem() 随机返回并删除字典中的最后一对键和值。

Python集合

集合（set）是一个**无序的不重复**元素序列。

可以使用大括号 **{ }** 或者 **set()** 函数创建集合，注意：**创建一个空集合必须用 set() 而不是 { }**，因为 **{ }** 是用来创建一个空字典。

创建格式：

```
parame = {value01,value02,...}
或者
set(value)
```

集合的交并等操作

```
a = set('abracadabra')
b = set('alacazam')
a
>>> {'a', 'r', 'b', 'c', 'd'}
a - b                                   # 集合a中包含而集合b中不包含的元素
>>> {'r', 'd', 'b'}
a | b                                   # 集合a或b中包含的所有元素
>>> {'a', 'c', 'r', 'd', 'b', 'm', 'z', 'l'}
a & b                                   # 集合a和b中都包含了的元素
>>> {'a', 'c'}
a ^ b                                   # 不同时包含于a和b的元素
{'r', 'd', 'b', 'm', 'z', 'l'}
```

添加元素

```
s.add( x )
```

移除元素

```
#将元素 x 从集合 s 中移除，如果元素不存在，则会发生错误。
s.remove( x )
#如果元素不存在，不会发生错误
s.discard( x )
#我们也可以设置随机删除集合中的一个元素
s.pop()
```

集合内置方法

方法	描述
add()	为集合添加元素
clear()	移除集合中的所有元素
copy()	拷贝一个集合
difference()	返回多个集合的差集
difference_update()	移除集合中的元素，该元素在指定的集合也存在。
discard()	删除集合中指定的元素
intersection()	返回集合的交集
intersection_update()	返回集合的交集。
isdisjoint()	判断两个集合是否包含相同的元素，如果没有返回 True，否则返回 False。
issubset()	判断指定集合是否为该方法参数集合的子集。
issuperset()	判断该方法的参数集合是否为指定集合的子集
pop()	随机移除元素
remove()	移除指定元素
symmetric_difference()	返回两个集合中不重复的元素集合。
symmetric_difference_update()	移除当前集合中在另外一个指定集合相同的元素，并将另外一个指定集合中不同的元素插入到当前集合中。
union()	返回两个集合的并集
update()	给集合添加元素

Python函数

以下是调用函数时可使用的正式参数类型：

- 必需参数
- 关键字参数
- 默认参数
- 不定长参数

必需参数

必需参数须以正确的顺序传入函数。调用时的**数量**必须和声明时的一样。

关键字参数

关键字参数和函数调用关系紧密，函数调用**使用关键字参数来确定传入的参数值。**

使用关键字参数允许**函数调用时参数的顺序与声明时不一致**，因为 Python 解释器能够用参数名匹配参数值。

```
可写函数说明
def printinfo( name, age ):
   "打印任何传入的字符串"
   print ("名字: ", name)
   print ("年龄: ", age)
   return

#调用printinfo函数
printinfo( age=50, name="runoob" )
```

默认参数

调用函数时，**如果没有传递参数，则会使用默认参数。**以下实例中如果没有传入 age 参数，则使用默认值

```
#可写函数说明
def printinfo( name, age = 35 ):
   "打印任何传入的字符串"
   print ("名字: ", name)
   print ("年龄: ", age)
   return

#调用printinfo函数
printinfo( age=50, name="runoob" )
print ("------------------------")
printinfo( name="runoob" )
```

匿名函数

python 使用 lambda 来创建匿名函数。

所谓匿名，意即不再使用 def 语句这样标准的形式定义一个函数。

- lambda 只是一个表达式，函数体比 def 简单很多。
- lambda的主体是一个表达式，而不是一个代码块。仅仅能在lambda表达式中封装有限的逻辑进去。
- lambda 函数拥有自己的命名空间，且不能访问自己参数列表之外或全局命名空间里的参数。
- 虽然lambda函数看起来只能写一行，却不等同于C或C++的内联函数，后者的目的是调用小函数时不占用栈内存从而增加运行效率。

语法

lambda 函数的语法只包含一个语句，如下：

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

```
# 可写函数说明
sum = lambda arg1, arg2: arg1 + arg2

# 调用sum函数
print ("相加后的值为 : ", sum( 10, 20 ))
print ("相加后的值为 : ", sum( 20, 20 ))
```

Python数据结构

堆栈

- 用**List模拟**即可，List的方法挺适用于**先进后出**的形式

队列

Python的Queue模块中提供了队列类，包括**FIFO（先入先出)队列Queue**，**LIFO（后入先出）队列LifoQueue**，和**优先级队列PriorityQueue。**

常用方法：

- Queue.qsize() 返回队列的大小
- **Queue.empty() 如果队列为空，返回True,反之False**
- Queue.full() 如果队列满了，返回True,反之False，Queue.full 与 maxsize 大小对应
- **Queue.get([block[, timeout]])获取队列，timeout等待时间**
- Queue.get_nowait() 相当于Queue.get(False)，非阻塞方法
- **Queue.put(item) 写入队列，timeout等待时间**
- Queue.task_done() 在完成一项工作之后，Queue.task_done()函数向任务已经完成的队列发送一个信号。每个get()调用得到一个任务，接下来task_done()调用告诉队列该任务已经处理完毕。
- Queue.join() 实际上意味着等到队列为空，再执行别的操作

遍历技巧

在**字典**中遍历时，**关键字和对应的值可以使用 items() 方法同时解读**出来：

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

在**序列**中遍历时，**索引位置和对应值可以使用 enumerate() 函数**同时得到：

```
for i, v in enumerate(['tic', 'tac', 'toe']):
    print(i, v)
```

要**反向遍历**一个序列，首先指定这个序列，然后调用 **reversed() 函数**

```
for i in reversed(range(1, 10, 2)):
    print(i)
```

Python文件方法

open() 方法

Python open() 方法用于打开一个文件，并返回文件对象，在对文件进行处理过程都需要使用到这个函数，如果该文件无法被打开，会抛出 OSError。

注意：使用 open() 方法**一定要保证关闭文件对象，即调用 close() 方法。**

open() 函数常用形式是**接收两个参数：文件名(file)和模式(mode)。**

```
open(file, mode='r')
```

file 对象

file 对象使用 open 函数来创建，下表列出了 file 对象常用的函数：

序号	方法及描述
1	file.close()关闭文件。关闭后文件不能再进行读写操作。
2	file.flush()刷新文件内部缓冲，直接把内部缓冲区的数据立刻写入文件, 而不是被动的等待输出缓冲区写入。
3	file.fileno()返回一个整型的文件描述符(file descriptor FD 整型), 可以用在如os模块的read方法等一些底层操作上。
4	file.isatty()如果文件连接到一个终端设备返回 True，否则返回 False。
5	file.next()**Python 3 中的 File 对象不支持 next() 方法。**返回文件下一行。
6	[file.read(size])从文件读取指定的字节数，如果未给定或为负则读取所有。
7	[file.readline(size])读取整行，包括 "\n" 字符。
8	[file.readlines(sizeint])读取所有行并返回列表，若给定sizeint>0，返回总和大约为sizeint字节的行, 实际读取值可能比 sizeint 较大, 因为需要填充缓冲区。
9	[file.seek(offset, whence])移动文件读取指针到指定位置
10	file.tell()返回文件当前位置。
11	[file.truncate(size])从文件的首行首字符开始截断，截断文件为 size 个字符，无 size 表示从当前位置截断；截断之后后面的所有字符被删除，其中 windows 系统下的换行代表2个字符大小。
12	file.write(str)将字符串写入文件，返回的是写入的字符长度。
13	file.writelines(sequence)向文件写入一个序列字符串列表，如果需要换行则要自己加入每行的换行符。

Python OS模块

os 模块提供了非常丰富的方法用来处理文件和目录。常用的方法如下表所示：

序号	方法及描述
1	os.access(path, mode) 检验权限模式
2	os.chdir(path) 改变当前工作目录
3	os.chflags(path, flags) 设置路径的标记为数字标记。
4	os.chmod(path, mode) 更改权限
5	os.chown(path, uid, gid) 更改文件所有者
6	os.chroot(path) 改变当前进程的根目录
7	os.close(fd) 关闭文件描述符 fd
8	os.closerange(fd_low, fd_high) 关闭所有文件描述符，从 fd_low (包含) 到 fd_high (不包含), 错误会忽略
9	os.dup(fd) 复制文件描述符 fd
10	os.dup2(fd, fd2) 将一个文件描述符 fd 复制到另一个 fd2
11	os.fchdir(fd) 通过文件描述符改变当前工作目录
12	os.fchmod(fd, mode) 改变一个文件的访问权限，该文件由参数fd指定，参数mode是Unix下的文件访问权限。
13	os.fchown(fd, uid, gid) 修改一个文件的所有权，这个函数修改一个文件的用户ID和用户组ID，该文件由文件描述符fd指定。
14	os.fdatasync(fd) 强制将文件写入磁盘，该文件由文件描述符fd指定，但是不强制更新文件的状态信息。
15	os.fdopen(fd[, mode[, bufsize]]) 通过文件描述符 fd 创建一个文件对象，并返回这个文件对象
16	os.fpathconf(fd, name) 返回一个打开的文件的系统配置信息。name为检索的系统配置的值，它也许是一个定义系统值的字符串，这些名字在很多标准中指定（POSIX.1, Unix 95, Unix 98, 和其它）。
17	os.fstat(fd) 返回文件描述符fd的状态，像stat()。
18	os.fstatvfs(fd) 返回包含文件描述符fd的文件的文件系统的信息，Python 3.3 相等于statvfs()。
19	os.fsync(fd) 强制将文件描述符为fd的文件写入硬盘。
20	os.ftruncate(fd, length) 裁剪文件描述符fd对应的文件, 所以它最大不能超过文件大小。
21	os.getcwd() 返回当前工作目录
22	os.getcwdb() 返回一个当前工作目录的Unicode对象
23	os.isatty(fd) 如果文件描述符fd是打开的，同时与tty(-like)设备相连，则返回true, 否则False。
24	os.lchflags(path, flags) 设置路径的标记为数字标记，类似 chflags()，但是没有软链接

序号	方法及描述
25	os.lchmod(path, mode) 修改连接文件权限
26	os.lchown(path, uid, gid) 更改文件所有者，类似 chown，但是不追踪链接。
27	os.link(src, dst) 创建硬链接，名为参数 dst，指向参数 src
28	os.listdir(path) 返回path指定的文件夹包含的文件或文件夹的名字的列表。
29	os.lseek(fd, pos, how) 设置文件描述符 fd当前位置为pos, how方式修改: SEEK_SET 或者 0 设置从文件开始的计算的pos; SEEK_CUR或者 1 则从当前位置计算; os.SEEK_END或者2则从文件尾部开始. 在unix，Windows中有效
30	os.lstat(path) 像stat(),但是没有软链接
31	os.major(device) 从原始的设备号中提取设备major号码 (使用stat中的st_dev或者st_rdev field)。
32	os.makedev(major, minor) 以major和minor设备号组成一个原始设备号
33	[os.makedirs(path, mode]) 递归文件夹创建函数。像mkdir(), 但创建的所有intermediate-level文件夹需要包含子文件夹。
34	os.minor(device) 从原始的设备号中提取设备minor号码 (使用stat中的st_dev或者st_rdev field)。
35	[os.mkdir(path, mode]) 以数字mode的mode创建一个名为path的文件夹.默认的 mode 是 0777 (八进制)。
36	[os.mkfifo(path, mode]) 创建命名管道，mode 为数字，默认为 0666 (八进制)
37	[os.mknod(filename, mode=0600, device]) 创建一个名为filename文件系统节点（文件，设备特别文件或者命名pipe）。
38	[os.open(file, flags, mode]) 打开一个文件，并且设置需要的打开选项，mode参数是可选的
39	os.openpty() 打开一个新的伪终端对。返回 pty 和 tty的文件描述符。
40	os.pathconf(path, name) 返回相关文件的系统配置信息。
41	os.pipe() 创建一个管道. 返回一对文件描述符(r, w) 分别为读和写
42	os.popen(command[, mode[, bufsize]]) 从一个 command 打开一个管道
43	os.read(fd, n) 从文件描述符 fd 中读取最多 n 个字节，返回包含读取字节的字符串，文件描述符 fd对应文件已达到结尾, 返回一个空字符串。
44	os.readlink(path) 返回软链接所指向的文件
45	os.remove(path) 删除路径为path的文件。如果path 是一个文件夹，将抛出OSError; 查看下面的rmdir()删除一个 directory。
46	os.removedirs(path) 递归删除目录。
47	os.rename(src, dst) 重命名文件或目录，从 src 到 dst
48	os.renames(old, new) 递归地对目录进行更名，也可以对文件进行更名。

序号	方法及描述
49	os.rmdir(path) 删除path指定的空目录，如果目录非空，则抛出一个OSError异常。
50	os.stat(path) 获取path指定的路径的信息，功能等同于C API中的stat()系统调用。
51	[os.stat_float_times(newvalue]) 决定stat_result是否以float对象显示时间戳
52	os.statvfs(path) 获取指定路径的文件系统统计信息
53	os.symlink(src, dst) 创建一个软链接
54	os.tcgetpgrp(fd) 返回与终端fd（一个由os.open()返回的打开的文件描述符）关联的进程组
55	os.tcsetpgrp(fd, pg) 设置与终端fd（一个由os.open()返回的打开的文件描述符）关联的进程组为pg。
56	os.tempnam([dir[, prefix]]) **Python3 中已删除。**返回唯一的路径名用于创建临时文件。
57	os.tmpfile() **Python3 中已删除。**返回一个打开的模式为(w+b)的文件对象 .这文件对象没有文件夹入口，没有文件描述符，将会自动删除。
58	os.tmpnam() **Python3 中已删除。**为创建一个临时文件返回一个唯一的路径
59	os.ttyname(fd) 返回一个字符串，它表示与文件描述符fd 关联的终端设备。如果fd 没有与终端设备关联，则引发一个异常。
60	os.unlink(path) 删除文件路径
61	os.utime(path, times) 返回指定的path文件的访问和修改的时间。
62	[os.walk(top[, topdown=True[, onerror=None[, followlinks=False]]])](https://www.runoob.com/python3/python3-os-walk.html) 输出在文件夹中的文件名通过在树中游走，向上或者向下。
63	os.write(fd, str) 写入字符串到文件描述符 fd中. 返回实际写入的字符串长度
64	os.path 模块 获取文件的属性信息。
65	os.pardir() 获取当前目录的父目录，以字符串形式显示目录名。

Python 错误和异常

异常处理

try/except

异常捕捉可以使用 **try/except** 语句

- 实例

```
while True:
    try:
        x = int(input("请输入一个数字: "))
        break
    except ValueError:
        print("您输入的不是数字，请再次尝试输入！")
```

try 语句按照如下方式工作；

- 首先，执行 try 子句（在关键字 try 和关键字 except 之间的语句）。
- 如果没有异常发生，忽略 except 子句，try 子句执行后结束。
- 如果在执行 try 子句的过程中发生了异常，那么 try 子句余下的部分将被忽略。如果异常的类型和 except 之后的名称相符，那么对应的 except 子句将被执行。
- 如果一个异常没有与任何的 except 匹配，那么这个异常将会传递给上层的 try 中。

一个 try 语句可能包含多个except子句，分别来处理不同的特定的异常。最多只有一个分支会被执行。

处理程序将只针对对应的 try 子句中的异常进行处理，而不是其他的 try 的处理程序中的异常。

最后一个except子句可以忽略异常的名称，它将被当作通配符使用。你可以使用这种方法打印一个错误信息，然后再次把异常抛出。

```
import sys

try:
    f = open('myfile.txt')
    s = f.readline()
    i = int(s.strip())
except OSError as err:
    print("OS error: {0}".format(err))
except ValueError:
    print("Could not convert data to an integer.")
except:
    print("Unexpected error:", sys.exc_info()[0])
    raise
```

try-finally 语句

try-finally 语句无论是否发生异常都将执行最后的代码。

抛出异常

Python 使用 raise 语句抛出一个指定的异常。

raise语法格式如下：

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

- 实例

```
x = 10
if x > 5:
    raise Exception('x 不能大于 5。x 的值为: {}'.format(x))
```

Python面向对象

- Java模块已经有了很详细的学习

Python命名空间/作用域

global 和 nonlocal关键字

当内部作用域想修改外部作用域的变量时，就要用到 global 和 nonlocal 关键字了。

```
num = 1
def fun1():
    global num  # 需要使用 global 关键字声明
    print(num)
    num = 123
    print(num)
fun1()
print(num)
```

如果要修改嵌套作用域（**enclosing 作用域，外层非全局作用域**）中的变量则需要 nonlocal 关键字了

```
def outer():
    num = 10
    def inner():
        nonlocal num   # nonlocal关键字声明
        num = 100
        print(num)
    inner()
    print(num)
outer()
```

Python排序

- operator模块有itemgetter，attrgetter

```
>>> from operator import itemgetter, attrgetter
>>> sorted(student_tuples, key=itemgetter(2))
[('dave', 'B', 10), ('jane', 'B', 12), ('john', 'A', 15)]
>>> sorted(student_objects, key=attrgetter('age'))
[('dave', 'B', 10), ('jane', 'B', 12), ('john', 'A', 15)]
```

- operator模块还允许多级的排序，例如，先以grade，然后再以age来排序：

```

>>> sorted(student_tuples, key=itemgetter(1,2))
[('john', 'A', 15), ('dave', 'B', 10), ('jane', 'B', 12)]
>>> sorted(student_objects, key=attrgetter('grade', 'age'))
[('john', 'A', 15), ('dave', 'B', 10), ('jane', 'B', 12)]
```

Python urllib

urllib 包 包含以下几个模块：

- **urllib.request** - 打开和读取 URL。
- **urllib.error** - 包含 urllib.request 抛出的异常。
- **urllib.parse** - 解析 URL。
- **urllib.robotparser** - 解析 robots.txt 文件。

urllib.request

urllib.request 定义了一些打开 URL 的函数和类，包含授权验证、重定向、浏览器 cookies等。

urllib.request 可以**模拟浏览器的一个请求发起过程。**

我们可以使用 **urllib.request 的 urlopen 方法来打开一个 URL**，语法格式如下：

```
urllib.request.urlopen(url, data=None, [timeout, ]*, cafile=None, capath=None,
cadefault=False, context=None)
```

- **url**：url 地址。
- **data**：发送到服务器的其他数据对象，默认为 None。
- **timeout**：设置访问超时时间。
- **cafile 和 capath**：cafile 为 CA 证书， capath 为 CA 证书的路径，使用 HTTPS 需要用到。
- **cadefault**：已经被弃用。
- **context**：ssl.SSLContext类型，用来指定 SSL 设置。

- 实例：

```
from urllib.request import urlopen

myURL = urlopen("https://www.runoob.com/")
print(myURL.read().decode('utf-8'))

#以上代码使用 urlopen 打开一个 URL，然后使用 read() 函数获取网页的 HTML 实体代码。
#read() 是读取整个网页内容，我们可以指定读取的长度
```

除了 read() 函数外，还包含以下两个读取网页内容的函数：

- **readline()** - 读取文件的一行内容

  ```
  from urllib.request import urlopen

  myURL = urlopen("https://www.runoob.com/")
  print(myURL.readline().decode('utf-8')) #读取一行内容
  ```

- **readlines()** - 读取文件的全部内容，它会把读取的内容赋值给一个列表变量。

  ```
  from urllib.request import urlopen

  myURL = urlopen("https://www.runoob.com/")
  lines = myURL.readlines()
  for line in lines:
      print(line)
  ```

我们在对网页进行抓取时，经常需要判断网页是否可以正常访问，这里我们就可以使用 getcode() 函数获取网页状态码，返回 200 说明网页正常，返回 404 说明网页不存在:

- 实例:

```
import urllib.request

myURL1 = urllib.request.urlopen("https://www.runoob.com/")
print(myURL1.getcode())   # 200

try:
    myURL2 = urllib.request.urlopen("https://www.runoob.com/no.html")
except urllib.error.HTTPError as e:
    if e.code == 404:
        print(404)   # 404
```

模拟头部信息

我们抓取网页一般需要对 **headers（网页头信息）进行模拟**，这时候需要使用到 **urllib.request.Request** 类:

```
class urllib.request.Request(url, data=None, headers={}, origin_req_host=None,
unverifiable=False, method=None)
```

- **url**：url 地址。
- **data**：发送到服务器的其他数据对象，默认为 None。
- **headers**：HTTP 请求的头部信息，字典格式。
- **origin_req_host**：请求的主机地址，IP 或域名。
- **unverifiable**：很少用整个参数，用于设置网页是否需要验证，默认是False。。
- **method**：请求方法， 如 GET、POST、DELETE、PUT等。

urllib.parse

urllib.parse 用于解析 URL

```
urllib.parse.urlencode(data).encode('utf-8')
```

一个爬虫范例

```
import urllib.request
import urllib.parse
import json
content = input("请输入需要翻译的内容: ")
# 网址
url = "https://fanyi.youdao.com/translate"

# header = {}
# header['User-Agent'] = 'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/75.0.3770.100 Safari/537.36'
```

```
# 提交数据
data = {}
data['i'] = content
data['from'] = 'AUTO'
data['to'] = 'AUTO'
data['smartresult'] = 'dict'
data['client'] = 'fanyideskweb'
data['salt'] = '16261468224906'
data['sign'] = 'bc84f69a9e3553Od06029798c32431f4'
data['lts'] = '1626146822490'
data['bv'] = 'b396e111b686137a6ec711ea651ad37c'
data['doctype'] = 'json'
data['version'] = '2.1'
data['keyfrom'] = 'fanyi.web'
data['action'] = 'FY_BY_CLICKBUTTION'

# 编码
data = urllib.parse.urlencode(data).encode('utf-8')

# 模拟头部信息，发送请求
req = urllib.request.Request(url,data)

# 添加头部信息
# req.add_header('User-Agent','Mozilla/5.0 (Windows NT 10.0; WOW64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/75.0.3770.100 Safari/537.36')

# 获取发送请求后的网页
response = urllib.request.urlopen(req)

# 以utf-8的形式读取
html = response.read().decode('utf-8')

# json解析
target = json.loads(html)
print(target['translateResult'][0][0]['tgt'])
```

Python JSON 数据解析

JSON (JavaScript Object Notation) 是一种轻量级的数据交换格式。

Python3 中可以使用 json 模块来对 JSON 数据进行编解码，它包含了两个函数：

- **json.dumps():** 对数据进行编码。
- **json.loads():** 对数据进行解码。