# Python基础

## 数据类型和变量

### 转义字符 ’\’ ：

\n 换行；\t 制表符；\\ 字符\；

r’ ‘ 表示‘’内部字符不转义；‘’’…’’’ 表示多行内容；

### 空值

None

### 变量

变量本身类型不固定的语言称之为动态语言，与之对应的是静态语言。

### 除法运算

/ 除法计算结果是浮点数，即使是两个整数恰好整除，结果也是浮点数；

// 除法只取结果的整数部分；

% 余数运算。

无论整数做 // 除法还是取余数，结果永远是整数，所以，整数运算结果永远是精确的。

## 字符串和编码

### 字符编码

ord()函数获取字符的整数表示，chr()函数把编码转换为对应的字符

以Unicode表示的str通过encode()方法可以编码为指定的bytes

把bytes变为str，就需要用decode()方法，如果bytes中只有一小部分无效的字节，可以传入errors='ignore'忽略错误的字节：

# -\*- coding: utf-8 -\*- 为了告诉Python解释器，按照UTF-8编码读取源代码，否则，你在源代码中写的中文输出可能会有乱码。

| **占位符** | **替换内容** |
| --- | --- |
| %d | 整数 |
| %f | 浮点数 |
| %s | 字符串 |
| %x | 十六进制整数 |

## 3. 使用list和tuple

### 1. list

往list中追加元素到末尾：classmates.append(' ')

把元素插入到指定的位置，比如索引号为1的位置：classmates.insert(1, 'Jack')

删除list末尾的元素：classmates.pop()

要删除指定位置的元素，用pop(i)方法：classmates[1] = 'Sarah'

### Tuple

tuple和list非常类似，但是tuple一旦初始化就不能修改。因为tuple不可变，所以代码更安全。如果可能，能用tuple代替list就尽量用tuple。

classmates = ('Michael', 'Bob', 'Tracy')

只有1个元素的tuple定义时必须加一个逗号,，来消除歧义：t = (1,)

## 4. 条件判断

If语句注意不要少写了冒号:

if语句的完整形式就是：

if <条件判断1>:

<执行1>

elif <条件判断2>:

<执行2>

elif <条件判断3>:

<执行3>

else:

<执行4>

## 循环

1. for x in ...循环就是把每个元素代入变量x，然后执行缩进块的语句

range()函数，可以生成一个整数序列，比如range(5)生成的序列是从0开始小于5的整数：

1. while循环
2. break 退出循环
3. continue 跳过当前循环，开始下一循环。

## 6. Dict和set

1. dict

>>> d = {'Michael': 95, 'Bob': 75, 'Tracy': 85}

>>> d['Michael']

95

要删除一个key，用pop(key)方法，对应的value也会从dict中删除：

>>> d.pop('Bob')

75

>>> d

{'Michael': 95, 'Tracy': 85}

请务必注意，dict内部存放的顺序和key放入的顺序是没有关系的。

和list比较，dict有以下几个特点：

1.查找和插入的速度极快，不会随着key的增加而变慢；

2.需要占用大量的内存，内存浪费多。

而list相反：

1.查找和插入的时间随着元素的增加而增加；

2.占用空间小，浪费内存很少。

所以，dict是用空间来换取时间的一种方法。

**要牢记的第一条就是dict的key必须是不可变对象。**

2. set

通过add(key)方法可以添加元素到set中，可以重复添加，但不会有效果：

>>> s.add(4)

>>> s

{1, 2, 3, 4}

>>> s.add(4)

>>> s

{1, 2, 3, 4}

通过remove(key)方法可以删除元素：

>>> s.remove(4)

>>> s

{1, 2, 3}

set和dict的唯一区别仅在于没有存储对应的value，但是，set的原理和dict一样，所以，同样不可以放入可变对象，

# 2. 函数

## 1. 定义函数

### 1. 空函数

pass语句什么都不做，那有什么用？实际上pass可以用来作为占位符，比如现在还没想好怎么写函数的代码，就可以先放一个pass，让代码能运行起来。

### 2. 参数检查

让我们修改一下my\_abs的定义，对参数类型做检查，只允许整数和浮点数类型的参数。数据类型检查可以用内置函数isinstance()实现：

def my\_abs(x):

if not isinstance(x, (int, float)):

raise TypeError('bad operand type')

小结

定义函数时，需要确定函数名和参数个数；

如果有必要，可以先对参数的数据类型做检查；

函数体内部可以用return随时返回函数结果；

函数执行完毕也没有return语句时，自动return None。

函数可以同时返回多个值，但其实就是一个tuple。

## 2. 函数参数

### 1. 默认参数

def power(x, n=2):

s = 1

while n > 0:

n = n - 1

s = s \* x

return s

**定义默认参数要牢记一点：默认参数必须指向不变对象！**

### 2. 可变参数

定义可变参数和定义一个list或tuple参数相比，仅仅在参数前面加了一个\*号。在函数内部，参数numbers接收到的是一个tuple，因此，函数代码完全不变。但是，调用该函数时，可以传入任意个参数，包括0个参数：

\*nums表示把nums这个list的所有元素作为可变参数传进去。这种写法相当有用，而且很常见。

### 3. 关键字参数

def person(name, age, **\*\*kw**):

print('name:', name, 'age:', age, 'other:', kw)

### 4. 命名关键字参数

使用命名关键字参数时，要特别注意，如果没有可变参数，就必须加一个\*作为特殊分隔符。如果缺少\*，Python解释器将无法识别位置参数和命名关键字参数：

### 参数组合

在Python中定义函数，可以用必选参数、默认参数、可变参数、关键字参数和命名关键字参数，这5种参数都可以组合使用。但是请注意，参数定义的顺序必须是：**必选参数、默认参数、可变参数、命名关键字参数和关键字参数**。

**小结**

Python的函数具有非常灵活的参数形态，既可以实现简单的调用，又可以传入非常复杂的参数。

默认参数一定要用不可变对象，如果是可变对象，程序运行时会有逻辑错误！

要注意定义可变参数和关键字参数的语法：

\*args是可变参数，args接收的是一个tuple；

\*\*kw是关键字参数，kw接收的是一个dict。

以及调用函数时如何传入可变参数和关键字参数的语法：

可变参数既可以直接传入：func(1, 2, 3)，又可以先组装list或tuple，再通过\*args传入：func(\*(1, 2, 3))；

关键字参数既可以直接传入：func(a=1, b=2)，又可以先组装dict，再通过\*\*kw传入：func(\*\*{'a': 1, 'b': 2})。

使用\*args和\*\*kw是Python的习惯写法，当然也可以用其他参数名，但最好使用习惯用法。

命名的关键字参数是为了限制调用者可以传入的参数名，同时可以提供默认值。

定义命名的关键字参数在没有可变参数的情况下不要忘了写分隔符\*，否则定义的将是位置参数。

## 递归函数

解决递归调用栈溢出的方法是通过尾递归优化，事实上尾递归和循环的效果是一样的，所以，把循环看成是一种特殊的尾递归函数也是可以的。

尾递归是指，在函数返回的时候，调用自身本身，并且，return语句不能包含表达式。这样，编译器或者解释器就可以把尾递归做优化，使递归本身无论调用多少次，都只占用一个栈帧，不会出现栈溢出的情况。

上面的fact(n)函数由于return n \* fact(n - 1)引入了乘法表达式，所以就不是尾递归了。要改成尾递归方式，需要多一点代码，主要是要把每一步的乘积传入到递归函数中：

def fact(n):

return fact\_iter(n, 1)

def fact\_iter(num, product):

if num == 1:

return product

return fact\_iter(num - 1, num \* product)

# 3. 高级特性

## 1. 切片

L[0:3]表示，从索引0开始取，直到索引3为止，但不包括索引3。即索引0，1，2，正好是3个元素。

如果第一个索引是0，还可以省略：

List, tuple, 字符串均可以使用切片

## 2. 迭代

For…in…:

默认情况下，dict迭代的是key。如果要迭代value，可以用for value in d.values()，如果要同时迭代key和value，可以用for k, v in d.items()。

Python内置的enumerate函数可以把一个list变成索引-元素对，这样就可以在for循环中同时迭代索引和元素本身：

>>> for i, value in enumerate(['A', 'B', 'C']):

... print(i, value)

...

0 A

1 B

2 C

如何判断一个对象是可迭代对象呢？方法是通过collections模块的Iterable类型判断：

>>> from collections import Iterable

>>> isinstance('abc', Iterable) # str是否可迭代

True

>>> isinstance([1,2,3], Iterable) # list是否可迭代

True

>>> isinstance(123, Iterable) # 整数是否可迭代

False

## 3. 列表生成式

列表生成式则可以用一行语句代替循环生成上面的list：

>>> [x \* x for x in range(1, 11)]

[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]

写列表生成式时，把要生成的元素x \* x放到前面，后面跟for循环，就可以把list创建出来。

for循环后面还可以加上if判断，这样我们就可以筛选出仅偶数的平方：

>>> [x \* x for x in range(1, 11) if x % 2 == 0]

[4, 16, 36, 64, 100]

还可以使用两层循环，可以生成全排列：

>>> [m + n for m in 'ABC' for n in 'XYZ']

['AX', 'AY', 'AZ', 'BX', 'BY', 'BZ', 'CX', 'CY', 'CZ']

最后把一个list中所有的字符串变成小写：

>>> L = ['Hello', 'World', 'IBM', 'Apple']

>>> [s.lower() for s in L]

['hello', 'world', 'ibm', 'apple']

## 生成器

要创建一个generator，有很多种方法。第一种方法很简单，只要把一个列表生成式的[]改成()，就创建了一个generator：

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

## 迭代器

可以使用isinstance()判断一个对象是否是Iterator对象：

>>> from collections import Iterator

>>> isinstance((x for x in range(10)), Iterator)

True

>>> isinstance([], Iterator)

False

>>> isinstance({}, Iterator)

False

>>> isinstance('abc', Iterator)

False

**小结**

凡是可作用于for循环的对象都是Iterable类型；

凡是可作用于next()函数的对象都是Iterator类型，它们表示一个惰性计算的序列；

集合数据类型如list、dict、str等是Iterable但不是Iterator，不过可以通过iter()函数获得一个Iterator对象。

Python的for循环本质上就是通过不断调用next()函数实现的，