列表推导式 list comprehension

列表推导式是用可迭代对象创建列表的表达式

作用:

用简易方法生成列表

语法:

[ 表达式 for 变量 in 可迭代对象 ]

或

[ 表达式 for 变量 in 可迭代对象 if 真值表达式]

说明:

if 子句部分可以省略,省略后将对所有生成的对象进行处理

如果if真值表达式的布尔值为False,则可迭代对象生成的

数据将被丢弃

示例:

# 以下生成一个数值为 1 ~ 9之间整数的平方的列表

L = [ x\*\*2 for x in range(1, 10)] # L = [1, 4, 9, 16, ... 81]

# 等同于:

L = []

for x in range(1, 10):

L.append(x \*\* 2)

练习:

以下生成1个数值在 1~9之间的奇数的平方的列表

L = [x\*\*2 for x in range(1, 9) if x % 2 == 1]

等同于:

L = []

for x in range(1, 9):

if x % 2 == 1:

1. append(x \*\* 2)

输入一个开始的整数用begin绑定

输入一个结束的整数用end绑定

将从begin开始,到end结束的所有偶数存于列表中,并打印这

个列表

(建议用列表推导式实现)

列表推导式嵌套

语法:

[ 表达式 for 变量1 in 可迭代对象 if 真值表达式1

for 变量2 in 可迭代对象2 if 真值表达式2

...

]

示例:

将列表[10, 20, 30] 中的元素与 列表[1, 2, 3]分别

相加,将得到的数据放于一个列表中

L = [x + y for x in [10, 20, 30]

for y in [1,2,3]]

等同于:

L = []

for x in [10, 20, 30]:

for y in [1, 2, 3]:

L.append(x + y)

练习:

1. 有一些数存在于列表中,如:

L = [1, 3, 2, 1, 6, 4, 2, .... 98, 82]

1) 将列表中出现的数字存入另一个列表L2中

　　　要求:

重复出现多次的数字，只在L2列表中保留一份(去重)

2) 将列表中出现两次的数字存于列表L3中，在L3中只保留

　　 一份

2. 生成前40 个斐波那契数(Fibonacci)

1 1 2 3 5 8 13 ...

要求，将这些数保存在列表中

　最后打印出这些数

元组 tuple

元组的定义:

元组是不可改变的序列容器

同list一样，元组可以存放任意类型的数据元素

注:

不可改变是指一但创建，不可以添加和删除数据元素

元组的表示方式:

用小括号 () 括起来．单个元素括起来后加逗号(,) 号区分

单个对象还是元组

创建空元组的字面值:

t = () # t绑定空元组

创建非空元组的字面值:

t = 200,

t = (20,)

t = (1, 2, 3)

t = 100,200,300

type(x)　 函数用来返回x的类型

元组的错误示例:

t = (20) # t 绑定整数20,不是元组

x, y, z = 100, 200, 300 # 序列赋值

x, y, z = (1, 2, 3) # 序列赋值

x, y, z = "ABC" # 序列赋值

x, y, z = [10, 20, 30] # 序列赋值

x, y = y, x # 为什么能实现交换? y, x就是在创建元组

格式化字符串中:

"姓名:%s, 年龄:%d" % ("tarena", 15)

元组的构造(创建)函数 tuple

tuple() 创建一个空的元组，等同于 ()

tuple(iterable) 用可迭代对象创建一个空的元组

示例:

t = tuple()

t = tuple("ABC")

t = tuple([x\*\*2 for x in range(10)])

t = tuple(range(10))

元组的运算:

算术运算:

+ += \* \*=

+ 加号用于拼接

t = (1, 2, 3) + (4, 5, 6)

+= 用于拼接运算后，用变量绑定新元组

x = (1, 2, 3)

x += (4, 5, 6) # x = (1, 2, 3, 4, 5, 6)

\* 生成重复的元组

x = (1, 2) \* 3 # x = (1,2, 1, 2, 1, 2)

\*= 生成重复的元组，用变量绑定新元组

x = (1, 2)

x \*= 3 # (1, 2, 1, 2, 1, 2)

可变和不可变对象的本质区别:

列表:

L1 = [1, 2, 3]

L2 = L1

L2 += [4, 5, 6]

print(L2) # [1, 2, 3, 4, 5, 6]

print(L1) # [1, 2, 3, 4, 5, 6]

元组:

L1 = (1, 2, 3)

L2 = L1

L2 += (4, 5, 6)

print(L2) # (1, 2, 3, 4, 5, 6)

print(L1) # (1, 2, 3) 为什么？

元组的比较运算符

< <= > >= == !=

规则同列表的比较规则

in / not in 运算符

规则同列表的规则一致

索引/切片运算

v = 元组[整数]

t = 元组[:]

t = 元组[::]

同列表取值规则完全相同,元组不支持索引赋值和切片赋值

元组的方法:

文档参见:

python\_base\_docs\_html/tuple.html

序列相关的函数

len(x) max(x) min(x) sum(x) any(x) all(x)

三个序列的构造函数:

str([obj])

list([可迭代对象])

tuple([可迭代对象])

两个内建函数:

reversed(可迭代对象) 返回反向顺序的可迭代对象

sorted(可迭代对象, reverse=False) 返回已排序对象的

列表

示例:

L = [8, 6, 4, 3, 7]

for x in L:

print(x) # ???

for x in reversed(L):

print(x) # 7, 3, 4, 6, 8

L2 = sorted(L) # L2 = [3, 4, 6, 7, 8]

L3 = sorted(L, reverse=True) # L3 = [8,7,6,4,3]

字典 dict

什么是字典

字典是一种可变的容器,可以存储任意类型的数据

字典中的每个数据都是用"键"(key) 进行索引,而不像序列

(字符串,列表,元组)可以用下标来索引

字典中的数据以键(key)-值(value)对形式进行映射存储

字典中的数据没有先后顺序关系,字典的存储是无序的

字典的键不能重复,且只能用不可变类型作为字典的键

字面的表示方式:

字典的表示方式是{} 括起来,以冒号(:)分隔键-值对,各

键-值对之间用逗号分隔开

创建空典的字面值

d = {} # d 绑定空字典

创建非空的字典:

d = {'name': 'tarena', 'age': 15}

d = {'name': None, 'age': 1}

d = {'a': list(range(10))}

d = {100: "一百"}

d = {(1970,1,1): "计算机元年"}

# d = {[1970,1,1]: '计算机元年'} # TypeError

d = {3.14: {"b": 100, "c": 200}}

d = {'一': "One", '二': 'Two', '三': 'Three'}

字典的构造函数 dict

dict() 生成一个空的字典,等同于 {}

dict(iterable) 用可迭代对象生成一个字典

dict(\*\*kwargs) 用关键字传参形式生成一个字典

示例:

d = dict()

L = [(1, 2), [3, 4], "AB"]

d = dict(L) # L中每个元素都是含有两个元素的序列

d = dict(name="tarena", age=15)

字典的键(key) 必须为不可变类型

bool, int, float, complex, str, tuple

(以下后面才学)

frozenset, bytes

可变类型不能作为字典的键

list, dict,

set, bytearray(后面才学)

字典的基本操作

字典的键索引:

用[] 运算符可以获取字典的'键' 对应的 '值'

取值语法:

v = 字典[键]

示例:

d = {'name': 'tarena', 'age':20}

print(d['name'], '今年', d['age'], '岁')

赋值的语法:

字典[键] = 表达式

说明:

键不存在,创建键,并绑定键对应的值

键存在,修改绑的绑定关系

示例:

d = {}

d['name'] = "小张" # 创建'name'键 同时绑定'小张'

d['age'] = 20

d['age'] = 21 # 修改'age'这个键绑定的值

删除字典的键值对

del 语句

语法:

del 字典[键]

作用:

删除字典内的键,同时解除对值的绑定

示例:

d = {"a": "一", 'b':'二', 3:333}

in / not in 运算符

成员资格运算

作用:

判断一个键是否存在于字典中.如果存在则返回True, 否

则返回False

说明:

not in 返回结果与 in 相反

示例:

d = {'name': "小张", 'age':15}

'name' in d # True

'小张' in d # False

练习:

写程序,实现以下要求:

1) 将如下数据形成一个字典 seasons

键 值

1 '春季有1,2,3月'

2 '夏季有4,5,6月'

3 '秋季有7,8,9月'

4 '冬季有10,11,12月'

2) 让用户输入一个整数代表这个季度,打印这个季度的信息

如果用户输入的信息不在字典内,则打印"信息不存在"

seasons.py

# 方法1

# L = [

# (1, '春季有1,2,3月'),

# (2, '夏季有4,5,6月'),

# (3, '秋季有7,8,9月'),

# (4, '冬季有10,11,12月')

# ]

# seasons = dict(L)

# 方法2

# seasons = {}

# seasons[1] = '春季有1,2,3月'

# seasons[2] = '夏季有4,5,6月'

# seasons[3] = '秋季有7,8,9月'

# seasons[4] = '冬季有10,11,12月'

# 方法3

seasons = {

1: '春季有1,2,3月',

2: '夏季有4,5,6月',

3: '秋季有7,8,9月',

4: '冬季有10,11,12月'

}

# print(seasons)

# 2) 让用户输入一个整数代表这个季度,打印这个季度的信息

# 如果用户输入的信息不在字典内,则打印"信息不存在"

n = int(input("请输入季度: "))

if n in seasons:

print(seasons[n])

else:

print("信息不存在")

字典的迭代访问:

字典是可迭代对象,字典只能对键进行迭代访问

示例:

d = {1:"一元", 2:"二元", 5:"五元", 10:"十元"}

for x in d:

print(x) # 1 2 5 10

可以用于字典的内建函数

len(x) 返回字典的键-值对的个数

max(x) 返回字典中键的最大值

min(x) 返回字典中键的最小值

sum(x) 返回字典中所有键的和

any(x) 真值测试,如果字典中其中一个键为真值则返回True

all(x) 真值测试,所有键为真值才返回True

示例:

d = {0:'零', 5:'伍', 8:'捌', 3:'叁'}

print(len(d)) # 4

print(max(d)) # 8

print(min(d)) # 0

print(sum(d)) # 16

print(any(d)) # True

print(all(d)) # False

字典的方法

文档参见:

python\_base\_docs\_html/dict.html

示例:

d = {1: "One", 2: "two"}

d2 = {2:"二", 3:"三"}

d.update(d2) # d = {1: 'One', 2: '二', 3: '三'}

print(d[5]) # KeyError

print(d.get(5)) # None

print(d.get(5, "不存在")) # 不存在

print(d.get(2, "不存在)) # 二

d2 = {0:'零', 5:'伍', 8:'捌', 3:'叁'}

for x in d2.keys():

print(x) # 0, 5, 8, 3

for x in d2.values():

print(x) # '零', '捌', '叁', '伍'

for t in d2.items():

print(t) # (0, '零') (8, '捌') (3, '叁') (5, '伍')

for k, v in d2.items():

print("键:", k, '的值是', v)

练习:

输入一段字符串,打印出这个字符串中出现过的字符及出现的次数

如:

输入: ABCDABCABA

输出:

A: 4次

B: 3次

D: 1次

C: 2次

注: 不要求打印的顺序

字典推导式

字典推导式是用可迭代对象生成字典的表达式

语法:

{键表达式: 值表达式 for 变量 in 可迭代对象

[if 真值表达式] }

注: [] 的内容代表可省略

示例:

# 生成一个字典,键为数字 1~9, 值为键的平方

# {1:1, 2:4, 3:9, .... 9:81}

d = {x : x\*\*2 for x in range(1, 10)}

练习:

1. 有如下字符串列表

L = ['tarena', 'xiaozhang', 'hello']

生成如下字典:

d = {'tarena': 6, 'xiaozhang': 9, 'hello':5}

注: 字典的值为键的长度

答案:

d = {x : len(x) for x in L}

2. 已知有两个等长度的列表list1和lists2, 生成相应字典

list1 = [1001, 1002, 1005, 1008]

list2 = ['Tom', 'Jerry', 'Spike', 'Tyke']

生成的字典为:

{'Tom':1001, 'Jerry': 1002, ....}

字典推导式嵌套

语法等同于列表推导式的语法

字典 VS 列表

1. 都是可变对象

2. 索引方式不同,列表用整数索引,字典用键索引，索引是键经过哈希函数处理后得到的。哈希函数的目的是使键均匀地分布在数组中。

3. 字典的插入,删除,修改的速度可能会快于列表(重要)

4. 列表的存储是有序的,字典的存储是无序的

示例:

L = list(range(10))

for x in L:

L.remove(x)

请问最终 L的结果是什么?

结果: [1, 3, 5, 7, 9]

练习:

1. 打印九九乘法表:

1x1=1

1x2=2 2x2=4

1x3=3 2x3=6 3x3=9

....

.... 9x9=81

2. 输入一个数,代表图形的宽度, 打印如下正方形

如:

输入: 5

打印:

1 2 3 4 5

2 3 4 5 6

3 4 5 6 7

4 5 6 7 8

5 6 7 8 9

如:

输入: 3

打印:

1 2 3

2 3 4

3 4 5

3. 完全数:

1 + 2 + 3 = 6 (6为完全数)

1,2,3都为6的因数(能被一个数x整数的数为y,则y为x的因数)

1 x 6 = 6

2 x 3 = 6

完全数是指除自身以外,所有的因数相加之和等于自身的数

求 4 个完全数,并打印结果

答案:

6

28

496

...

4. 输入任意个学生的姓名,年龄和成绩,每个学生的信息存入

字典,然后放入到列表中.

要求: 每个学生的信息需要手动输入,姓名直接回车,结束输入

如:

请输入姓名: tarena

请输入年龄: 15

请输入成绩: 99

请输入姓名: name2

请输入年龄: 20

请输入成绩: 88

请输入姓名: <回车> # 结束输入

存内部存储格式如下:

[{'name':'tarena', 'age':15, 'score':99},

{'name':'name2', 'age':20, 'score':88}]

然后再用上面的列表,打印学生信息(形成表格)

+---------------+----------+----------+

| 姓名 | 年龄 | 成绩 |

+---------------+----------+----------+

| tarena | 15 | 99 |

| name2 | 20 | 88 |

...........

+---------------+----------+----------+

day08笔记:

集合 set

集合是可变的容器

集合内的数据对象都是唯一的(不能重复多次的)

集合是无序的存储结构,集合中的数据没有先后顺序关系

集合内的元素必须是不可变对象

集合是可迭代对象

集合是相当于只有键没有值的字典(键则是集合的数据)

创建空集合

set()

创建非空集合的字面值:

s = {1, 2, 3}

s = {"hello", "china"}

集合的构造(创建)函数 set

set() 创建一个空的集合对象(不能用{} 来创建空集合)

set(iterable) 用可迭代对象创建一个新的集合对象

示例:

s = set(range(10)) # {0, 1, 2, 3, 4...., 9}

s = set("ABCD")

s = set("ABCCBA") # s ={'A', 'B', 'C'}

s = set({1:'One', 5:'five'}) # {1, 5}

s = set([1, 0, 3.14, 0.618])

s = {True, None, "ABC", (1, 2, 3)}

集合的运算:

交集&, 并集|, 补集-, 对象补集^, 子集< 超集>

& 生成两个集合的交集

s1 = {1, 2, 3}

s2 = {2, 3, 4}

s3 = s1 & s2 # s3 = {2, 3}

| 生成两个集合的并集

s1 = {1, 2, 3}

s2 = {2, 3, 4}

s3 = s1 | s2 # s3 = {1, 2, 3, 4}

- 生成两个集合的补集

s1 = {1, 2, 3}

s2 = {2, 3, 4}

s1 - s2 # {1}, 生成属于s1,但不属于s2的所有元素的集合

^ 生成两个集合的对称补集

s1 = {1, 2, 3}

s2 = {2, 3, 4}

s3 = s1 ^ s2 # {1, 4} # 等同于(s1-s2 | s2-s1)

< 判断一个集合是另一个集合的子集

> 判断一个集合是另一个集合的超集

s1 = {1, 2, 3}

s2 = {2, 3}

s1 > s2 # True

s2 < s1 # True

== != 集合相同或不同

s1 = {1, 2, 3}

s2 = {3, 2, 1}

s1 == s2 # True

s1 != s2 # False

<= >= 子集或相同,超集或相同

示例略

in , not in 运算符

等同于字典, in 运算符用于集合中时,判断某个元素是否存在于

集合中,如果是则返回True,否则返回False

用于集合的函数:

len(x), max(x), min(x), sum(x), any(x), all(x)

集合是可迭代对象

集合练习:

经理有: 曹操, 刘备, 孙权

技术员有: 曹操,刘备,张飞,关羽

用集合求:

1. 即是经理也是技术员的有谁?

2. 是经理,但不是技术人员的有谁?

3. 是技术人员,但不是经理的人都有谁?

4. 张飞是经理吗?

5. 身兼一职的人都有谁?

6. 经理和技术员共有几个人?

集合的方法:

文档参见:

python\_base\_docs\_html/set.html

集合推导式

集合推导式是用可迭代对象创建集合的表达式

语法:

{表达式 for 变量 in 可迭代对象 [if 真值表达式]}

注[] 代表其中的内容可省略

示例:

numbers = [1, 3, 5, 3, 5, 1]

s = {x\*\*2 for x in numbers} # s = {1, 9, 25}

集合推导式嵌套

嵌套规则同列表推导式嵌套

固定集合 frozenset

固定集合是不可变的,无序的,含有唯一元素的集合

作用:

固定集合可以作为字典的键,也可以作为集合的值

固定集合的构造函数 frozenset

frozenset() 创建一个空的固定集合

frozenset(iterable) 用可迭代对象创建一个新的固定集合

示例:

fz = frozenset("hello") # frozenset({'l', 'o', 'e', 'h'})

固定集合的运算:

& 交集 ---> &=

| 并集 ---> |=

- 补集 ---> -=

^ 对称补集 ---> ^=

in / not in

> >= < <= == !=

(以上运算规则等同于集合set中的运算规则)

固定集合的方法:

相当于集合的全部方法去掉修改集合的方法

阶段总结

数据类型:

不可变数据类型:

bool, int, float, complex

str, tuple, frozenset, bytes(字节串,后面才学)

可变数据类型:

list dict, set, bytearray(字节数组,后面才学)

值:

None, False, True

运算符:

+ - \* / // % \*\*

< <= > >= == !=

is, is not # 验证id

in, not in # 判断一个值是否在容器内

not and or # 布尔运算

& | ^ # 集合运算(位运算符)

+(正号) - (负号)

表达式:

1

1 + 2

sum([1, 2, 3]) + sum([4, 5, 6])

print("hello")

条件表达式 x if x > y else y

全部的推导式: [x for x in range(...)]

列表,字典,集合推导式

lambda (返回一个函数, 后面才学)

语句

表达式语句

input("请输入字符串") # 函数调用是表达式,

"abcd"

赋值语句

变量名(标识符) = 表达式

a = 100

a = b = c = 200

x, y, z = 100, 200, 300

del 语句

删除变量,可能会释放对象(引用计数)

if 语句

pass 语句

while 语句

for 语句

break 语句

continue 语句

内建函数

len(x)

max(x)

min(x)

sum(x)

any(x)

all(x)

------ 构造(创建)函数 ----

bool(x)

int(x)

float(x)

complex(x)

str(x)

list(x)

tuple(x)

dict(x)

set(x)

frozenset(x)

------- 数值相关的函数-------

abs(x)

round(x)

pow(x, y, z=None)

------- 字符串相关的函数----

bin(x)

oct(x)

hex(x)

chr(x)

ord(x)

------- 可迭代对象相关的函数---

range(start, stop, step)

reversed(x)

------- 输入输出函数---------

input(x)

print(...)

-------- 对象和类型---------

type(x)

id(x)

--------------

sorted(x)

详见:

>>> help(\_\_builtins\_\_)

函数 function

什么是函数

函数是可以重复执行的语句块,可以重复调用

函数是面向过程编程的最小单位

作用:

1. 用于封装语句块,提高代码的重用性

2. 定义用户级别的函数

def 语句

语法:

def 函数名(形参列表):

语句块(代码块)

说明:

1. 函数的名字是语句块的名称

2. 函数名的命名规则与变量名相同(函数名必须为标识符)

3. 函数名是一个变量(不要轻易对其赋值)

4. 函数有自己的名字空间,在函数外部不可以访问函数内部的

变量,在函数内部可以访问函数外部的变量(取值)

5. 函数如果不需要传入参数,则参数列表可以为空

6. 语句部分不能为空,如果为空需要填充pass语句

示例见:

def.py

def2.py

函数调用

语法:

函数名(实际调用传递参数)

说明:

函数调用是一个表达式

如果没有return语句,函数执行完毕后返回None对象

练习

1. 写一个函数myadd, 此函数中的参数列表里有两个参数x, y

此函数打印两个参数的和(即, x + y 的和)

def myadd(...):

....

myadd(100, 200) # 打印 300

myadd("ABC", "123") # 打印 ABC123

2. 写一个函数print\_even,传入一个参数n代表终止整数

此函数打印 2 4 6 .... n(不包含n)以内的所有偶数

函数定义如下:

def print\_even(n):

..... # 此处自己实现

print\_even(10)

打印

2

4

6

8

return 语句

语法:

return [表达式]

注: []代表可以省略

作用:

用于函数中,结束当前函数的执行,返回到调用该函数的地方,同

时返回一个对象的引用关系

说明:

return 语句后跟的表达式可以省略,省略后相当于return None

如果函数内没有return语句,则函数执行完最后一条语句后

返回None(相当于在最后加了一条return None语句)

示例见:

return.py

练习:

1. 写一个函数mymax(实现返回两个数的最大值)

如:

def mymax(a, b):

....

print(mymax(100, 200)) # 200

print(mymax("abc", 'ABCD')) # abc

2. 写一个函数 mysum, 实现给出两个数,返回这两个数的和

def mysum(x, y):

...

a = int(input("请输入第一个数: "))

b = int(input("请输入第二个数: "))

print("您输入的两个数的和是:", mysum(a, b))

3. 写一个函数input\_number

def input\_number():

....

函数用来获取用户循环输入的整数,当用户输入负数时结束输入

将用户输入的数字以列表的形式返回,再用内建函数max,min

sum等求出用户输入的最大值,最小值及和

L = input\_number()

print(L)

print("用户输入的最大值是:", max(L))

print("用户输入的最小值是:", min(L))

print("用户输入的数据之和是:", sum(L))

练习:

1. 定义两个函数:

sum3(a, b, c) 用于返回三个数的和

pow3(x) 用于返回x的三次方(立方)

用以上函数计算:

1) 计算 1的立方+2的立方+3的立方的和

2) 计算 1+2+3的和的立方

即: 1\*\*3+2\*\*3+3\*\*3 和 (1+2+3)\*\*3

2. 写一个函数 get\_chinese\_char\_count(s) , 此函数返回

一个字符串s中所有中文字符的个数

def get\_chinese\_char\_count(s):

...

s = input("请输入中英文混合的字符串: ")

print("您输入的中文字符的个数是:",

get\_chinese\_char\_count(x))

注:

中文编码范围是: 0x4E00 ~ 0x9FA5(包含)

3. 修改学生信息管理程序,将其修改为两个函数实现

函数1:

def input\_student():

.... # 此函数返回输入学生信息的列表(格式不变)

函数2:

def output\_student(L):

... 此处以表格式形式打印学生信息(表格样式不变)

测试函数:

infos = input\_student()

print(infos) # 打印字典组成的列表

output\_student(infos) # 打印表格

保证修改后原功能不变