# 10.Python标准库

## 10.1.time库

Python自身提供了比较丰富的生态，拿来即用，可极大的提高开发效率

time库为Python处理时间的标准库

**1、获取现在时间**

（1）time.localtime() 本地时间

（2）time.gmtime() UTC世界统一时间

北京时间比时间统一时间UTC早8个小时

**import** **time**

t\_local = time.localtime()

t\_UTC = time.gmtime()

print("t\_local", t\_local) *# 本地时间*

print("t\_UTC", t\_UTC) *# UTC统一时间*

t\_local time.struct\_time(tm\_year=2019, tm\_mon=8, tm\_mday=29, tm\_hour=16, tm\_min=43, tm\_sec=37, tm\_wday=3, tm\_yday=241, tm\_isdst=0)

t\_UTC time.struct\_time(tm\_year=2019, tm\_mon=8, tm\_mday=29, tm\_hour=8, tm\_min=43, tm\_sec=37, tm\_wday=3, tm\_yday=241, tm\_isdst=0)

time.ctime() *# 返回本地时间的字符串*

'Thu Aug 29 16:44:52 2019'

**2、时间戳与计时器**

（1）time.time()   返回自1970年以来的秒数，记录sleep

（2）time.perf\_counter()   随意选取一个时间点，记录现在时间到该时间点的间隔秒数，记录sleep

（3）time.process\_time()   随意选取一个时间点，记录现在时间到该时间点的间隔秒数，不记录sleep

t\_1\_start = time.time()

t\_2\_start = time.perf\_counter()

t\_3\_start = time.process\_time()

print(t\_1\_start)

print(t\_2\_start)

print(t\_3\_start)

res = 0

**for** i **in** range(1000000):

res += i

time.sleep(5)

t\_1\_end = time.time()

t\_2\_end = time.perf\_counter()

t\_3\_end = time.process\_time()

print("time方法：**{:.3f}**秒".format(t\_1\_end-t\_1\_start))

print("perf\_counter方法：**{:.3f}**秒".format(t\_2\_end-t\_2\_start))

print("process\_time方法：**{:.3f}**秒".format(t\_3\_end-t\_3\_start))

1567068710.7269545

6009.0814064

2.25

time方法：5.128秒

perf\_counter方法：5.128秒

process\_time方法：0.125秒

**3、格式化**

time.strftime 自定义格式化输出

lctime = time.localtime()

time.strftime("%Y-%m-**%d** %A %H:%M:%S", lctime)

'2019-08-29 Thursday 16:54:35'

**4、睡觉觉**

time.sleep()

## 10.2.random库

随机数在计算机应用中十分常见

Python通过random库提供各种伪随机数

基本可以用于绝大多数工程应用

**1、随机种子——seed(a=None)**

（1）相同种子会产生相同的随机数

（2）如果不设置随机种子，以系统当前时间为默认值

**from** **random** **import** \*

seed(10)

print(random())

seed(10)

print(random())

0.5714025946899135

0.5714025946899135

print(random())

0.20609823213950174

**2、产生随机整数**

（1）randint(a, b)——产生[a, b]之间的随机整数

numbers = [randint(1,10) **for** i **in** range(10)]

numbers

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

（2）randrange(a)——产生[0, a)之间的随机整数

numbers = [randrange(10) **for** i **in** range(10)]

numbers

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

（3）randrange(a, b, step)——产生[a, b)之间以setp为步长的随机整数

numbers = [randrange(0, 10, 2) **for** i **in** range(10)]

numbers

[2, 6, 8, 4, 8, 2, 0, 0, 6, 2]

**3、产生随机浮点数**

（1）random()——产生[0.0, 1.0)之间的随机浮点数

numbers = [random() **for** i **in** range(5)]

numbers

[0.9819392547566425,

0.14351321471670841,

0.5218569500629634,

0.8648825892767497,

0.26702706855337954]

（2）uniform(a, b)——产生[a, b]之间的随机浮点数

numbers = [uniform(2.1, 3.5) **for** i **in** range(5)]

numbers

[2.523598043850906,

3.0245903649048116,

2.413131937436849,

2.8627907782614415,

2.16114212173462]

**4、序列用函数**

（1）choice(seq)——从序列类型中随机返回一个元素

choice(['win', 'lose', 'draw'])

'draw'

choice("python")

'h'

（2）choices(seq,weights=None, k)——对序列类型进行k次重复采样，可设置权重

choices(['win', 'lose', 'draw'], k=5)

['draw', 'lose', 'draw', 'draw', 'draw']

choices(['win', 'lose', 'draw'], [4,4,2], k=10)

['lose', 'draw', 'lose', 'win', 'draw', 'lose', 'draw', 'win', 'win', 'lose']

（3）shuffle(seq)——将序列类型中元素随机排列，返回打乱后的序列

numbers = ["one", "two", "three", "four"]

shuffle(numbers)

numbers

['four', 'one', 'three', 'two']

（4）sample(pop, k)——从pop类型中随机选取k个元素，以列表类型返回

sample([10, 20, 30, 40, 50], k=3)

[20, 30, 10]

5、概率分布——以高斯分布为例

gauss(mean, std)——生产一个符合高斯分布的随机数

number = gauss(0, 1)

number

0.6331522345532208

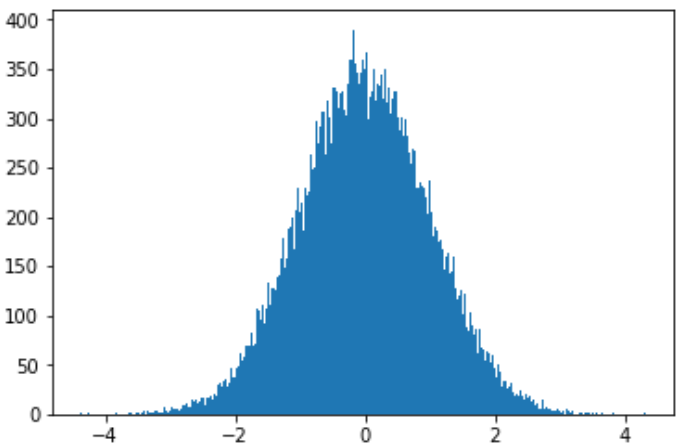
多生成几个

**import** **matplotlib.pyplot** **as** **plt**

res = [gauss(0, 1) **for** i **in** range(100000)]

plt.hist(res, bins=1000)

plt.show()



【例1】用random库实现简单的微信红包分配

**import** **random**

**def** red\_packet(total, num):

**for** i **in** range(1, num):

per = random.uniform(0.01, total/(num-i+1)\*2) *# 保证每个人获得红包的期望是total/num*

total = total - per

print("第**{}**位红包金额： **{:.2f}**元".format(i, per))

**else**:

print("第**{}**位红包金额： **{:.2f}**元".format(num, total))

red\_packet(10, 5)

第1位红包金额： 1.85元

第2位红包金额： 3.90元

第3位红包金额： 0.41元

第4位红包金额： 3.30元

第5位红包金额： 0.54元

每个人获得红包的期望值

**import** **random**

**import** **numpy** **as** **np**

**def** red\_packet(total, num):

ls = []

**for** i **in** range(1, num):

per = round(random.uniform(0.01, total/(num-i+1)\*2), 2) *# 保证每人红包的期望是total/num*

ls.append(per)

total = total - per

**else**:

ls.append(total)

**return** ls

*# 重复发十万次红包，统计每个位置的平均值（约等于期望）*

res = []

**for** i **in** range(100000):

ls = red\_packet(10,5)

res.append(ls)

res = np.array(res)

print(res[:5])

np.mean(res, axis=0)

[[1.71 1.57 0.36 1.25 5.11]

[1.96 0.85 1.46 3.29 2.44]

[3.34 0.27 1.9 0.64 3.85]

[3.16 1.2 0.3 3.66 1.68]

[2.43 0.16 0.11 0.79 6.51]]

array([1.9991849, 2.0055725, 2.0018144, 2.0022472, 1.991181 ])

【例2】生产4位由数字和英文字母构成的验证码

**import** **random**

**import** **string**

print(string.digits)

print(string.ascii\_letters)

s=string.digits + string.ascii\_letters

v=random.sample(s,4)

print(v)

print(''.join(v))

0123456789

abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ

['n', 'Q', '4', '7']

nQ47

## 10.3.collections库——容器数据类型

**1、namedtuple——具名元组**

点的坐标，仅看数据，很难知道表达的是一个点的坐标

**import** **collections**

p = (1, 2)

构建一个新的元组子类

定义方法如下：typename 是元组名字，field\_names 是字段名

collections.namedtuple(typename, field\_names, \*, rename=**False**, defaults=**None**, module=**None**)

Point = collections.namedtuple("Point", ["x", "y"])

p = Point(1, y=2)

p

Point(x=1, y=2)

可以调用属性

print(p.x)

print(p.y)

1

2

有元组的性质

print(p[0])

print(p[1])

x, y = p

print(x)

print(y)

1

2

1

2

确实是元组的子类

print(isinstance(p, tuple))

True

【例】模拟扑克牌

Card = collections.namedtuple("Card", ["rank", "suit"])

ranks = [str(n) **for** n **in** range(2, 11)] + list("JQKA")

suits = "spades diamonds clubs hearts".split()

print("ranks", ranks)

print("suits", suits)

cards = [Card(rank, suit) **for** rank **in** ranks

**for** suit **in** suits]

cards

ranks ['2', '3', '4', '5', '6', '7', '8', '9', '10', 'J', 'Q', 'K', 'A']

suits ['spades', 'diamonds', 'clubs', 'hearts']

[Card(rank='2', suit='spades'),

Card(rank='2', suit='diamonds'),

...

Card(rank='A', suit='hearts')]

**from** **random** **import** \*

*# 洗牌*

shuffle(cards)

cards

[Card(rank='J', suit='hearts'),

Card(rank='A', suit='hearts'),

...

Card(rank='9', suit='diamonds')]

*# 随机抽一张牌*

choice(cards)

Card(rank='4', suit='hearts')

*# 随机抽多张牌*

sample(cards, k=5)

[Card(rank='4', suit='hearts'),

Card(rank='2', suit='clubs'),

Card(rank='Q', suit='diamonds'),

Card(rank='9', suit='spades'),

Card(rank='10', suit='hearts')]

**2、Counter——计数器工具**

**from** **collections** **import** Counter

s = "牛奶奶找刘奶奶买牛奶"

colors = ['red', 'blue', 'red', 'green', 'blue', 'blue']

cnt\_str = Counter(s)

cnt\_color = Counter(colors)

print(cnt\_str)

print(cnt\_color)

Counter({'奶': 5, '牛': 2, '找': 1, '刘': 1, '买': 1})

Counter({'blue': 3, 'red': 2, 'green': 1})

是字典的一个子类

print(isinstance(Counter(), dict))

True

最常见的统计——most\_commom(n)  
 提供 n 个频率最高的元素和计数

cnt\_color.most\_common(2)

[('blue', 3), ('red', 2)]

元素展开——elements()

list(cnt\_str.elements())

['牛', '牛', '奶', '奶', '奶', '奶', '奶', '找', '刘', '买']

其他一些加减操作

c = Counter(a=3, b=1)

d = Counter(a=1, b=2)

c+d

Counter({'a': 4, 'b': 3})

【例】从一副牌中抽取10张，大于10的比例有多少

cards = collections.Counter(tens=16, low\_cards=36)

seen = sample(list(cards.elements()), k=10)

print(seen)

['tens', 'low\_cards', 'low\_cards', 'low\_cards', 'tens', 'tens', 'low\_cards', 'low\_cards', 'low\_cards', 'low\_cards']

seen.count('tens') / 10

0.3

3、deque——双向队列

列表访问数据非常快速

插入和删除操作非常慢——通过移动元素位置来实现

特别是 insert(0, v) 和 pop(0)，在列表开始进行的插入和删除操作

双向队列可以方便的在队列两边高效、快速的增加和删除元素

**from** **collections** **import** deque

d = deque('cde')

d

deque(['c', 'd', 'e'])

d.append("f") *# 右端增加*

d.append("g")

d.appendleft("b") *# 左端增加*

d.appendleft("a")

d

deque(['a', 'b', 'c', 'd', 'e', 'f', 'g'])

d.pop() *# 右端删除*

d.popleft() *# 左端删除*

d

deque(['b', 'c', 'd', 'e', 'f'])

## 10.4.itertools库——迭代器

1、排列组合迭代器

（1）product——笛卡尔积

**import** **itertools**

**for** i **in** itertools.product('ABC', '01'):

print(i)

('A', '0')

('A', '1')

('B', '0')

('B', '1')

('C', '0')

('C', '1')

**for** i **in** itertools.product('ABC', repeat=3): *# 求3个ABC的笛卡尔积*

print(i)

('A', 'A', 'A')

('A', 'A', 'B')

...

('C', 'C', 'C')

（2） permutations——排列

**for** i **in** itertools.permutations('ABCD', 3): *# 3 是排列的长度*

print(i)

('A', 'B', 'C')

('A', 'B', 'D')

...

('D', 'C', 'B')

**for** i **in** itertools.permutations(range(3)):

print(i)

(0, 1, 2)

(0, 2, 1)

(1, 0, 2)

(1, 2, 0)

(2, 0, 1)

(2, 1, 0)

（3）combinations——组合

**for** i **in** itertools.combinations('ABCD', 2): *# 2是组合的长度*

print(i)

('A', 'B')

('A', 'C')

('A', 'D')

('B', 'C')

('B', 'D')

('C', 'D')

**for** i **in** itertools.combinations(range(4), 3):

print(i)

(0, 1, 2)

(0, 1, 3)

(0, 2, 3)

(1, 2, 3)

（4）combinations\_with\_replacement——元素可重复组合

**for** i **in** itertools.combinations\_with\_replacement('ABC', 2): *# 2是组合的长度*

print(i)

('A', 'A')

('A', 'B')

('A', 'C')

('B', 'B')

('B', 'C')

('C', 'C')

**for** i **in** itertools.product('ABC',repeat=2):

print(i)

('A', 'A')

('A', 'B')

('A', 'C')

('B', 'A')

('B', 'B')

('B', 'C')

('C', 'A')

('C', 'B')

('C', 'C')

**2、拉链**

（1）zip——短拉链

**for** i **in** zip("ABC", "012", "xyz"):

print(i)

('A', '0', 'x')

('B', '1', 'y')

('C', '2', 'z')

长度不一时，执行到最短的对象处，就停止

**for** i **in** zip("ABC", [0, 1, 2, 3, 4, 5]): *# 注意zip是内置的，不需要加itertools*

print(i)

('A', 0)

('B', 1)

('C', 2)

（2）zip\_longest——长拉链

长度不一时，执行到最长的对象处，就停止，缺省元素用None或指定字符替代

**for** i **in** itertools.zip\_longest("ABC", "012345"):

print(i)

('A', '0')

('B', '1')

('C', '2')

(None, '3')

(None, '4')

(None, '5')

**for** i **in** itertools.zip\_longest("ABC", "012345", fillvalue = "?"):

print(i)

('A', '0')

('B', '1')

('C', '2')

('?', '3')

('?', '4')

('?', '5')

3、其他

（1）enumerate(iterable, start=0)——枚举（Python内置）

产出由两个元素组成的元组，结构是（index, item）,其中index 从start开始，item从iterable中取

**for** i **in** enumerate("Python", start=1):

print(i)

(1, 'P')

(2, 'y')

(3, 't')

(4, 'h')

(5, 'o')

(6, 'n')

（2）groupby(iterable, key=None)——分组

创建一个迭代器，按照key指定的方式，返回 iterable 中连续的键和组

一般来说，要预先对数据进行排序

key为None默认把连续重复元素分组

**for** key, group **in** itertools.groupby('AAAABBBCCDAABBB'):

print(key, list(group))

A ['A', 'A', 'A', 'A']

B ['B', 'B', 'B']

C ['C', 'C']

D ['D']

A ['A', 'A']

B ['B', 'B', 'B']

animals = ["duck", "eagle", "rat", "giraffe", "bear", "bat", "dolphin", "shark", "lion"]

animals.sort(key=len)

print(animals)

['rat', 'bat', 'duck', 'bear', 'lion', 'eagle', 'shark', 'giraffe', 'dolphin']

**for** key, group **in** itertools.groupby(animals, key=len):

print(key, list(group))

3 ['rat', 'bat']

4 ['duck', 'bear', 'lion']

5 ['eagle', 'shark']

7 ['giraffe', 'dolphin']

animals = ["duck", "eagle", "rat", "giraffe", "bear", "bat", "dolphin", "shark", "lion"]

animals.sort(key=**lambda** x: x[0])

print(animals)

**for** key, group **in** itertools.groupby(animals, key=**lambda** x: x[0]):

print(key, list(group))

['bear', 'bat', 'duck', 'dolphin', 'eagle', 'giraffe', 'lion', 'rat', 'shark']

b ['bear', 'bat']

d ['duck', 'dolphin']

e ['eagle']

g ['giraffe']

l ['lion']

r ['rat']

s ['shark']

## 10.5.作业练习

[24点问题](https://www.cnblogs.com/WhyEngine/p/3520515.html)

24点游戏是一种使用扑克牌来进行的益智类游戏，游戏内容是：  
 从一副扑克牌中抽去大小王剩下52张，任意抽取4张牌，把牌面上的数（A代表 1）运用加、减、乘、除和括号进行运算得出24。  
 每张牌都必须使用一次，但不能重复使用。 有些组合有不同种算法，例如要用2， 4，6，12四张牌组合成24点，可以有如下几种组合方法：  
 2 + 4 + 6 + 12 = 24  
 4 × 6 ÷ 2 + 12 = 24  
 12 ÷ 4 × (6 + 2) = 24  
 当然，也有些组合算不出24，如1、1、1、1 和 6、7、8、8等组合.

答案：

**import** **operator**

**from** **itertools**

**import** product, permutations**import** **time**

**def** mydiv(n, d):

**return** n / d **if** d != 0 **else** 999999999

syms = [operator.add, operator.sub, operator.mul, mydiv]

op = {sym: ch **for** sym, ch **in** zip(syms, '+-\*/')}

**def** solve24(nums):

**for** x, y, z **in** product(syms, repeat=3):

**for** a, b, c, d **in** permutations(nums):

**if** round(x(y(a, b), z(c, d)), 5) == 24:

**return** f"(**{a}** **{op[y]}** **{b}**) **{op[x]}** (**{c}** **{op[z]}** **{d}**)"

**elif** round(x(a, y(b, z(c, d))), 5) == 24:

**return** f"**{a}** **{op[x]}** (**{b}** **{op[y]}** (**{c}** **{op[z]}** **{d}**))"

**elif** round(x(y(z(c,d), b), a), 5) == 24:

**return** f"((**{c}** **{op[z]}** **{d}**) **{op[y]}** **{b}**) **{op[x]}** **{a}**"

**elif** round(x(y(b, z(c, d)), a), 5) == 24:

**return** f"(**{b}** **{op[y]}** (**{c}** **{op[z]}** **{d}**)) **{op[x]}** **{a}** "

**return** "No Answer!"

start\_time = time.time()

**for** nums **in** [ [2, 2, 2, 9],

[7, 9, 2, 8],

[7, 7, 1, 2],

[10, 10, 4, 4],

[9, 9, 10, 6],

[5, 5, 1, 5],

[5, 5, 2, 10],

[4, 1, 5, 6],

[7, 3, 3, 7],

[8, 3, 8, 3]]:

print(f"solve24(**{nums}**) -> {solve24(nums)}")

end\_time = time.time()

print("All the time spended: **{}**s".format(end\_time - start\_time))

solve24([2, 2, 2, 9]) -> 2 + (2 \* (2 + 9))

solve24([7, 9, 2, 8]) -> ((7 + 9) \* 2) - 8

solve24([7, 7, 1, 2]) -> ((7 \* 7) - 1) / 2

solve24([10, 10, 4, 4]) -> ((10 \* 10) - 4) / 4

solve24([9, 9, 10, 6]) -> 9 + (9 \* (10 / 6))

solve24([5, 5, 1, 5]) -> 5 \* (5 - (1 / 5))

solve24([5, 5, 2, 10]) -> 5 \* (5 - (2 / 10))

solve24([4, 1, 5, 6]) -> 4 / (1 - (5 / 6))

solve24([7, 3, 3, 7]) -> 7 \* (3 + (3 / 7))

solve24([8, 3, 8, 3]) -> 8 / (3 - (8 / 3))

All the time spended: 0.022938013076782227s