

第六章 函数

主讲人：刘潇潇





请思考

如果你和小组成员要写一个纸牌游戏：1.你负责写发牌功能。
2.游戏玩家有4个，每轮游戏开局后你要发4次牌。你会怎么设计代码？写好发牌代码后，执行4次相同代码，你有什么反思？





什么是函数

函数是组织好的，可重复使用的，用来实现单一或相关联功能的代码段，它能够提高应用的模块性和代码的重复利用率。

简化
复用
可扩展



主要内容

通过本次学习，掌握如何自定义以及调用函数、变量的作用域、递归函数的应用以及常用函数的应用



函数的
创建和调用



变量的
作用域



递归函数和
匿名函数



常用函数



函数的创建和调用



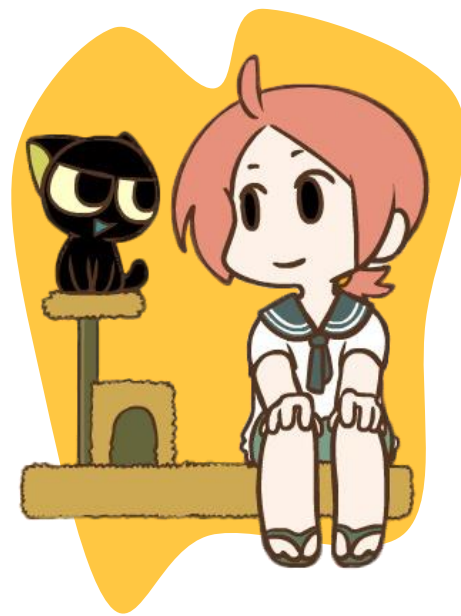
定义函数



调用函数



函数的返回值



函数的参数



函数的定义

Python 定义函数使用 def 关键字，格式如下：

```
def 函数名 ( ) :  
    函数体
```



函数的定义

#定义函数`init_cards`, 用于生成52张牌

```
def init_cards():  
    red_blacks = ['♥', '♣', '♦', '♠']  
    values = ['A', 'J', 'Q', 'K', '2', '3', '4', '5', '6', '7', '8', '9', '10']  
    # 生成52张牌  
    cards = []  
    for rb in red_blacks:  
        for value in values:  
            cards.append(rb + value)  
    print(cards)
```



函数的调用

定义了函数之后，想要让这些代码能够执行，需要调用函数。语法如下：

```
# 调用刚才定义的函数  
函数名()
```



函数的调用

#定义函数`init_cards`, 用于生成52张牌

`def init_cards():...`

函数定义好可以折叠, 来简化代码

#调用`init_cards`函数
`init_cards()`

调用函数

/Library/Frameworks/Python.framework/Versions/3.7/bin/python3.7 /Users/liuxiaoxiao/PycharmProjects/untitled3/test.py

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

Process finished with exit code 0



函数的 返回值

函数的返回值是使用return语句来完成的

```
def 函数名 ( ) :
```

```
    函数体
```

```
    return 返回值 #注意缩进
```



函数的 返回值

#定义函数`init_cards`, 用于生成52张牌

```
def init_cards():  
    red_blacks = ['♥', '♣', '♦', '♠']  
    values = ['A', 'J', 'Q', 'K', '2', '3', '4', '5', '6', '7', '8', '9', '10']  
    # 生成52张牌  
    cards = []  
    for rb in red_blacks:  
        for value in values:  
            cards.append(rb + value)  
    return cards
```

使用返回值, 可以把cards内容传到函数外部使用

#调用`init_cards`函数

```
new_cards = init_cards()  
print(new_cards)
```

通过调用函数, 获取到函数的返回值, 可以对其进行赋值、打印等操作

练习



写一个发牌函数
`deal_cards()`，实现给4
个玩家发牌的功能，在
函数体中。思考：你在
写本函数中遇到了什么
问题？



函数的 嵌套调用

```
def test_A():                # 4
    print('程序开始')        # 5
    print('程序执行中...')    # 6
def test_B():                # 2
    test_A()                  # 3
    print('程序结束')        # 7
test_B()                      # 1
```



函数的 嵌套调用

#定义函数`deal_cards`，实现发牌给4个玩家的功能

```
def deal_cards():  
    new_cards = init_cards()#调用init_cards()函数，获取52张牌的列表  
    for i in range(0,4): #循环4次，每次发牌给一个玩家  
        print(new_cards[random.randint(0,len(new_cards)-1)]) #通过随机指针生成随机纸牌
```



函数的 参数

如果希望让函数接收数据，这就是函数的参数。

```
def 函数名（参数列表）：  
    函数体
```



函数的参数

#定义函数`init_cards`, 用于生成52张牌

```
def init_cards(jokers):  
    red_blacks = ['♥', '♣', '♦', '♠']  
    values = ['A', 'J', 'Q', 'K', '2', '3', '4', '5', '6', '7', '8', '9', '10']  
    # 生成52张牌  
    cards = jokers  
    for rb in red_blacks:  
        for value in values:  
            cards.append(rb + value)  
    return cards
```

加入参数

在函数体中使用参数

```
jokers = ['red_joker', 'black_joker']
```

#调用`init_cards`函数

```
new_cards = init_cards(jokers)  
print(new_cards)
```

调用函数时必须传入对应的参数

练习



将joker的定义放在
deal_cards()调用之前，
将joker作为参数传入到
deal_cards()中，然后
调用deal_cards函数。



函数的 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。带有默认值的参数一定要位于参数列表的**最后面**。否则程序会报错。

```
def 函数名（参数列表=默认值）：  
    函数体
```



函数的 默认参数

```
def test_A(args_1, args_2 = 3):  
    result = args_1+args_2  
    return result  
print(test_A(1))
```



不定长参数

有时可能需要一个函数能处理比当初声明时更多的参数，这些参数叫做不定长参数，声明时不会命名。

```
def 函数名 (参数, *args, **kwargs) :  
    函数体
```

- 加了星号 (*) 的变量args会存放所有未命名的变量参数，args为元组；
- 加**的变量kwargs会存放命名参数，即形如key=value的参数，kwargs为字典。



不定长 参数

```
def test_A(args_1, *args):  
    result = args_1+1  
    print(args)
```

```
test_A(1,2,3)
```



练习



完成发牌程序，分别发
放给四个玩家。思考：
一轮游戏给四个玩家发
牌，但是我们有很多轮
游戏，是否应该把发牌
给四个玩家这个动作封
装成一个函数？





变量的作用域



作用域



局部变量



全局变量





作用域

- 为了避免变量的名称发生冲突，Python引入了命名空间的概念。
- 命名空间指的是名称到对象的映射，类似于字典，键名是变量的名字，值是变量的值。
- 命名空间是相互独立存在的，而且它们被安排在某个特定的层次，把这些层次结构组合起来就是作用域。



作用域

内置作用域

文件作用域

函数嵌套作用域

本地作用域

- 本地作用域
- 函数嵌套作用域
- 文件作用域
- 内置作用域





局部变量

- 所谓局部变量，就是在函数内部定义的变量。
- 局部变量的作用域是函数内部，意味着它只在定义它的函数中有效，一旦函数结束就会消失。





全局变量

- 定义在函数外的拥有全局作用域。
- 全局变量可以在整个程序范围内访问。
- 如果出现全局变量和局部变量名字相同的情况，则在函数中访问的是局部变量。



思考



在目前我们的代码中，
哪些是局部变量，哪些
是全局变量，他们的作
用域是什么？





递归函数和匿名函数



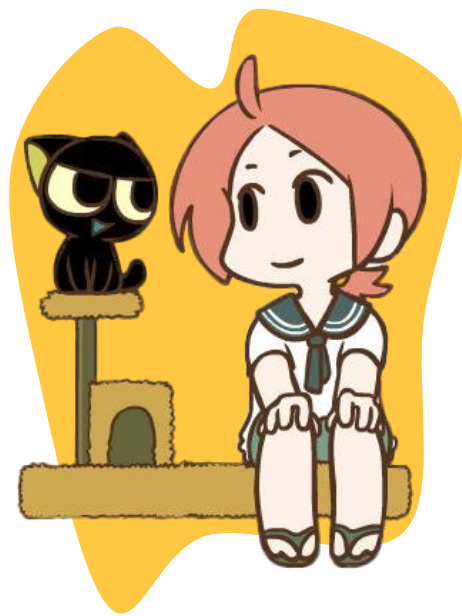
递归函数



匿名函数



内置函数：
Map、Reduce、
Filter



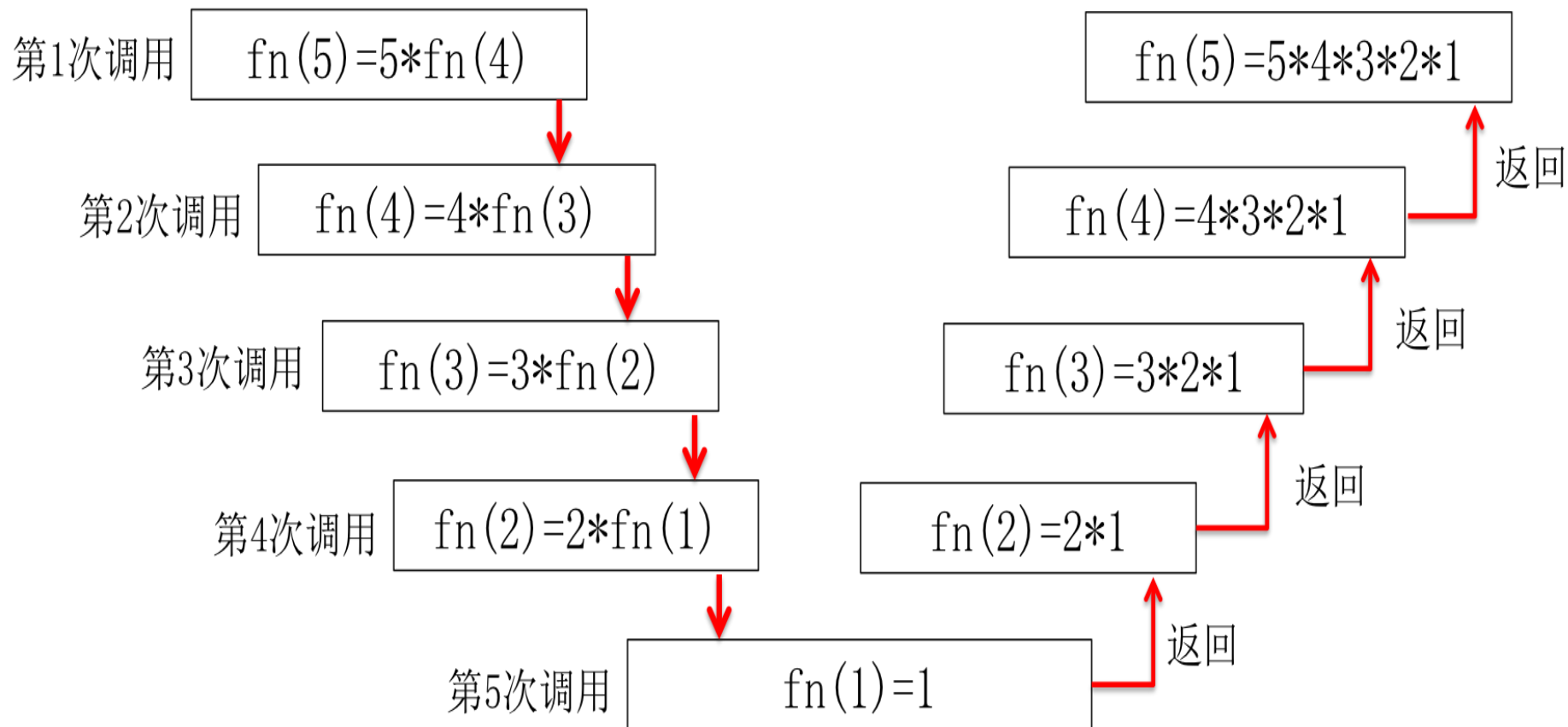


递归函数

- 一个函数的内部可以调用其他函数。但是，如果一个函数在内部调用自己本身的话，这个函数就是递归函数。



使用递归，实现阶乘 $n! = 1 * 2 * 3 * \dots * n$ 的计算。



使用递归函数，计算 $6! = 1*2*3*4*5*6$

```
def factorial(num):  
    if num == 1:  
        return num  
    else:  
        return num * factorial(num-1)  
  
print(factorial(6))
```

/Library/Frameworks/Python.framework/Versions/3.7/bin/python3.7 /Users/liuxiaoxiao/PyCha
720

Process finished with exit code 0



练习



使用递归函数，计算
100以内所有奇数的和





匿名函数

- 匿名函数就是没有名称的函数，也就是不再使用def语句定义的函数。如果要声明匿名函数，则需要使用lambda关键字，

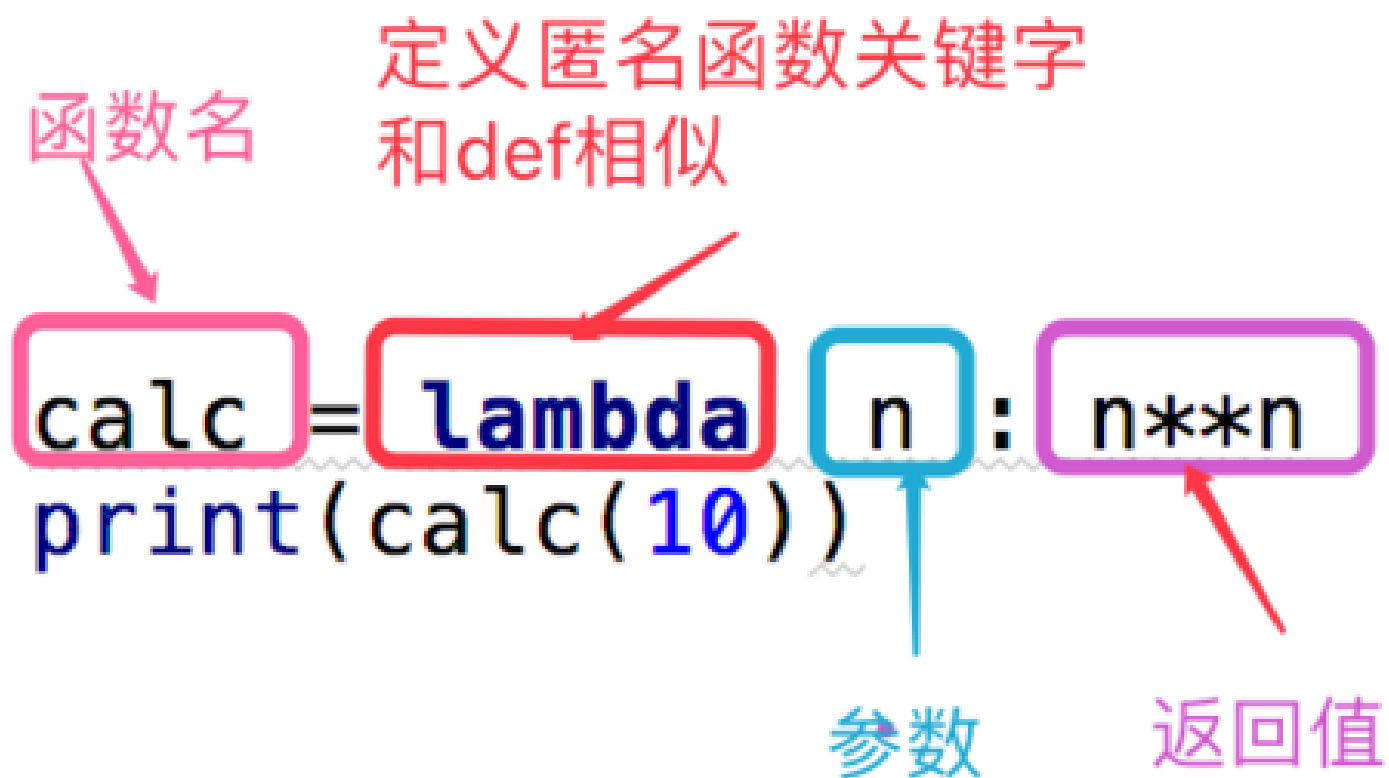


匿名函数的声明格式如下所示：

函数名 定义匿名函数关键字
 和def相似

```
calc = lambda n : n**n  
print(calc(10))
```

参数 返回值

The diagram illustrates the syntax of a lambda function. It shows the code 'calc = lambda n : n**n' and 'print(calc(10))'. Annotations include: '函数名' (Function Name) pointing to 'calc' with a pink box; '定义匿名函数关键字 和def相似' (Define anonymous function keyword, similar to def) pointing to 'lambda' with a red box; '参数' (Parameter) pointing to 'n' with a blue box; and '返回值' (Return Value) pointing to 'n**n' with a purple box. Each of these four elements is enclosed in a colored rounded rectangle matching its annotation's color.

看个栗子：

```
sum = lambda arg1, arg2: arg1 + arg2  
print(sum(10, 20))  
print(sum(20, 20 ))
```



map() 函数

map() 会根据提供的函数对指定序列做映射。
第一个参数 function 以参数序列中的每一个元素调用 function 函数，返回包含每次 function 函数返回值的新列表。

Map(函数，一个或多个序列)



map() 函数

#计算[1, 2, 3, 4, 5, 6, 7, 8, 9]每个元素的平方数

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

#方法一

```
def squares(x):
```

```
    squ_result = x*x
```

```
    return squ_result
```

```
result_list = []
```

```
for num in num_list:
```

```
    result_list.append(squares(num))
```

```
print(result_list)
```

#方法二

```
print(list(map(lambda num : num * num, num_list)))
```

test

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

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



filter() 函数

`filter()` 函数用于过滤序列，过滤掉不符合条件的元素，返回由符合条件元素组成的新列表。`Filter()` 函数接收两个参数，第一个为函数，第二个为序列，序列的每个元素作为参数传递给函数进行判断，然后返回 `True` 或 `False`，最后将返回 `True` 的元素放到新列表中。

`filter`(函数, 序列)



filter() 函数

#筛选出列表中所有的奇数

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

#方法一

```
def is_odd(num):  
    if num % 2 != 0:  
        return True  
    else:  
        return False
```

```
result_1 = filter(is_odd,num_lsit)
```

```
print(list(result_1))
```

#方法二

```
result_2 = filter(lambda num : num % 2 != 0,num_lsit)
```

```
print(list(result_2))
```

```
[1, 3, 5, 7, 9]
```

```
[1, 3, 5, 7, 9]
```



练习



使用`filter()`创建匿名函数，筛选出
`num_list=[15,24,14,9]`
中大于21的元素。



reduce() 函数

`reduce()` 函数会对参数序列中元素进行累积。
函数将一个数据集合（链表，元组等）中的所有数据进行下列操作：用传给 `reduce` 中的函数 `function`（有两个参数）先对集合中的第 1、2 个元素进行操作，得到的结果再与第三个数据用 `function` 函数运算，最后得到一个结果。

`reduce(函数, 序列)`



reduce() 函数

#计算列表元素的累加和

```
num_list = [1,2,3,4,5,6,7,8,9]
```

#方法一

```
def accumulation(num,result):
```

```
    return num + result
```

```
result_1 = reduce(accumulation,num_list)
```

```
print(result_1)
```

#方法二

```
num_list = [1,2,3,4,5,6,7,8,9]
```

```
result_2 = reduce(lambda num,num_list : num + num_list , num_list)
```

```
print(result_2)
```



练习



使用reduce函数，计算
数组cards= [3,5,7,11]
和cards= [3,5,7,1]的加
和。





注意

- 使用Lambda声明的匿名函数能接收任何数量的参数，但只能返回一个表达式的值。匿名函数不能直接调用print，因为lambda需要一个表达式。





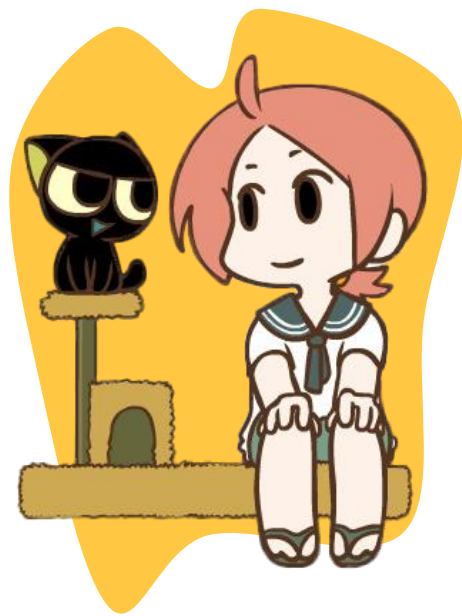
常用函数



时间日期
函数



随机数函数





时间函数

在Python中，通常有如下几种方式表示时间：

- (1) 时间戳；
- (2) 格式化的时间字符串；
- (3) 时间元组（struct_time）。



通常来讲，时间戳表示的是从1970年1月1日00:00:00开始按秒计算的偏移量。



```
import time; # 引入time模块
ticks = time.time()
print("当前时间戳为:", ticks)
```

```
t
/Library/Frameworks/Python.framework/Versions/3.7/bin/python3.7 /Users/l
当前时间戳为: 1572855586.637662
```

```
Process finished with exit code 0
```

时间戳





格式化的时间字符串

我们可以使用time模块的strftime方法来格式化日期。

```
import time
# 将当前时间格式化成2016-03-20 11:45:39形式
print(time.strftime("%Y-%m-%d %H:%M:%S", time.localtime()))
```

test

```
/Library/Frameworks/Python.framework/Versions/3.7/bin/python3.7 /Users/liuxiaoxiao/PycharmProjects/untitled4/test.py
2019-11-04 16:20:43
```

Process finished with exit code 0

格式化的时间字符串

格式化符号	含义
%y	两位数的年份表示 (00-99)
%Y	四位数的年份表示 (000-9999)
%m	月份 (01-12)
%d	月内中的一天

格式化的时间字符串

格式化符号	含义
%y	两位数的年份表示 (00-99)
%Y	四位数的年份表示 (000-9999)
%m	月份 (01-12)
%d	月内中的一天



时间元组

返回 struct_time 的函数主要有 gmtime()、localtime() 和 strptime(), struct_time 元组。

序号	字段	值
0	tm_year	2008
1	tm_mon	1到12
2	tm_mday	1到31
3	tm_hour	0到23

序号	字段	值
4	tm_min	0到59
5	tm_sec	0到61
6	tm_wday	0到6
7	tm_yday	1到366
8	tm_isdst	-1, 0, 1, -1是决定是否夏令时的旗帜



时间元组

`time.altzone`

返回格林威治西部的夏令时地区的偏移秒数。如果该地区在格林威治东部会返回负值（如西欧，包括英国）。对夏令时启用地区才能使用。



时间元组

`time.asctime([tupletime])`

接受时间元组并返回一个可读的形式为"Tue Dec 11 18:07:14 2008"（2008年12月11日 周二18时07分14秒）的24个字符的字符串。



时间元组

time.clock()

用以浮点数计算的秒数返回当前的CPU时间。用来衡量不同程序的耗时，比time.time()更有用。



时间元组

time.ctime([secs])

作用相当于asctime(localtime(secs))，未给参数相当于asctime()。



时间元组

`time.gmtime([secs])`

接收时间辮（1970纪元后经过的浮点秒数）并返回格林威治天文时间下的时间元组t。



时间元组

`time.localtime([secs])`

接收时间辏（1970纪元后经过的浮点秒数）并返回当地时间下的时间元组t（t.tm_isdst可取0或1，取决于当地当时是不是夏令时）。



时间元组

time.localtime([secs])

接收时间辏（1970纪元后经过的浮点秒数）并返回当地时间下的时间元组t。

time.mktime(tupletime)

接受时间元组并返回时间辏。



时间元组

time.sleep(secs)

推迟调用线程的运行，secs指秒数。

time.strftime(fmt[,tupletime])

接收以时间元组，并返回以可读字符串表示的当地时间，格式由fmt决定。



时间元组

`time.strptime(str,fmt='%a %b %d %H:%M:%S %Y')`

根据fmt的格式把一个时间字符串解析为时间元组。



时间元组

time.time()

返回当前时间的时间戳

time.tzset()

根据环境变量TZ重新初始化时间相关设置。



时间元组

time.timezone

表示当地时区（未启动夏令时）距离格林威治的偏移秒数（ >0 ，美洲； ≤ 0 大部分欧洲，亚洲，非洲）



时间元组

`time.tzname`

包含一对根据情况的不同而不同的字符串，分别是带夏令时的本地时区名称，和不带的。

日历函数

calendar.calendar(year,w=2,l=1,c=6)

返回一个多行字符串格式的year年年历，3个月一行，间隔距离为c。每日宽度间隔为w字符。每行长度为 $21 * W + 18 + 2 * C$ 。l是每星期行数。



日历函数

```
import calendar
print(calendar.calendar(2019))
```

est

```
/Library/Frameworks/Python.framework/Versions/3.7/bin/python3.7 /Users/liuxiaoxia
2019
```

January

Mo	Tu	We	Th	Fr	Sa	Su
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

February

Mo	Tu	We	Th	Fr	Sa	Su
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28			

March

Mo	Tu	We	Th	Fr	Sa	Su
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

April

Mo	Tu	We	Th	Fr	Sa	Su
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					

May

Mo	Tu	We	Th	Fr	Sa	Su
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

June

Mo	Tu	We	Th	Fr	Sa	Su
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

July

Mo	Tu	We	Th	Fr	Sa	Su
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

August

Mo	Tu	We	Th	Fr	Sa	Su
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

September

Mo	Tu	We	Th	Fr	Sa	Su
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30						

October

Mo	Tu	We	Th	Fr	Sa	Su
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

November

Mo	Tu	We	Th	Fr	Sa	Su
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	

December

Mo	Tu	We	Th	Fr	Sa	Su
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					



日历函数

- **calendar.isleap(year)**

如果是闰年返回True，否则为false。

```
import calendar  
print(calendar.isleap(2000))
```

test

```
/Library/Frameworks/Python.framework/Versions/3.7/bin/python3.  
True
```

```
Process finished with exit code 0
```



日历函数

- `calendar.leapdays(y1,y2)`

返回在Y1，Y2两年之间的闰年总数。

- `calendar.month(year,month,w=2,l=1)`

返回一个多行字符串格式的year年month月日历，两行标题，一周一行。每日宽度间隔为w字符。

每行的长度为 $7 * w + 6$ 。l是每星期的行数。



日历函数

• `calendar.month(year, month, w=2, l=1)`

返回一个多行字符串格式的year年month月日历，两行标题，一周一行。每日宽度间隔为w字符。每行的长度为 $7 * w + 6$ 。l是每星期的行数

```
import calendar  
print(calendar.month(2019, 2))
```

```
/Library/Frameworks/Python.framework/Versions/3.7/bin/python3.7 /Users  
February 2019  
Mo Tu We Th Fr Sa Su  
      1  2  3  
 4  5  6  7  8  9 10  
11 12 13 14 15 16 17  
18 19 20 21 22 23 24  
25 26 27 28
```



日历函数

- **calendar.monthcalendar(year, month)**

返回一个整数的单层嵌套列表。每个子列表装载代表一个星期的整数。Year年month月外的日期都设为0;范围内的日子都由该月第几日表示,从1开始。

```
import calendar
print(calendar.monthcalendar(2019, 2))
```

test

```
/Library/Frameworks/Python.framework/Versions/3.7/bin/python3.7 /Users/liuxiaoxiao/PycharmProjects/untitled4/test.py
[[0, 0, 0, 0, 1, 2, 3], [4, 5, 6, 7, 8, 9, 10], [11, 12, 13, 14, 15, 16, 17], [18, 19, 20, 21, 22, 23, 24], [25, 26, 27, 28, 0, 0, 0]]
```

Process finished with exit code 0

日历函数

- **calendar.monthrange(year, month)**

返回两个整数。第一个是该月的星期几的日期码，第二个是该月的日期码。日从0（星期一）到6（星期日）；月从1到12。

- **calendar.prcal(year, w=2, l=1, c=6)**

相当于 `print(calendar.calendar(year, w, l, c))`



随机数函数

- `random.random()`

用于生成一个0到1的随机浮点数: $0 \leq n < 1.0$ 。

```
import random
# 生成第一个随机数
print("random():", random.random())
# 生成第二个随机数
print("random():", random.random())
```

demo

test

```
/Library/Frameworks/Python.framework/Versions/3.7/bin/python3.7 /Users/liuxiaoxia
random(): 0.09916671037992852
random(): 0.4012715929302604
```

Process finished with exit code 0



随机数 函数

- `random.uniform(a,b)`

返回a,b之间的随机浮点数，范围[a,b]或[a,b]取决于四舍五入，a不一定要比b小。

```
import random
print("random:", random.uniform(50, 100))
print("random:", random.uniform(100, 50))
```

demo

test

/Library/Frameworks/Python.framework/Versions/3.7/bin/python3.7 /Users/liuxiaoxiao/PycharmProje

random: 93.89670606167616

random: 55.78465288980606

Process finished with exit code 0



随机数函数

- `random.randint(a,b)`

返回a,b之间的整数，范围[a,b]，注意：传入参数必须是整数，a一定要比b小。

```
import random  
print(random.randint(12,20))
```

demo

test

```
/Library/Frameworks/Python.framework/Versions/3.7/bin/python3.7 /Users/liuxiaoxiao/  
14
```

```
Process finished with exit code 0
```



随机数 函数

- `random.randrange([start], stop[, step])`

返回有个区间内的整数，可以设置step。只能传入整数，`random.randrange(10, 100, 2)`，结果相当于从[10, 12, 14, 16, ... 96, 98]序列中获取一个随机数。

```
import random  
print(random.randrange(10, 100, 2))
```

est

```
/Library/Frameworks/Python.framework/Versions/3.7/bin/python3.7 /Users/liuxiaoxiao/PycharmProjects/  
74
```

```
Process finished with exit code 0
```



随机数函数

- `random.choice(sequence)`

从sequence（序列，是有序类型的）中随机获取一个元素，列表、元组、字符串都属于sequence。

```
import random
print(random.randrange(10,100,2))
#结果等效于
print(random.choice(range(10,100,2)))
```

demo

test

```
/Library/Frameworks/Python.framework/Versions/3.7/bin/python3.7 /Users/liuxiaoxi
92
20
```

Process finished with exit code 0



随机数 函数

- `random.shuffle(x[,random])`

用于将列表中的元素打乱顺序，俗称为洗牌。

```
import random
p = ["Python", "is", "powerful", "simple"]
random.shuffle(p)
print(p)
```

demo

test

```
/Library/Frameworks/Python.framework/Versions/3.7/bin/python3.7 /Users/liuxiaoxiao/f
['simple', 'powerful', 'Python', 'is']
```

```
Process finished with exit code 0
```



随机数 函数

- `random.sample(sequence,k)`
从指定序列中随机获取k个元素作为一个片段返回，
`sample`函数不会修改原有序列

```
import random
list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
slice = random.sample(list, 5)
print(slice)
print(list)
```

demo

test

```
/Library/Frameworks/Python.framework/Versions/3.7/bin/python3.7 /Users/liuxiaoxiao/Pychar
[9, 8, 6, 2, 5]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Process finished with exit code 0



Thanks for watching

