

# 常用模块

## 简介

本节实验将介绍 Python 中常用的模块，这些模块能够让我们更有效率地开发一个 Python 项目，在后续的项目实战中都会用到。

## 知识点

- os：操作系统相关的操作
- sys：获取 Python 解释器状态
- datetime：时间日期及相关计算
- time：处理时间、打印当前时间、强制挂起当前进程
- requests：网络请求标准库
- base64：用字符表示二进制数据
- copy：深复制与浅复制，复制可变数据类型
- configparser：读取配置文件
- collections：提供一系列特殊的容器类
- re：正则表达式库

本节操作的内容比较简单，只需要在 Python 3 的交互式环境中尝试每个模块中的方法。

## OS

os 模块提供了一些接口来获取操作系统的一些信息和使用操作系统功能。

在 posix 操作系统（Unix、Linux、Mac OS X）中，我们可以使用 mkdir、touch、rm、cp 等命令来创建、删除文件和目录，以上系统命令只是调用了操作系统提供的接口，Python 内置的 os 模块也可以直接调用这些系统接口。

本节我们使用 Python 交互命令行来演示 os 模块中关于文件和目录的基本操作：

```
>>> import os
# posix 操作系统，指的是 Linux、Unix、MacOS 操作系统，使用此方法可以查看
# 在 Windows 操作系统中，该值应为 ne
>>> os.name
'posix'
```

首先来看 `os.path` 这个非常常用的标准库，这个库主要的用途是获取和处理文件及文件夹属性。

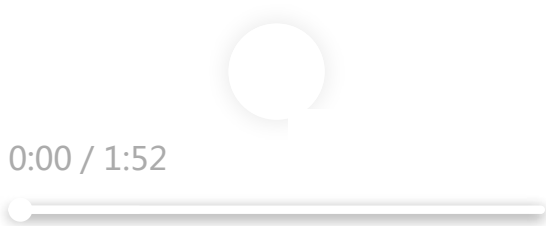
🔗 楼+之Python实战第10期 (/courses/1190)

下面代码举例介绍几个常用的方法，更多的内容在使用到的时候查阅文档。

实验代码内容，需要在 Python 交互环境中操作：

```
>>> import os    # 引入os模块
>>> filename = '/home/shiyanlou/test.txt'
>>> os.path.abspath(filename) # 返回文件的绝对路径
'/home/shiyanlou/test.txt'
>>>
>>> os.path.basename(filename) # 返回文件名
'test.txt'
>>> os.path.dirname(filename) # 返回文件路径
'/home/shiyanlou'
>>>
>>> os.path.isfile(filename) # 判断路径是否为文件
True
>>>
>>> os.path.isdir(filename) # 判断路径是否为目录
False
>>> os.path.exists(filename) # 判断路径是否存在
True
>>> os.path.join('/home/shiyanlou', 'test.txt') # 把目录和文件名合成一个路径
'/home/shiyanlou/test.txt'
>>>
>>> os.path.split(filename) # 该方法将绝对路径分为目录和文件名两部分，并放入元组中返回
('/home/shiyanlou', 'test.txt')
>>> os.path.splitext('test.txt') # 该方法可以将文件名和扩展名分开，放入元组中返回
('test', '.txt')
>>>
```

os.path 操作视频：



拓展阅读：os.path — Common pathname manipulations (英文)  
(<https://docs.python.org/3.5/library/os.path.html>)

下面演示一下在 `os` 模块中其它常用的方法：

```

>>> import os
>>> dirname = '/home/shiyanlou/testdir'
>>> filename = '/home/shiyanlou/test1.txt'
>>> os.mkdir(dirname)    # 新建目录
>>> os.rmdir(dirname)    # 删除目录
>>> os.rename('/home/shiyanlou/test.txt', filename)    # 将 test.txt 改名为 test1.txt
>>> os.remove(filename)   # 删除 test1.py 文件
>>> os.listdir('.')      # 该方法可以获得参数目录下的全部目录和文件的名字，包括隐藏文件和隐藏目录
['.oh-my-zsh', '.cache', '.bash_logout', 'Code', '.pydistutils.cfg', '.zshrc', '.profile',
'.codebox', 'file2.txt', 'Desktop', '.pip', '.local', '.ICEauthority', '.config', '.vnc', '.
dbus', '.zsh_history', '.bashrc', '.Xauthority', '.zcompdump-ac994418981f-5.0.2', '.hushlogi
n', '.zsh-update', '.gemrc', '.npm', '.scim', 'test', '.zcompdump']

```

## 练习题：创建多个文件夹及文件

本节将考察创建使用 os 模块创建多个文件夹以及文件，请按照题目要求完成，点击 [下一步](#) 系统将自动检测完成结果。

在 /home/shiyanlou 目录下创建代码文件 add\_file.py 文件：

```

cd /home/shiyanlou
touch add_file.py

```

在 /home/shiyanlou/add\_file.py 中写入代码，要求运行代码文件后可以自动在 /home/shiyanlou 目录下创建如下的项目结构：

```

syl
├─ A
│   └─ __init__.py
├─ B
│   └─ __init__.py
├─ C
│   └─ __init__.py
└─ __init__.py

```

程序完成后，点击 [下一步](#)，系统将自动检测完成结果。

拓展阅读：Python OS 文件 / 目录方法 (<http://www.runoob.com/python/os-file-methods.html>)

## sys

sys 模块提供了一些对于 Python 解释器的相关操作。

### sys.version

该属性可以获得 Python 解释器的版本信息：  
📍 楼+之Python实战第10期 (/courses/1190)

```
>>> import sys
>>> sys.version
'3.5.3 (default, Apr 22 2017, 00:00:00) \n[GCC 4.8.4]'
```

## sys.path

该属性是一个列表，里面是 Python 解释器的搜索路径，其中第一个元素是空字符串，表示当前相对路径：

```
>>> sys.path
['', '/usr/lib/python3.5', '/usr/lib/python3.5/plat-x86_64-linux-gnu', '/usr/lib/python3.5/lib-dynload', '/usr/local/lib/python3.5/dist-packages', '/usr/lib/python3/dist-packages']
```

## sys.argv

在终端运行程序时，该属性可以获得命令行参数列表，列表中每个元素都是字符串，第一个元素为程序名。将以下代码写入 `argv.py` 文件中：

```
import sys

print(sys.argv)
```

在终端运行程序，查看结果：

```
shiyancelou:~/ $ python3 argv.py arg1 arg2 arg3
['argv.py', 'arg1', 'arg2', 'arg3']
```

- 拓展阅读 `sys` - 系统特定的参数和功能 (<https://docs.python.org/3/library/sys.html>)

## datetime

`datetime` 模块提供了一些类用于操作日期时间及其相关的计算。比较常用三个类型：

- `date` 封装了日期的操作
- `datetime` 封装日期+时间操作
- `strptime` 方法将字符串转换为 `datetime` 数据类型
- `strftime` 方法将 `datetime` 数据类型转换为字符串
- `timedelta` 表示一个时间间隔，也就是日期时间的差值

日期时间的获取：

楼之Python实战第10期(courses/1190)

```

>>> from datetime import datetime # 引入模块
>>> date.today() # 获得此时日期
datetime.date(2018, 9, 2)
>>> datetime.utcnow() # 获得格林威治时间，即伦敦时间，比北京时间慢 8 小时
datetime.datetime(2018, 9, 2, 2, 19, 28, 309096)
>>> datetime.now() # 获得本地时间，即北京时间
datetime.datetime(2018, 9, 2, 10, 19, 36, 790761)
>>> print(datetime.now()) # 用 print 格式化打印样式
2018-09-02 10:19:47.566207
>>> datetime.now().date() # 取日期
datetime.date(2018, 9, 2)
>>> datetime.now().time() # 取时间
datetime.time(10, 20, 3, 559274)
>>> print(datetime.now().date())
2018-09-02
>>> print(datetime.now().time())
10:20:20.324323
>>> t = datetime.now()
>>> t.year
2018
>>> t.month
9
>>> t.day
2
>>> t.hour
10
>>> t.minute
29

```

`datetime.datetime` 对象与字符串之间的相互转换：

```


>>> type(datetime.now())
<class 'datetime.datetime'>
>>> s = '2020-01-31SYL 11:22:33ok'

# 用 striptime 方法将字符串转换为 datetime 数据类型
>>> d = datetime.strptime(s, '%Y-%m-%dSYL %H:%M:%Sok')
>>> d
datetime.datetime(2020, 1, 31, 11, 22, 33)
>>> print(d)
2020-01-31 11:22:33
>>> type(d)
<class 'datetime.datetime'>

# 用 strftime 方法将 datetime 数据类型转换为字符串
# %a 简化英文星期名称
# %m 月份 (01-12)
# %d 月中的一天 (0-31)
>>> s = d.strftime('%a %m %d %H:%M:%S')
>>> s
'Fri 01 31 11:22:33'

```

用 `timedelta` 表示时间差值，可以精确到微秒：

 `timedelta(days=0, hours=0, minutes=0, microseconds=0, milliseconds=0, minutes=0, hours=0, weeks=0)`

用 `timedelta` 对 `datetime` 进行加减操作：

```
>>> from datetime import timedelta # 引入方法
>>> now = datetime.now()
>>> now
datetime.datetime(2018, 9, 2, 10, 44, 11, 624481)

# 参数包括 days、hours、seconds 等
# 所有参数均为可选参数，且默认都是 0，参数值可以是整数、浮点数、正数或负数
>>> now + timedelta(days=1)
datetime.datetime(2018, 9, 3, 10, 44, 11, 624481)
>>> now + timedelta(days=3, hours=-1.5)
datetime.datetime(2018, 9, 5, 9, 14, 11, 624481)
>>> now - timedelta(days=-3, hours=1.5)
datetime.datetime(2018, 9, 5, 9, 14, 11, 624481)
```

更多关于 `datetime` 模块的内容，可以阅读：

- 拓展阅读：datetime 模块官方文档（英文）  
(<https://docs.python.org/3/library/datetime.html>)
- 拓展阅读：Python 中的 datetime 模块的简单使用  
(<https://www.jianshu.com/p/552dfd0dc6e9>)

## time

`time` 模块用于处理时间，与 `datetime` 有些类似，下面举例说明一些常用的属性和方法。

`time.ctime` 方法获取当前时间的字符串：

```
>>> import time
# 格式为：星期 月 日 时:分:秒 年份
>>> time.ctime()
'Sat Dec  8 13:28:36 2018'
```

`time.localtime` 方法的返回值是 `struct_time` 类型的对象，该对象有一些属性可以获取当前各种时间：

### 🔗 楼中之Python实战第10期 (/courses/1190)

```
time.struct_time(tm_year=2018, tm_mon=12, tm_mday=8, tm_hour=13, tm_min=30, tm_sec=16, tm_wday=5, tm_yday=342, tm_isdst=0)
>>> st = time.localtime()
>>> st.tm_yday # 今天是今年的第几天
342
>>> st.tm_hour # 现在是今天的第几个小时
13
>>> st.tm_mon # 现在是今年的第几个月
12
```

time.strftime 方法可以将 struct\_time 类型对象格式化为字符串：

```
>>> st
time.struct_time(tm_year=2018, tm_mon=12, tm_mday=8, tm_hour=13, tm_min=30, tm_sec=37, tm_wday=5, tm_yday=342, tm_isdst=0)
>>> time.strftime('%Y-%m-%d %H:%M:%S', st)
'2018-12-08 13:30:37'
```

time.time 方法的返回值是从公元 1970 年至此时此刻的秒数，它是一个浮点数，经常与之配合使用的另一个方法 time.sleep 可以强制挂起当前进程，即在某一段时间内，什么也不做，举例说明：

```
>>> time.time()
1544283831.692716
>>> def test():
...     start_time = time.time() # 记录开始时间
...     time.sleep(1.2)         # 挂起 1.2 秒
...     end_time = time.time()   # 记录结束时间
...     print('运行耗时: {:.2f}s'.format(end_time-start_time))
...
>>> test()
运行耗时: 1.2s
```

## requests

如果你写过爬虫，对这个库应该不陌生。在 requests 库出现之前，网络请求通常用标准库中的 urllib。requests 出现之后，它俨然已经成了 Python 事实上的网络请求标准库。

requests 的接口非常简单：

```

>>> import requests
>>> r = requests.get('https://www.shiyanlou.com')
>>> r.status_code
200
>>> r.headers['content-type']
'text/html; charset=utf-8'
>>> r.text
'\n<!DOCTYPE html>\n<html lang="zh-CN">\n    <head>\n        <meta charset="utf-8">\n
    <meta http-equiv="X-UA-Compatible" content="IE=edge">\n    <title>实验楼 - Python 教程网</title>\n    <meta name="description" content="实验楼 - Python 教程网">\n    <meta name="keywords" content="Python, 教程, 实验楼">\n    </head>\n    <body>\n        <div class="container">\n            <div class="row">\n                <div class="col-md-12">\n                    <div class="header">\n                        <h1>实验楼 - Python 教程网</h1>\n                    </div>\n                </div>\n            </div>\n        </div>\n    </body>\n    </html>\n'

```

请求 JSON 数据：

```

>>> r = requests.get('https://api.github.com')
>>> r.json()
{'current_user_url': 'https://api.github.com/user', ... }

```

json() 方法会将返回的 JSON 数据转化为一个 Python 字典。

还可以用 requests 执行 POST , DELETE 等其它的 HTTP 方法。

requests 模块操作视频：



0:00 / 2:38



- 拓展阅读 《Requests 模块官方文档 - 快速上手》 ([http://docs.python-requests.org/zh\\_CN/latest/user/quickstart.html](http://docs.python-requests.org/zh_CN/latest/user/quickstart.html))

## base64

base64 是一种编码方式，它可以将二进制数据编码 64 个可打印的 ASCII 字符。Base64 要求把每三个 8Bit 的字节转换为四个 6Bit 的字节（ $3 \times 8 = 4 \times 6 = 24$ ），然后把 6Bit 再添两位高位 0，组成四个 8Bit 的字节，也就是说，转换后的字符串理论上将要比原来的长 1/3。

```

import base64
>>> base64.b64encode(b'Hello, shiyanlou!')
b'SGVsbG8sIHNoaXlhbmxvdSE='
>>> base64.b64decode(b'SGVsbG8sIHNoaXlhbmxvdSE=')
b'Hello, shiyanlou!'

```



- 拓展阅读：编码解码 Base64 数据 ([https://python3-cookbook.readthedocs.io/zh\\_CN/latest/c06/p10\\_decode\\_encode\\_base64.html?highlight=base64](https://python3-cookbook.readthedocs.io/zh_CN/latest/c06/p10_decode_encode_base64.html?highlight=base64))

## copy

介绍 copy 模块之前，我们先来讨论一下如何复制一个可变对象，用列表来举例说明：

```
# 首先创建一个列表
>>> l = [1, 'hello', time.time()]
>>> l
[1, 'hello', 1544285192.3967364]
# id 方法查看对象在内存中的 ID
>>> id(l)
140146856824904
# 用等号赋值的方法将 l 赋值给 l1 这个变量
>>> l1 = l
# l1 的 ID 与 l 相同，修改 l1 就会自动改变 l 的值，因为它们拥有相同的内存地址
>>> id(l1)
140146856824904
>>> l1.append('world')
>>> l1
[1, 'hello', 1544285192.3967364, 'world']
>>> l
[1, 'hello', 1544285192.3967364, 'world']
```

如何复制一个 l 列表，同时新列表与 l 无关，修改一个列表不会影响另一个列表呢？此时可以使用 copy.copy 方法来实现：

```
# 使用 copy.copy 方法获得的新列表与原列表值相同，但内存地址不同
>>> l2 = copy.copy(l)
>>> l2
[1, 'hello', 1544285192.3967364, 'world']
>>> l
[1, 'hello', 1544285192.3967364, 'world']
>>> id(l2)
140146856826952
>>> id(l)
140146856824904
# 修改其中一个的值，不会影响另一个
>>> l2.pop()
'world'
>>> l2
[1, 'hello', 1544285192.3967364]
>>> l
[1, 'hello', 1544285192.3967364, 'world']
```

copy.copy 方法叫做浅复制，与之对应的还有 copy.deepcopy 深复制，举例说明：

### 🔗 楼之Python实战第10期 (/courses/1190)

```
>>> l = [1, [2, 3]]
# 按照前面的方法来复制一个列表 l1
>>> l1 = copy.copy(l)
>>> l1
[1, [2, 3]]
>>> l
[1, [2, 3]]
# 修改列表的最后一个元素的值
>>> l1[-1]
[2, 3]
>>> l1[-1].append(4)
>>> l1
[1, [2, 3, 4]]
# 原列表中的最后一个元素也被修改，说明这两个列表中的子列表指向相同的内存地址
>>> l
[1, [2, 3, 4]]
>>> id(l[-1])
140146856679368
>>> id(l1[-1])
140146856679368
```

现在我们用深复制 `copy.deepcopy` 方法来复制出来一个新的列表：

```
>>> l2 = copy.deepcopy(l)
>>> l
[1, [2, 3, 4]]
>>> l2
[1, [2, 3, 4]]
# 修改 l2 的最后一个元素
>>> l2[-1].pop()
4
# l2 的最后一个元素被修改，但 l 列表没有变化，说明 deepcopy 方法可以将可变对象的所有元素都克隆一次并放到新的内存地址中
>>> l2
[1, [2, 3]]
>>> l
[1, [2, 3, 4]]
# 验证它们的内存 ID，确实是不同的
>>> id(l[-1])
140146856679368
>>> id(l2[-1])
140146856824904
```

## configparser

配置文件通常是在程序初始化时对程序或服务进行配置的文件。配置文件有很多种，例如 ini、json、xml、yaml 等，`configparser` 模块支持对 ini 格式的配置文件进行读取以及修改，这一类配置文件的样式如下：

## 🔗 楼+之Python实战第10期 (/courses/1190)

```
[section0]      # 配置组0
key0 = value0   # 配置项以键值对的形式填写
key1 = value1

[section1]      # 配置组1
key2 = value2
key3 = value3

.....
```

下面对 configparser 模块的使用进行简单说明。

在 /home/shiyanlou 目录下新建 syl.ini 文件，并向其中写入如下配置内容：

```
[user]
name = shixiaolou
id = 123
data = 2018-01-01

[courses]
python = python1
java = java1
php = php1
```

在 Python3 交互式命令行中进行操作：

```
>>> from configparser import ConfigParser # 引入模块中的 ConfigParser 类
>>> config = ConfigParser() # 对类进行实例化
>>> config.read('syl.ini', encoding='UTF-8') # 读取文件，指定编码格式
['syl.ini']
>>> config.sections() # 获取所有的配置组
['user', 'courses']
>>> config.options('user') # 获取 user 配置组下的所有 options
['name', 'id', 'data']
>>> config.items('user') # 获取 user 配置组下的所有键值对，返回为一个列表
[('name', 'shixiaolou'), ('id', '123'), ('data', '2018-01-01')]
>>> config.get('user', 'name') # 获取 user 配置组下的 name 对应的值
'shixiaolou'
>>> config.add_section('permission') # 添加新的 permission 配置组
>>> config.set('permission', 'isMember', 'true') # 对 permission 配置组添加新的 option
>>> config['permission']['isMember'] = 'false' # 修改 permission 配置组中的 isMember 为 false
>>> with open('syl.ini', 'w', encoding='utf-8') as file:
...     config.write(file) # 将修改后的值写入配置文件中
...
>>>
```

最后查看修改后的配置文件内容：

```
shixianlou@py:~$ cat syl.ini
楼之Python实战第10期 (/courses/1190)
[user]
name = shixiaolou
id = 123
data = 2018-01-01

[courses]
python = python1
java = java1
php = php1

[permission]
ismember = false
```

可以看到 syl.ini 配置文件中的内容被成功修改了。

## collections

`collections` 模块主要提供了一些特别的容器，在特定的情况下我们使用这些容器可以使问题处理更容易一些。

下面，我们就针对 `collections` 模块常见用法举例说明：

### 判断数据类型

```
>>> from collections import Iterable # 判断对象是否为可迭代对象
>>> isinstance('shiyianlou', Iterable) # 字符串是可迭代对象
True
>>> isinstance(123, Iterable) # 数值不是可迭代对象
False

>>> from collections import Iterator # 判断对象是否为迭代器
>>> isinstance('shiyianlou', Iterator) # 字符串不是迭代器
False
>>> isinstance(iter('shiyianlou'), Iterator) # iter 对象为迭代器
True
>>> from collections import Generator # 判断对象是否为生成器
>>> isinstance(iter('shiyianlou'), Generator) # iter 对象不是生成器
False
>>> g = (i**2 for i in range(9)) # 创建生成器
>>> g
<generator object <genexpr> at 0x1045c0750>
>>> isinstance(g, Generator) # 显然肯定是生成器了
True
```

### OrderedDict

`OrderedDict` 是一个特殊的字典。字典本质上是一个哈希表，其实现一般是无序的，`OrderedDict` 能保持元素插入的顺序：

```
楼之Python实战第10期 (/courses/1190)
>>> from collections import OrderedDict
>>> d = OrderedDict()
>>> d['apple'] = 1
>>> d['google'] = 2
>>> d['facebook'] = 3
>>> d['amazon'] = 4
>>> d
OrderedDict([('apple', 1), ('google', 2), ('facebook', 3), ('amazon', 4)])
```

OrderedDict 同样能以元素插入的顺序来进行迭代或者序列化：

```
>>> for key in d:
...     print(key, d[key])
...
apple 1
google 2
facebook 3
amazon 4
>>> import json
>>> json.dumps(d)
'{"apple": 1, "google": 2, "facebook": 3, "amazon": 4}'
```

## namedtuple

使用普通的元组 ( tuple ) 存在一个问题，每次用下标去获取元素，可能会不知道你这个下标下的元素到底代表什么。namedtuple 能够用来创建类似于元组的类型，可以用索引来访问数据，能够迭代，也可以通过属性名访问数据。

下面使用命名元组表示坐标系中的点：

```
>>> from collections import namedtuple
>>> Point = namedtuple('Point', ['x', 'y'])
>>> p = Point(10, 12)
>>> p.x
10
>>> p.y
12
```

## Counter

Counter 用来统计一个可迭代对象中各个元素出现的次数，以字符串为例：

```
>>> from collections import Counter
>>> c = Counter('https://www.shiyanlou.com')
>>> c
Counter({'w': 3, 'h': 2, 't': 2, 's': 2, '/': 2, '.': 2, 'o': 2, 'p': 1, ':': 1, 'i': 1, 'y': 1, 'a': 1, 'n': 1, 'l': 1, 'u': 1, 'c': 1, 'm': 1})
```

找出出现次数最多的前 n 个元素：

🔗 楼+之Python实战第10期 (/courses/1190)

```
>>> c.most_common(3)
[('w', 3), ('h', 2), ('t', 2)]
```

如果你想了解更多关于 collections 模块的用例，可以阅读下面这篇文章：

- 拓展阅读 《collections 模块作用和用法》(<https://eastlakeside.gitbooks.io/interpy-zh/content/collections/collections.html>)

## re 正则表达式

在做文字处理或编写程序时，用到查找、替换等功能，使用正则表达式能够简单快捷地完成目标。简单而言，正则表达式通过一些特殊符号的帮助，使用户可以轻松快捷地完成查找、删除、替换等处理程序。例如 Linux 系统中的 grep, expr, sed, awk 等高级命令都离不开正则表达式。正规表示法基本上是一种『表示法』，只要工具程序支持这种表示法，那么该工具程序就可以用来作为正规表示法的字符串处理之用。

本节内容仅涉及 Python 中使用正则表达式的常规操作，这些操作可以应对绝大多数正则表达式的使用场景。本节内容不会使用常见的陈列一堆令初学者眼晕的正则符号及其说明的形式，而是一步一步地，以可以直接动手操作的形式讲述。大家只需按照课程文档的步骤进行操作并稍作思考，即可快速掌握正则表达式的基本套路。

正则表达式是一个特殊的字符序列，从对象字符串中匹配它，可以获得想要的字符串片段。切记，正则表达式不需要背，熟练操作即可。

首先，引入 re ，在 Python 中，使用该模块来写正则表达式：

```
>>> import re
```

要获取某个字符或字符串，使用 findall 方法可以获取全部能够匹配的片段，放到一个列表里。findall 方法第一个参数为正则表达式，规定匹配规则，第二个参数为被匹配对象，下文将大量使用该方法。

## 普通字符

下面代码中的参数 "hello"、"实验楼"、"o" 均为普通字符/字符串：

```
>>> s = '楼之Python实战第10期 (你好像实验楼)'
>>> re.findall('hello', s)
['hello']
>>> re.findall('实验楼', s)
['实验楼']
>>> re.findall('o', s)
['o', 'o']
```

## 元字符

- `\d` 获取所有数字 0 - 9
- `\D` 匹配所有非数字
- `\w` 匹配所有单词字符，包括大小写字母、数字、下划线、中文
- `\W` 匹配剩下的，空格、换行符、特殊字符等：

```
>>> s = 'abc_004\nGT# 预言'
>>> re.findall('\d', s)
['0', '0', '4']
>>> re.findall('\D', s)
['a', 'b', 'c', '_', '\n', 'G', 'T', '#', ' ', '预', '言']
>>> re.findall('\w', s)
['a', 'b', 'c', '_', '0', '0', '4', 'G', 'T', '预', '言']
>>> re.findall('\W', s)
['\n', '#', ' ']
```

## 字符集

顾名思义，就是字符的集合，用中括号表示 `[ ]`，它匹配任意一个符合条件的字符。字符集内 `^` 表示“非”，它只能在字符集内被使用；字符集外 `^` 在模式的开始处使用，表示需要在行的开始处进行匹配，后面会讲到。

因此，`\d` 等于 `[0-9]`，`\D` 就等于 `[^0-9]`。注意：`\w` 不等于 `[a-zA-Z_0-9]`，因为后者不能匹配中文。

具体用法举例：

```

楼之Python实战第10期 (courses/1190)
>>> re.findall('char_[ad]', s)
['char_a', 'char_d']
>>> re.findall('char_[^ad]', s)
['char_b', 'char_c']
>>> re.findall('char_[a-c]', s) # a-c 表示 a 至 c
['char_a', 'char_b', 'char_c']
>>> re.findall('[^0-9a-zA-Z_]', s)
[' ', ' ', ' ', '正', '则', '#']
>>> re.findall('[^\w]', s)
[' ', ' ', ' ', '#']

```

## 空白字符

空白字符共有这四种：' ' 空格、\n 换行符、\t 制表符、\r 回车符，使用元字符 \s 匹配它们，自然 \S 匹配任意非空白字符：

```

>>> s = '楼+ \nPython\t'
>>> re.findall('\s', s)
[' ', '\n', '\t']
>>> re.findall('[^\s]', s)
['楼', '+', 'P', 'y', 't', 'h', 'o', 'n']
>>> re.findall('\S', s)
['楼', '+', 'P', 'y', 't', 'h', 'o', 'n']

```

## 数量符号和特殊数量符号

使用大括号 {} 标定匹配字符的数量。默认的匹配模式为贪婪模式，即选取尽可能多的匹配字符；? 表示非贪婪模式，即选取最少的匹配字符，所以 {3, 6}? 就等同于 {3}：

```

>>> s = 'linux,python-git123'
>>> re.findall('[a-z]{3}', s)
['lin', 'pyt', 'hon', 'git']
>>> re.findall('[a-z]{3,6}', s)
['linux', 'python', 'git']
>>> re.findall('[a-z]{3,6}?', s)
['lin', 'pyt', 'hon', 'git']

```

特殊数量符号：

- \* 匹配任意数量的字符
- ? 匹配 0 或 1 个字符，这也是它可以设置非贪婪模式的原因。也就是说，所谓的非贪婪，只是被设定为最多匹配 1 个的贪婪模式，贪婪是永恒的
- + 匹配 1 个或多个字符
- . 匹配除换行符 \n 以外任意 1 个字符



注意：特殊数字符号控制的是紧挨着该符号左边的字符或字符集：

🔗 楼+之Python实战第10期 (/courses/1190)

```
>>> s = 'hell#hello$helloo'
>>> re.findall('hello*', s) # 尽可能多地匹配任意数量的字符 "o"
['hell', 'hello', 'helloo']
>>> re.findall('hello?', s)
['hell', 'hello', 'hello']
>>> re.findall('hello+', s)
['hello', 'helloo']
>>> re.findall('H.', 'Hello')
['He']
>>> re.findall('H.', 'H\nello') # "." 不能匹配换行符
[]
```

## 边界字符

在模式的开始处使用 `^` 表示需要在行的开始处进行匹配。例如 `^abc` 可以匹配 `abc123` 但不匹配 `123abc`。在模式的末尾处使用 `$` 表示需要在行的末端进行匹配。例如 `abc$` 可以匹配 `123abc` 但不能匹配 `abc123`。

简单的例子：

```
>>> s = 'hello world'
>>> re.findall('wo', s)
['wo']
>>> re.findall('^wo', s)
[]
>>> re.findall('.*lo', s)
['hello']
>>> re.findall('.*lo$', s)
[]
```

## 字符组

把任意数量的字符用小括号 `()` 括起来，就是字符组，目的是为匹配成功的字符串进行分组。`findall` 方法会匹配第一个参数的正则表达式并过滤掉字符组外面的字符，保留小括号内的部分：

```
>>> s = ' 实验楼 Python\n'
# \w 不能匹配“楼”和"P"中间的空格，所以结果为两个成功匹配的字符串
>>> re.findall('\s*(\w+)\s*', s)
['实验楼', 'Python']
>>> re.findall('\s*(.+)\s*', s)
['实验楼 Python']
```

## 模式参数

`findall` 方法还有第三个参数，模式参数。下面介绍两个较为常用的：`re.I` 忽略大小写，`re.S` 匹配空白字符。多个模式参数用 `|` 隔开，举例说明：

📖 楼+之Python实战第10期 (/courses/1190)

```
>>> s = 'Hello\nWorld'
>>> re.findall('Lo', s) # 不能匹配 "L"
[]
>>> re.findall('Lo', s, re.I) # re.I 忽略大小写
['lo']
>>> re.findall('Lo.', s, re.I) # "." 不能匹配换行符
[]
>>> re.findall('Lo.', s, re.I | re.S) # re.S 匹配任意空白字符，包括换行符
['lo\n']
```

## 正则表达式小结

本节实验讲解了 Python 正则表达式的基本规则，能够满足日常大多数使用场景。当然正则还有更为复杂的规则，各个语言、系统之间在使用正则表达式时也有微小的差异，`re` 模块还有其它比较实用的方法来利用正则表达式，例如 `compile`、`match`、`sub` 等，希望大家通过本节课程讲解的知识，在需要时可以很容易地自学掌握其它复杂的规则和方法。

## 总结

本节实验将介绍一些在后续项目实战中会用到的常用模块：

- `os`：操作系统相关的操作
- `sys`：获取 Python 解释器状态
- `datetime`：时间日期及相关计算
- `time`：处理时间、打印当前时间、强制挂起当前进程
- `requests`：网络请求标准库
- `base64`：用字符表示二进制数据
- `copy`：深复制与浅复制，复制可变数据类型
- `configparser`：读取配置文件
- `collections`：提供一系列特殊的容器类
- `re`：正则表达式库

Python 的模块实在是太多了，在使用中不能够靠纯粹的死记硬背，而是靠积累经验熟能生巧。一般项目开发的过程中遇到某种开发需求，首先去搜索下是否 Python 已经有现成的模块去处理这个需求了，如果有的话则直接在代码中使用就可以了，不用再自己从头开始实验。这也是 Python 高效开发的一大优势。

本周的实验已经完成，完成本周的学习之后，我们再次根据脑图的知识点进行回顾，让动手实践过程中学习到的知识点建立更加清晰的体系。

请点击以下链接回顾本周的 Python3 基础知识的学习：

🔗 楼+之Python实战第10期 (/courses/1190)

- Python3 基础知识点脑图

(<http://naotu.baidu.com/file/f3effc51a94420eb5686647e9f973326?token=00e911d0ed10e629>)

请注意实验只会包含常用的知识点，对于未涉及到的知识点，如果在脑图中看到可以按照实验中学习的方法使用 `help()` 进行查询或者查看官方文档，也非常欢迎在讨论组里与同学和助教进行讨论，技术交流也是学习成长的必经之路。

*\*本课程内容，由作者授权实验楼发布，未经允许，禁止转载、下载及非法传播。*

上一节：[选学] 挑战：多进程工资计算器 (/courses/1190/labs/8528/document)

下一节：[选学] 挑战：使用模块优化工资计算器 (/courses/1190/labs/8530/document)