

模块

自定义模块

定义：

一个文件就是一个模块（能被调用的文件，模块就是一个工具箱，工具就是函数）

作用：

1. 将代码文件化管理，提高可读性，避免重复代码
2. 拿来就可以用，避免重复造轮子，python中有许多类库，可以提升开发效率

import导入时做的三件事情

1. 将.py文件中的所有代码读取到当前文件
2. 在当前文件开辟空间
3. 等待被调用

注意的事情

1. import导入同一模块名时，只执行一次
2. 工具箱名字过长时可以用as起别名
3. 导入模块的时候不能加后缀
4. 每个模块都有一块内存空间，理论上是全局空间

from.....import

1. 飘红不代表报错
2. 好处：使用起来方便
3. 坏处：容易与当前执行文件中的定义功能冲突，名字一样就覆盖

from.....import 与 import的区别：

1. from只能执行导入的工具
2. import能够执行整个模块中所有的功能
3. import 只支持导入当前文件夹下的所有模块，from可以指定导入的功能
4. import 后面不能加 . 操作
5. from 比 import更灵活

模块导入的顺序

1. append
 - sys.path.append(r"文件路径")
 - 内存 > 内置 > 第三方 > 自定义
2. insert
 - sys.path.insert(指定插入的位置, r"文件路径")
 - 内存 > 自定义 > 内置 > 第三方

模块的两种用法：

- if __ name __ == "__ main __"

1. 当做模块被导入：import 与 from
2. 当做脚本被执行

导入模块时遇到的坑：

1. 注意自己定义的模块名字与系统名字冲突
2. 注意自己的思路--循环导入的时候建议导入模式放后边一点
3. 不建议一行导入多个

from 模块 import * -- 导入整个工具箱

只有py文件当做模块被导入时，字节码才会进行保留

通过 `__all__` 控制要导入的模块

as 支持import 和from ， 避免覆盖之前的内容

time模块

- 定义：与时间相关的模块，属于内置模块，也被称为标准库
1. `time.time()`：时间戳
 2. `time.sleep()`：睡眠
 3. python中时间日期格式化符号：

python中时间日期格式化符号：

- `%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（0-31）
- `%H` 24小时制小时数（0-23）
- `%I` 12小时制小时数（01-12）
- `%M` 分钟数（00-59）
- `%S` 秒（00-59）

时间格式转换

时间戳（以秒计算）转 结构化时间

- `time.localtime(time.time())`——是一个命名元祖，可以使用索引和名字查找

结构化时间转字符串时间：有时差，需要减去八个小时

- `time.strftime("%Y-%m-%d %H:%M:%S",结构化时间)`

字符串时间转结构化时间：

- `time.strptime("字符串时间", "%Y-%m-%d %H:%M:%S")`

结构化时间转时间戳：

- `time.mktime(结构化时间)`

datetime模块

格式：

- `from datetime import datetime`

- 获取当前时间格式: `print(datetime.now())`
- 获取指定日期和时间: `datetime(具体的时间)`

格式转换

datetime 与时间戳转换

```
t = datetime.now()
print(t.timestamp())
```

输出格式如下: 1567598242.879536

将时间戳转换为对象

```
import time
t1 = time.time()
print(datetime.fromtimestamp(t1))
```

输出格式如下: 2019-09-04 19:56:35

将对象转换成字符串

```
t = datetime.now()
d.strftime('%Y-%m-%d %H:%M:%S')
```

将字符串转换成对象

```
d = "2018-12-31 10:11:12"
datetime.strptime(d,"%Y-%m-%d %H-%M-%S")
```

datetime的加减

- 格式: `from datetime import datetime,timedelta`
注意减的时候最大只能是周

random: 随机数

随机整数: `random.randint(1,5)`--随机1, 5之间的整数

随机小数: `random.random()`:默认0-1之间的小数

不包含的小数: `random.uniform(1,3)`: 大于1小于3的小数, 不包含3

从容器中随机选择3个元素, 以列表的形式返回,会出现重复元素:
`random.choice((1,2,3,4,5))`

从容器中随机选择3个元素, 以列表的形式返回,会出现重复元素:
`random.choices((1,2,3,4,5), k=3)`

列表元素任意3个组合不出现重复: `random.sample([1,"23",3,54,[3,5,6]],k=3)`

随机出现1-9以内随机的奇数: `random.randrange(1,9,2)`

打乱次序: `random.shuffle`

软件开发规范：分文件存储

当代码存放在一个py文件时的缺点

1. 不便于管理（主要体现在修改，增加时）
2. 可读性差
3. 加载速度慢

规范型文件夹

1. 启动文件：存放启动接口，文件夹一般命名为bin，py文件命名为starts
2. 公共文件：大家都可以用的文件夹，文件夹命名为lib，py文件命名为common
3. 配置文件：也叫静态文件，存储的都是变量，数据库的一些链接方式，获取到的都是redis，文件夹命名为conf，py文件命名为settings
4. 主逻辑：是程序的核心，一般使用core命名文件夹，py文件使用src命名
5. 用户相关数据：存储用户账户，密码等的文件，文件夹命名为db，py文件命名为register
6. 日志：记录重要信息，记录开发人员的操作记录，文件夹使用log，py文件使用logg命名

序列化

json :

- 将数据类型转换成字符串（序列化），将字符串转换成原数据类型（反序列），支持dict, list, tuple等，序列后都变成了列表

用法:

- dumps, loads ----- 用于网络传输

json.dumps: 将数据类型转换成字符串

json.loads: 将字符串转换成原数据类型

- dump, load ----- 用于文件传输

json.dump: 一个load对应一个dump

注意

1. 中文转换时，必须加ensure_ascii = False
2. 转换后的数据类型排序: sort_keys = True

pickle:

- 只有python有，几乎可以序列Python中所有数据类型（匿名函数不行）

1. 用于网络传输--dumps, loads

1. dumps: 将原数据类型转换成类似字节的东西

2. loads: 将类似于字节的东西转换成源数据类型

2. 用于文件读写--dump, load

1. dump: 写入文件的时候用的是wb模式, 没有解码encoding
2. load: 反序列化

OS:

工作路径:

os.getcwd(): 获取当前文件的路径

os.chdir(绝对路径): 改变当前工作目录

os.curdir () : 返回当前目录: "."

os.pardir () : 返回父级目录: ".."

文件夹

os.mkdir(): 创建文件夹

os.rmdir(): 删除空的文件夹, 不为空的不删除

os.makedirs(): 创建多层文件夹, 以递归的方式创建

os.removedirs () : 若目录为空则删除, 并递归到上一层继续删除空文件夹

os.listdir(): 列表显示指定文件夹下的所有内容, 并以列表的形式打印

文件: 必背

os.remove(): 删除文件, 彻底删除, 不能撤回 ***

os.rename(): 重命名文件夹 ***

os.stat(): 获取文件/目录信息 **

路径: 必背

os.path.abspath(): 返回的是绝对路径 ***

os.path.split(): 返回的是将路径分割成目录和文件名的元祖

os.path.dirname(): 返回到上级目录

os.path.basename(): 获取到当前文件名

os.path.join(""): 路径拼接, 多个路径拼合后返回 ***

os.path.getsize(): 返回文件的大小 *** 获取文件较准确

os.path.exists(路径): 判断路径是否存在

os.path.isabs(): 判断是不是绝对路径

os.path.isfile(): 判断文件存不存在

os.path.isdir(): 判断是不是文件夹

os.path.getatime(): 返回文件所指向的文件或者目录的最后访问时间

os.path.getmtime(): 返回文件所指向的文件或者目录的最后修改时间

SYS: 与Python解释器做交互的一个接口

sys.path : 返回模块的搜索路径, 模块查找的顺序 ***

sys.modules: 查看所有已加载到内存的模块

sys.argv : 只能在终端执行

sys.platform: 查看当前操作系统平台

sys.version: 查看当前Python解释器版本

正则

元字符	匹配内容
\w	匹配字母（包含中文）或数字或下划线
\W	匹配非字母（包含中文）或数字或下划线
\s	匹配任意的空白符
\S	匹配任意非空白符
\d	匹配数字
\D	匹配非数字
^	从字符串开头匹配
\$	匹配字符串的结束，如果是换行，只匹配到换行前的结果
\n	匹配一个换行符
\t	匹配一个制表符
^	匹配字符串的开始
\$	匹配字符串的结尾
.	匹配任意字符，除了换行符，当re.DOTALL标记被指定时，则可以匹配包括换行符的任意字符。
[...]	匹配字符组中的字符
...	匹配除了字符组中的字符的所有字符
*	匹配0个或者多个左边的字符。
+	匹配一个或者多个左边的字符。
?	匹配0个或者1个左边的字符，非贪婪方式。
{n}	精准匹配n个前面的表达式。
{n,m}	匹配n到m次由前面的正则表达式定义的片段，贪婪方式
ab	匹配a或者b
()	匹配括号内的表达式，也表示一个组

findall 返回的是列表，使用re.findall()

finditer：返回的是一个迭代器

```
s = re.finditer("\w", "alex:dsb,wusir.djb")
for i in s:      # print(next(s).group())多次输出，继续写
print(i.group())
```

遇到的坑：

[^0-9]: 取非0-9之间的数字

[0-9]: 取0-9之间的数字

[-0-9]: 取负号, 然后0-9的数字

(a*): 匹配*左侧字符串0次或多次, 俗称贪婪匹配, 默认最后留一个空的保留位

("a+"): 匹配左侧字符串一次或多次, 贪婪匹配

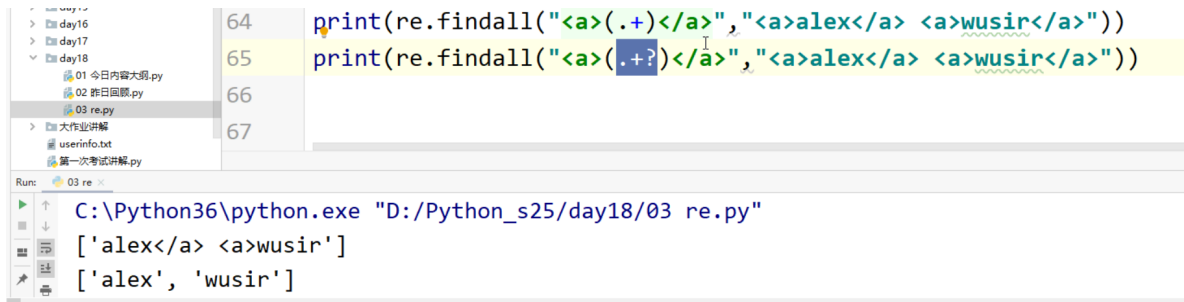
(?): 问号左右两遍任意1个或0个内容

a{3}: 匹配a出现3次的内容, 数字代表匹配次数, a代表匹配的内容, 程序中主要应用于匹配手机号等。

a{1,8}: 数字代表最少1个最多8个之间的所有

a | b: 匹配a或者b

() : 匹配括号内的表达式, 也表示一个组, 拿到的是括号里面的内容



面试题: search 和 match

search 拿到的一个对象, 取值在后面加.group(), 只要字符串中有符合规则的就开始查找, 任意位置
match 必须从头开始查找, 开头不符合规则, 不查找

分割: split

```
print(re.split("[:;,.!#]", "alex:dsb,wusir.djb"))
```

输出结果: ['alex', 'dsb', 'wusir', 'djb']

#分割的符号必须放在中括号内

替换: sub

```
s = "alex:dsb,wusir.djb"
```

```
print(re.sub('d', 'r', s)) #有次数限制
```

输出结果: 把d替换成r

字母, 数字, 下划线

```
print(re.findall("\w", "宝元-alex_dsb123日魔吃D烧饼")) #\w 字母.数字.下划线.中文
```

输出结果: ['宝', '元', 'a', 'l', 'e', 'x', '_', 'd', 's', 'b', '1', '2', '3', '日', '魔', '吃', 'D', '烧', '饼']

```
print(re.findall("\W", "宝元-alex_dsb123日魔吃D烧饼")) #
```

输出结果: ['-']

数字


```
print(re.findall("\d", "+10"))
```

#\d 匹配数字

输出结果: ['1', '0']

```
print(re.findall("\D", "+10"))
```

#\D 匹配非数字

输出结果: ['+', '@']

(点) 的用法:

```
print(re.findall("a.c", "abc,aec,a\nc,a,c"))
```

匹配任意一个字符串(\n除外)

输出结果: ['abc', 'aec', 'a,c']

```
print(re.findall("a.c", "abc,aec,a\nc,a,c", re.DOTALL))
```

输出结果: ['abc', 'aec', 'a\nc', 'a,c']

[]的用法:

[0-9] # 取0-9之前的数字

[^0-9] # 取非 0-9之间的数字

```
print(re.findall("[^0-9a-z]", "123alex456"))
```

```
print(re.findall('[0-9]', "alex123,日魔dsb,小黄人_229"))
```

输出结果: ['1', '2', '3', '2', '2', '9']

```
print(re.findall('[a-z]', "alex123,日魔DSB,小黄人_229"))
```

输出结果: ['a', 'l', 'e', 'x']

```
print(re.findall('[A-Z]', "alex123,日魔DSB,小黄人_229"))
```

输出结果: ['D', 'S', 'B']

※ 的用法: 匹配*左侧字符串0次或多次 贪婪匹配 ***

```
print(re.findall("a*", "alex,aa,aaaa,bbbbaaa,aaabbbaaa"))
```

匹配*左侧字符串0次或多次 贪婪匹配 ***

输出结果: ['a', '', '', '', '', 'aa', '', 'aaaa', '', '', '', '', '', 'aaa', '', 'aaa', '', '', '', 'aaa', '']

+: 匹配左侧字符串一次或多次 贪婪匹配

```
print(re.findall("a+", "alex,aa,aaaa,bbbbaaa,aaabbbaaa"))
```

输出结果: ['a', 'aa', 'aaaa', 'aaa', 'aaa', 'aaa']

? 的用法: 匹配?号左侧0个或1个 非贪婪匹配 ***

{}(大括号)的用法: 指定查找的元素个数 ***

点 (.) + 和 点 (.) +? 的区别:

```
print(re.findall("<a>(.)</a>", "<a>alex</a> <a>wusir</a>"))
```

分组

输出结果: ['alex <a>wusir']

```
print(re.findall("<a>(.*?)</a>", "<a>alex</a> <a>wusir</a>"))
```

控制贪婪

匹配

输出结果: ['alex', 'wusir']

包

- 管理模块的，文件化管理

定义：

- 文件夹下具有 `__init__.py` 的文件夹就是一个包、

路径：

1. 绝对路径：从包的最外层进行查找，就是绝对路径
 - `import 包. 模块` `from 包.包 import 模块`
2. 相对路径：. 是当前位置，..是上一级位置，...上上级位置
 - 同级目录不能使用相对路径（不能使用import）
 - `from ..包 import 模块`，都会自动触发 `__init__.py`

在启动文件启动包，包里导入了包中同级模块，需要添加到sys.path中

日志 (logging)

1. 记录用户的信息
2. 记录个人流水
3. 记录软件的运行状态
4. 跳板机：记录程序员发出的指令
5. 用于程序员调试
6. 用法：（）内写的都是日志里记录的信息，默认从warning开始记录

```
logging.basicConfig(level=10)    #配置信息
logging.debug("你是疯子")        #debug调试
logging.info("疯疯癫癫")        #info信息
logging.warning("缠缠绵绵")     #warning警告
logging.error("错误信息")        #error错误
logging.critical("你走吧")       # 危险
```

手动挡

```
import logging
logging.basicConfig(
    level=logging.DEBUG,
    format='%(asctime)s %(filename)s[line:%(lineno)d] %(levelname)s %(message)s',
    datefmt='%Y-%m-%d %H:%M:%S',
    filename="test.log",
    filemode="a",
)
```

```
logging.debug("你是疯儿,我是傻") # debug 调试
logging.info("疯疯癫癫去我家")   # info 信息
logging.warning("缠缠绵绵到天涯") # info 警告
logging.error("我下不床")          # error 错误
logging.critical("你回不了家")      # critical 危险
```