学习目标：

1. 知道什么是正则表达式
2. 掌握re模块的核心用法
3. 能独立完成字符串的匹配、提取、替换、验证等实际需求

什么是正则表达式

正则表达式的定义：

正则表达式（Regular Expression，简称 “regex”）是一种**用于描述字符串模式的 “规则语言”**，通过特殊符号组合，快速实现字符串的**「匹配、提取、替换、分割」**操作。

为什么需要正则表达式?

在实际开发过程中经常会有查找符合某些规则的字符串 比如：邮箱、图片地址、手机号码等。想匹配或者查找符合某些规则的字符串就可以使用正则表达式了。

比如：

1. 需求：在一个文件中，验证用户输入的邮箱格式是否合法（含@、.等关键符号）
2. 需求：从文本中批量提取所有手机号（11 位数字，特定开头）
3. 需求：将文本中所有 “敏感词” 统一替换为 `*`

注意：正则表达式并不是Python所特有的，在Java、PHP、Go以及JavaScript等语言中都是支持正则表达式的。

re模块操作

在Python中需要通过正则表达式对字符串进行匹配时，可使用re模块。

使用流程

re模块使用三步走

```python
# 第一步：导入re模块
import re

# 第二步：使用match方法(或其他方法)进行匹配操作
result = re.match(pattern正则表达式, string要匹配的字符串, flags=0)
#flags ： 可选，表示匹配模式，比如忽略大小写，多行模式等

# 第三步：如果数据匹配成功，使用group方法来提取数据
result.group()
```

案例-1

```python
# 1. 导入
import re

# 2. 匹配
original_data = 'wangdao@cskaoyan.com'
result = re.match('wangdao',original_data)

# 3. 提取
# 如果上一步匹配到数据的话，可以使用group来提取数据
result_data = result.group()

# 输出: wangdao
# 说明: match方法能够匹配以指定字符串开头的数据
print(result_data)
```

Python 通过内置的re模块实现正则功能，掌握以下 5 个基础函数，覆盖 80% 日常场景.

- match
- search
- findall
- sub
- split

match

语法格式：re.match(pattern, text, flags=0)

- 功能：从字符串**开头**开始匹配，仅验证 “开头是否符合规则”
- **返回值**：匹配成功返回Match对象，失败返回None
- **参数**：
  - pattern：正则规则字符串（必加r原始字符串，避免转义）

- text：待匹配的目标字符串
- flags：匹配模式（如re.IGNORECASE忽略大小写，后续进阶讲）

案例-2：验证手机号格式（完整字符串匹配）

```
import re
# 规则：11位数字，以13/14/15/17/18开头，且是完整字符串

# ^1：表示以1开头
# [34578]：第二位是3,4,5,7,8其中的一个
# \d：匹配任意数字
# {9}：任意数字出现9次
# $：结束符，表示到此结束
pattern = r"^1[34578]\d{9}$"
text1 = "13812345678"  # 合法手机号
text2 = "12345678901"  # 非法手机号（开头不符）

print(re.match(pattern, text1))  # 输出：<re.Match object; ...>（成功）
print(re.match(pattern, text2))  # 输出：None（失败）
```

search

语法格式：re.search(pattern, text, flags=0)

- 功能：从字符串**任意位置**匹配，找到“第一个符合规则的内容”即停止
- **区别match**：match只看开头，search遍历整个字符串

案例-3：提取文本中第一个手机号

```
text = "我的手机号：13812345678，备用号：13987654321"
pattern = r"1[34578]\d{9}"  # 不限制位置，只匹配手机号格式

result = re.search(pattern, text)
if result:
    print("第一个手机号：", result.group())  # 输出：第一个手机号：13812345678
    print("位置：", result.span())           # 输出：位置：(6, 17)（起始/结束索引）
```

findall

语法格式：re.findall(pattern, text, flags=0)

- **功能**：从字符串中找到“所有符合规则的内容”，返回列表
- **返回值**：列表（元素为匹配的字符串，若有分组则返回分组内容）

案例-4：批量提取所有手机号

```
text = "我的手机号：13812345678，备用号：13987654321"
pattern = r"1[34578]\d{9}"

phones = re.findall(pattern, text)
print("所有手机号：", phones)  # 输出：所有手机号：['13812345678', '13987654321']
```

sub

语法格式：re.sub(pattern, repl, text, count=0, flags=0)

- **功能**：将匹配到的内容替换为repl（字符串或函数）
- **参数**：count=0表示“替换所有匹配内容”，count=1表示“只替换第一个”

案例-5：敏感词替换

```
text = "这个内容是垃圾，不要传播垃圾信息"
pattern = r"垃圾"
new_text = re.sub(pattern, "*", text)  # 替换所有“垃圾”为*
print(new_text)  # 输出：这个内容是*，不要传播*信息
```

案例-6：手机号格式美化

```
text = "13812345678"

# ()：表示分组匹配，将内部字符视为一个整体，后续可以提取
pattern = r"(\d{3})(\d{4})(\d{4})"  # 分3个分组（前3/中4/后4位）

new_text = re.sub(pattern, r"\1 \2 \3", text)  # \1代表第1个分组
print(new_text)  # 输出：138 1234 5678
```

split

语法格式：re.split(pattern, text, maxsplit=0, flags=0)

- **功能**：用“匹配到的内容”作为分隔符，分割字符串，返回列表
- **参数**：maxsplit=0表示“全部分割”，maxsplit=1表示“只分割一次”

案例-7：按“空格 / 逗号 / 分号”分割字符串

```
text = "apple banana,orange;grape"
pattern = r"[ ,;]"  # 匹配空格、逗号、分号中的任意一个

result = re.split(pattern, text)
print("分割结果：", result)  # 输出：分割结果：['apple', 'banana', 'orange',
'grape']
```

正则表达式的编写

接下来，看一下正则表达式的具体语法规则。

注意：规则有很多，不用死记硬背，把常见的掌握，能看懂，用的时候会查就行。

匹配单个字符

通过符号匹配“单个字符”，核心符号及用法如下：

字符	功能	示例	匹配结果
.	匹配任意1个字符（除了\n）	`a.b`	`aab`、`acb`、`a1b`（不匹配`ab`）
[ ]	匹配[ ]中列举的字符	`a[0-9]b`	`a0b`、`a5b`（不匹配`aab`）
\d	匹配数字，即0-9,等价于[0-9]	`a\d`	`a1`、`a9`（不匹配`aa`）
\D	匹配非数字，即不是数字	`a\D`	`aa`、`ab`（不匹配`a1`）
\s	匹配空白，即 空格，tab键 space	`a\sb`	`a b`、`a\tb`（不匹配`aab`）
\S	匹配非空白	`a\Sb`	`aab`、`a1b`（不匹配`a b`）
\w	匹配字母、数字、下划线，即a-z、A-Z、0-9、_	`\w\w`	`a1`、`_3`、`Ab`（不匹配`@#`）
\W	匹配非单词字符	`\W`	`@`、`#`、空格（不匹配`a1`）

案例-8：匹配“字母 + 任意数字 + 字母”的字符串

```
pattern = r"[a-zA-Z]\d[a-zA-Z]"
texts = ["a1b", "B2C", "x3y", "1ab"]
for text in texts:
    if re.match(pattern, text):
        print(f"{text}: 匹配成功")  # 输出a1b、B2C、x3y
    else:
        print(f"{text}: 匹配失败")  # 输出1ab
```

匹配多个字符

通过“量词”控制“前面的字符”匹配的次数，核心量词如下：

字符	功能	示例	匹配结果（符合的字符串）
*	匹配前一个字符出现0次或者无限次，即可有可无	a*b	b、ab、aaab
+	匹配前一个字符出现1次或者无限次，即至少有1次	a+b	ab、aaab
?	匹配前一个字符出现1次或者0次，即要么有1次，要么没有	a?b	b、ab
{n}	匹配前一个字符出现n次	a{3}b	aaab
{m,n}	匹配前一个字符出现从m到n次	a{1,2}b	ab、aab

案例-9：匹配“1 个或多个数字”（如年龄）

```
pattern = r"\d+"  # 1次或多次数字
texts = ["20", "3", "abc", "123"]
for text in texts:
    print(f"{text}: {re.match(pattern, text) is not None}")
# 输出:
# 20: True
# 3: True
# abc: False
# 123: True
```

案例-10：匹配“3-6 位字母”（如用户名）

```
pattern = r"[a-zA-Z]{3,6}"
texts = ["abc", "abcd12", "ab", "abcdef"]
for text in texts:
    print(f"{text}: {re.match(pattern, text) is not None}")
# 输出:
# abc: True
# abcd12: False（含数字）
# ab: False（不足3位）
# abcdef: True（6位）
```

匹配开头结尾

通过“定位符”指定“匹配内容在字符串中的位置”，核心符号如下：

字符	功能	示例	匹配结果（符合的字符串）
^	匹配字符串开头	^abc	匹配 abc123（开头是 abc），不匹配 123abc

$	匹配字符串结尾	`abc$`	匹配 `123abc`（结尾是 abc），不匹配 `abc123`

案例-11：验证完整字符串是邮箱（必须以邮箱格式开头或结尾）

```
pattern = r"^[a-zA-Z0-9_\.]+@[a-zA-Z0-9]+\.[a-zA-Z]{2,4}$"
emails = ["user@163.com", "user@.com", "abc@123", "user@163.com.abc"]
for email in emails:
    if re.match(pattern, email):
        print(f"{email}: 合法邮箱")  # 仅user@163.com合法
    else:
        print(f"{email}: 非法邮箱")
```

匹配分组

通过()将“多个字符视为一个整体”，实现“复杂规则组合”和“结果提取”，核心用法如下：

字符	功能	示例	匹配结果
|	匹配左右任意一个表达式	`^[a-z]+$\|^[A-Z]+$`	`abc`、`ABC`匹配，不匹配`ABc`
()	将括号中字符作为一个分组	`(ab)+`	`ab`、`abab`（1 次或多次 ab），不匹配`aab`
\num	引用分组num匹配到的字符串，（替换时用）	`(a)(b)\1\2`	`abab`（\1 是 a，\2 是 b，组合为 abab）

案例-12：提取邮箱中的用户名和域名

```
pattern = r"(\w+)@(\w+\.\w+)"  # 分组1：用户名，分组2：域名
email = "user123@example.com"

result = re.match(pattern, email)
if result:
    print("用户名：", result.group(1))  # 输出：用户名：user123
    print("域名：", result.group(2))    # 输出：域名：example.com
    print("完整邮箱：", result.group(0)) # 输出：完整邮箱：user123@example.com
```

re模块的进阶操作(了解)

1．re.compile（pattern，flags=0）：预编译正则

- 功能：将正则规则预编译为Pattern对象，后续多次使用时提升效率（避免重复解析规则）
- **适用场景**：同一正则规则需要匹配多次（如循环处理大量文本）

案例-13

```
import re
# 预编译正则规则（只编译一次）
pattern = re.compile(r"1[34578]\d{9}")

# 多次使用预编译的Pattern对象
texts = ["文本1：13812345678", "文本2：13987654321", "文本3：abc123"]
for text in texts:
    result = pattern.search(text)  # 直接用Pattern对象调用search
    if result:
        print(f"{text}：提取到手机号：{result.group()}")
```

2．flags 参数：匹配模式控制

flags用于修改正则的匹配行为，常用值如下：

- re.IGNORECASE（简称re.I）：忽略大小写匹配
- re.DOTALL（简称re.S）：让.匹配换行符\n（默认.不匹配\n）
- re.MULTILINE（简称re.M）：让^和$匹配“每行的开头和结尾”（默认只匹配整个字符串的开头结尾）

案例-14：忽略大小写匹配hello

```
pattern = r"hello"
text = "Hello HELLO hello"

# 不忽略大小写：只匹配小写hello
print(re.findall(pattern, text))           # 输出：['hello']

# 忽略大小写：匹配所有大小写形式
print(re.findall(pattern, text, re.I))     # 输出：['Hello', 'HELLO',
'hello']
```

3．如何查找第二个

search只能查找第一个，如果要查找第二个，那么需要使用finditer外加next组合，next是迭代器走到下一个的含义

```
def find_second_match(pattern, text):
    matches = re.finditer(pattern, text)
    try:
        next(matches)  # 跳过第一个匹配项
        second_match = next(matches)  # 获取第二个匹配项
        return second_match.group()
    except StopIteration:
        return None



text = "abc123def456ghi789"
pattern = r"\d+"
```

```
second_match = find_second_match(pattern, text)
print(second_match)
```

贪婪与非贪婪

核心定义：

- **贪婪匹配**：默认行为，量词（`*` `+` `{n,}`）会“尽可能多”地匹配字符
- **非贪婪匹配**：在量词后加?（如`*?` `+?` `{n,}?`），会“尽可能少”地匹配字符

贪婪量词	非贪婪量词	含义	示例文本	贪婪匹配结果	非贪婪匹配结果
*	*?	0 次或多次	aabab	aabab	aab
+	+?	1 次或多次	aabab	aabab	aa
{n,}	{n,}?	至少 n 次	aaabbb	aaabbb	aaa

案例-15：从HTML标签中匹配 内的内容

```
import re
text = "<div>内容1</div><div>内容2</div>"

# 1. 贪婪匹配（.*尽可能多匹配，从第一个<div>到最后一个</div>）
greedy_pattern = r"<div>.*</div>"
print("贪婪匹配：", re.findall(greedy_pattern, text))
# 输出：贪婪匹配：['<div>内容1</div><div>内容2</div>']（匹配整个字符串）

# 2. 非贪婪匹配（.*?尽可能少匹配，从第一个<div>到最近的</div>）
non_greedy_pattern = r"<div>.*?</div>"
print("非贪婪匹配：", re.findall(non_greedy_pattern, text))
# 输出：非贪婪匹配：['<div>内容1</div>', '<div>内容2</div>']（匹配两个独立标签）
```

总结

1. 什么是正则表达式？

   是一种**用于描述字符串模式的“规则语言”**，通过特殊符号组合，快速实现字符串的**「匹配、提取、替换、分割」**操作

2. 为什么需要正则表达式？

   通过正则表达式，可以快速实现字符串的**「匹配、提取、替换、分割」**操作

3. 如何使用正则表达式？

   在Python中，可以使用 `re` 模块，使用流程如下：

   - 导入模块
   - 使用模块中的方法对数据进行操作
   - 提取数据

4. `re` 模块有哪些方法，各有什么作用？

   - match：从字符串**开头**开始匹配，仅验证 “开头是否符合规则”
   - search：从字符串**任意位置**匹配，找到 “第一个符合规则的内容” 即停止
   - findall：从字符串中找到 “所有符合规则的内容”，返回列表
   - sub：将匹配到的内容替换为指定字符串
   - split：用 “匹配到的内容” 作为分隔符，分割字符串，返回列表

5. 正则表达式有哪些匹配规则？

   - 匹配单个字符

     `.` `[]` `\d` `\s` `\w`

   - 匹配多个字符

     `*` `+` `?` `{n}` `{m,n}`

   - 匹配开头结尾

     `^` `$`

   - 匹配分组

     `|` `()` `\num`

6. 需不需要记住每一个正则表达式的规则？

   不需要，熟悉常用的规则即可，以后用到了，会查即可

补充说明：正则表达式的工具网站有很多，我们可以通过工具网站查询常用的正则表达式

常用网站1

常用网站2

当然今天因为AI非常强大，可以快速帮你编写合适的正则表达式，因此上面的两个网站可以不看