Python 的 `re` 模块提供了正则表达式操作的核心功能。以下是常用方法和示例详解：

---

### \*\*1. 基本匹配方法\*\*

#### (1) `re.match(pattern, string)`

- \*\*功能\*\*：从字符串\*\*开头\*\*匹配正则表达式（若开头不匹配则失败）

- \*\*返回值\*\*：成功返回 `Match` 对象，失败返回 `None`

- \*\*示例\*\*：

```python

import re

result = re.match(r'\d+', '123abc') # 匹配开头数字

if result:

print(result.group()) # 输出: 123

```

#### (2) `re.search(pattern, string)`

- \*\*功能\*\*：扫描整个字符串，找到\*\*第一个\*\*匹配项

- \*\*示例\*\*：

```python

result = re.search(r'\d+', 'abc456def')

print(result.group()) # 输出: 456

```

#### (3) `re.findall(pattern, string)`

- \*\*功能\*\*：返回所有匹配结果的\*\*列表\*\*

- \*\*示例\*\*：

```python

numbers = re.findall(r'\d+', '12 apples, 5 bananas')

print(numbers) # 输出: ['12', '5']

```

#### (4) `re.finditer(pattern, string)`

- \*\*功能\*\*：返回所有匹配结果的\*\*迭代器\*\*（适合处理大量数据）

- \*\*示例\*\*：

```python

matches = re.finditer(r'\d+', 'a1b22c333')

for match in matches:

print(match.group())

# 输出: 1 → 22 → 333

```

---

### \*\*2. 字符串替换\*\*

#### `re.sub(pattern, repl, string)`

- \*\*功能\*\*：替换所有匹配项

- \*\*示例\*\*：

```python

text = re.sub(r'\s+', '-', 'Hello World') # 替换连续空格为短横

print(text) # 输出: Hello-World

```

---

### \*\*3. 字符串分割\*\*

#### `re.split(pattern, string)`

- \*\*功能\*\*：按正则模式分割字符串

- \*\*示例\*\*：

```python

parts = re.split(r'[\s,;]+', 'apple, banana; cherry')

print(parts) # 输出: ['apple', 'banana', 'cherry']

```

---

### \*\*4. 正则表达式编译\*\*

- \*\*优化技巧\*\*：重复使用同一正则时，先编译可提升效率

```python

pattern = re.compile(r'\b[A-Z]+\b') # 匹配全大写单词

result = pattern.findall('HELLO world PYTHON')

print(result) # 输出: ['HELLO', 'PYTHON']

```

---

### \*\*5. 分组捕获\*\*

- \*\*使用 `()` 捕获分组内容\*\*

```python

text = 'John: 30, Alice: 25'

matches = re.findall(r'(\w+): (\d+)', text)

print(matches) # 输出: [('John', '30'), ('Alice', '25')]

```

---

### \*\*6. 常用正则标志\*\*

- `re.IGNORECASE`（`re.I`）：忽略大小写

- `re.MULTILINE`（`re.M`）：多行模式（影响 `^` 和 `$`）

- `re.DOTALL`（`re.S`）：让 `.` 匹配包括换行符的所有字符

\*\*示例\*\*：

```python

text = 'Name: Anna\nAge: 30'

result = re.search(r'^age: (\d+)', text, re.I | re.M)

print(result.group(1)) # 输出: 30

```

---

### \*\*注意事项\*\*

1. \*\*贪婪匹配\*\*：默认使用贪婪模式（尽可能多匹配），在量词后加 `?` 改为非贪婪模式

```python

re.match(r'<.\*?>', '<div>text</div>').group() # 输出: <div>

```

2. \*\*原始字符串\*\*：建议使用 `r''` 定义正则表达式，避免转义问题

```python

re.match(r'\d+\.\d+', '3.14') # 正确匹配小数

```

3. \*\*特殊字符转义\*\*：如匹配 `.` 需写成 `\.`

---

通过结合这些方法和正则表达式语法，可以实现复杂的文本处理逻辑。遇到具体匹配问题时，建议使用 [regex101.com](https://regex101.com/) 在线工具进行调试。