

정규표현식

강사장철원

□ 정규 표현식

정규표현식

Section 1. 정규표현식 기초

Section 1-1. 정규 표현식 개념

정규 표현식(regular expressions, regex or regexp)

Sequence of characters that specifies a match pattern in text

텍스트에서 특정 패턴을 매칭할 때 사용하는 기법

정규 표현식(regular expressions, regex or regexp)

이름에서 두번째 글자와, 전화번호에서 가운데 네자리를 마스킹 처리해주세요.

홍길동 010-1234-5678

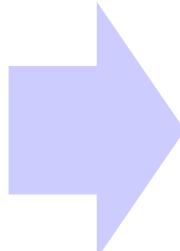
홍*동 010-****-5678

임꺽정 010-4321-8765

임*정 010-****-8765

성춘향 010-2345-6543

성*향 010-****-6543



정규 표현식없이 파일로 구현

```
[1]: text = """홍길동 010-1234-5678
임꺽정 010-4321-8765
성춘향 010-2345-6543
"""
```

```
[2]: print(text)
```

홍길동 010-1234-5678
 임꺽정 010-4321-8765
 성춘향 010-2345-6543

```
[3]: result = []
```

```
[4]: for line in text.splitlines():
    print(line)
    print('-----')
```

홍길동 010-1234-5678

 임꺽정 010-4321-8765

 성춘향 010-2345-6543

```
[5]: for line in text.splitlines():
    name, phone = line.split()
    print(name, phone)
    print('-----')
```

홍길동 010-1234-5678

 임꺽정 010-4321-8765

 성춘향 010-2345-6543

정규 표현식없이 파일 코드로 구현

```
[6]: for line in text.splitlines():
    name, phone = line.split()
    re_name = name[0] + '*' + name[-1]
    print(re_name, phone)
```

홍*동 010-1234-5678

임*정 010-4321-8765

성*향 010-2345-6543

```
[7]: for line in text.splitlines():
    name, phone = line.split()
    re_name = name[0] + '*' + name[-1]
    phone_part = phone.split('-')
    print(phone_part)
```

['010', '1234', '5678']

['010', '4321', '8765']

['010', '2345', '6543']

```
[8]: for line in text.splitlines():
    name, phone = line.split()
    re_name = name[0] + '*' + name[-1]
    phone_part = phone.split('-')
    re_phone = phone_part[0] + '-****-' + phone_part[2]
    print(re_phone)
```

010-****-5678

010-****-8765

010-****-6543

```
[9]: for line in text.splitlines():
    name, phone = line.split()
    re_name = name[0] + '*' + name[-1]
    phone_part = phone.split('-')
    re_phone = phone_part[0] + '-****-' + phone_part[2]
    new_info = re_name + ' ' + re_phone
    print(new_info)
```

홍*동 010-****-5678

임*정 010-****-8765

성*향 010-****-6543

정규 표현식없이 파이썬 코드로 구현

```
[10]: for line in text.splitlines():
    name, phone = line.split()
    re_name = name[0] + '*' + name[-1]
    phone_part = phone.split('-')
    re_phone = phone_part[0] + '-****-' + phone_part[2]
    new_info = re_name + ' ' + re_phone
    result.append(new_info)
```

```
[11]: print(result)
['홍*동 010-****-5678', '임*정 010-****-8765', '성*향 010-****-6543']
```

```
[12]: for person in result:
    print(person)
```

```
홍*동 010-****-5678
임*정 010-****-8765
성*향 010-****-6543
```

정규 표현식을 활용한 파이썬 코드

```
[17]: text = """홍길동 010-1234-5678  
임꺽정 010-4321-8765  
성춘향 010-2345-6543  
"""
```

```
[18]: import re  
  
pattern = r'(\w)(\w+)(\w)\s(010)-(\d{4})-(\d{4})'  
replacement = r'\1*\3 \4-****-\6'  
re_text = re.sub(pattern, replacement, text)  
  
# 결과 출력  
print(re_text)
```

홍*동 010-****-5678
임*정 010-****-8765
성*향 010-****-6543

이게 어떻게 가능할까?

Section

정규 표현식 개념

정규 표현식 연습 사이트 <https://regex101.com/>

The screenshot shows the homepage of regex101.com. The top navigation bar includes icons for back, forward, search, and a logo, followed by the URL 'regex101.com'. Below the header, the text 'regular expressions 101' is displayed. On the left side, there's a sidebar with various icons and options:

- SAVE & SHARE**: Includes 'Save new Regex' (ctrl+s) and 'Add to Community L...'.
- FLAVOR**: A list of regex engines:
 - </> PCRE2 (PHP >=7.3)
 - </> PCRE (PHP <7.3)
 - </> ECMAScript (JavaScri...)
 - </> Python** (highlighted with a yellow box and checked)
 - </> Golang
 - </> Java 8
 - </> .NET 7.0 (C#)
 - </> Rust
 - _REGEX FLAVOR GUIDE

The main workspace is titled 'REGULAR EXPRESSION' and contains the placeholder text ':r" insert your regular expression here'. To the right of the input field are buttons for 'no match', 'gm', and a copy icon. Below this is a 'TEST STRING' input field with the placeholder 'insert your test string here'.

메타문자 []

[와] 사이의 문자들 중 하나와 매치

[abc] = abc 중 하나의 글자와 매치

The screenshot shows the regex101.com interface. In the top right, there is red text: "abcde 중 하나의 문자와 매치". The main area displays a regular expression pattern `: r" [abcde]` and a test string `Jang•Cheolwon`. The Python flavor is selected. The sidebar on the left includes icons for code, clipboard, user, and settings.

regular expressions 101

abcde 중 하나의 문자와 매치

REGULAR EXPRESSION

: r" [abcde]

TEST STRING

Jang•Cheolwon

SAVE & SHARE

- Save new Re... **ctrl+s**
- Add to Community L...

FLAVOR

- </> PCRE2 (PHP >=7.3)
- </> PCRE (PHP <7.3)
- </> ECMAScript (JavaScri...
- </> **Python** ✓

메타문자 [] 실습

```
[20]: import re

text = 'Jang Cheolwon'
pattern = r'[abcde]'
matches = re.findall(pattern, text)
print(matches)
```



```
['a', 'e']
```

The screenshot shows the regex101.com interface. In the top right, the URL 'regex101.com' is visible. The main area displays the text 'regular expressions 101'. On the left, there's a sidebar with icons for file operations and sharing. Below that is a 'SAVE & SHARE' section with options to 'Save new Re...' (using `ctrl+s`) and 'Add to Community L...'. Further down is a 'FLAVOR' section listing several engines: PCRE2 (PHP >=7.3), PCRE (PHP <7.3), ECMAScript (JavaScript), and Python, with Python being the selected flavor indicated by a green checkmark. To the right, the 'REGULAR EXPRESSION' field contains the pattern `:r" [abcde]`. The 'TEST STRING' field contains the string `Jang•Cheolwon`, where the dot character is highlighted in blue.

r"과 f"의 차이

r' '(raw string) - 백슬래시(\)를 이스케이프 문자로 사용하지 않고 있는 그대로 사용

```
[23]: path = "C:\\Users\\gildong\\Documents"  
      print(path)
```

```
path = r"C:\\Users\\gildong\\Documents"  
      print(path)
```

C:\\Users\\gildong\\Documents

C:\\Users\\gildong\\Documents

r"과 f"의 차이

f' '(formatted string) - 문자열에 변수를 삽입할 때 사용

```
[24]: name = "Alice"  
      age = 30  
  
      # f-string 사용  
      sentence = f"My name is {name} and I am {age} years old."  
      print(sentence)
```

My name is Alice and I am 30 years old.

메타문자 []

```
[25]: import re

text = 'Jang Cheolwon'
pattern = r'[a-z]'
matches = re.findall(pattern, text)
print(matches)

['a', 'n', 'g', 'h', 'e', 'o', 'l', 'w', 'o', 'n']
```

The screenshot shows a web-based regular expression testing tool. At the top, the URL 'regex101.com' is visible. The main interface includes sections for 'regular expressions' and 'TEST STRING'. In the 'REGULAR EXPRESSION' section, the pattern `: r" [a-z]` is displayed. In the 'TEST STRING' section, the input `Jang•Cheolwon` is shown. On the left side, there's a sidebar with icons for file operations like 'Save new Re...', 'Add to Community L...', and a 'FLAVOR' dropdown. The 'Python' flavor is selected, indicated by a green checkmark.

메타 문자 -

^는 not이라는 의미

[a-z] 모든 소문자

[^a-z] 소문자 제외

[a-zA-Z] 모든 대소문자

[^a-zA-Z] 대소문자 제외

[0-9] 모든 숫자

[^0-9] 숫자 제외

메타 문자 -

| 문법 | 의미 | 동일 문법 |
|----|-----------------|----------------|
| \d | 숫자와 매치 | [0-9] |
| \D | 숫자가 아닌 것과 매치 | [^0-9] |
| \s | 공백과 매치 | [\t\n\r\f\v] |
| \S | 공백이 아닌 문자와 매치 | [^ \t\n\r\f\v] |
| \w | 문자+숫자와 매치 | [a-zA-Z0-9_] |
| \W | 문자+숫자가 아닌 것과 매치 | [^a-zA-Z0-9_] |

메타문자 * = 0부터 무한대까지 반복

REGULAR EXPRESSION

: r" ca*t

TEST STRING

cat

REGULAR EXPRESSION

: r" ca*t

TEST STRING

caaaaaat

REGULAR EXPRESSION

: r" ca*t

TEST STRING

ct

c와 t사이에 a가 반복

a가 여러번 반복되어도 OK

a가 포함되지 않아도 OK
(0번 반복이라고 인식)

Section

정규표현식개념

메타 문자 + = 1부터 무한대까지 반복

REGULAR EXPRESSION

```
:r" ca+t
```

TEST STRING

```
cat
```

c와 t사이에 a가 반복

REGULAR EXPRESSION

```
:r" ca+t
```

TEST STRING

```
caaaaaat
```

a가 여러번 반복되어도 OK

REGULAR EXPRESSION

```
:r" ca+t
```

TEST STRING

```
ct
```

a가 포함되지 않으면 안됨
(*과 +의 차이점)

메타문자 {m, n} - 반복 횟수가 m이상 n이하

REGULAR EXPRESSION

```
:r" ca{2}t
```

TEST STRING

```
caat
```

c와 t사이에 a를 정확히 두번 반복

REGULAR EXPRESSION

```
:r" ca{2}t
```

TEST STRING

```
caaat
```

세 번 반복하면 안됨

Section

정규표현식개념

메타문자 {2, 5} - 반복 횟수가 2이상 50이하

REGULAR EXPRESSION
: r" ca{2,5}t

TEST STRING
cat

한 번 반복 안됨

REGULAR EXPRESSION
: r" ca{2,5}t

TEST STRING
caat

두 번 반복 됨

REGULAR EXPRESSION
: r" ca{2,5}t

TEST STRING
caaat

세 번 반복 됨

REGULAR EXPRESSION
: r" ca{2,5}t

TEST STRING
caaaaat

다섯 번 반복 됨

REGULAR EXPRESSION
: r" ca{2,5}t

TEST STRING
aaaaaaaaat

여섯 번 반복 안됨

Section

정규 표현식 개념

메타문자 ? = {0,1}

c와 t 사이에 a가 없거나 한 번 있거나

REGULAR EXPRESSION

```
:r" ca?t
```

TEST STRING

```
ct
```

0번 포함 OK

REGULAR EXPRESSION

```
:r" ca?t
```

TEST STRING

```
cat
```

한 번 포함 OK

REGULAR EXPRESSION

```
:r" ca?t
```

TEST STRING

```
caat
```

두 번 포함 NO

메타문자 () - 그룹화

```
[38]: import re
```

```
text = "example@gmail.com"
pattern = r"\w+\@\w+\.\w+"
match = re.findall(pattern, text)
print(match)
```



```
['example@gmail.com']
```

```
[39]: import re
```

```
text = "example@gmail.com"
pattern = r"(\w+)(@)(\w+)(\.)(\w+)"
match = re.findall(pattern, text)
print(match)
```

```
[('example', '@', 'gmail', '.', 'com')]
```

```
[40]: import re
```

```
text = "example@gmail.com"
pattern = r"(\w+)\@(\w+)\.( \w+)"
match = re.findall(pattern, text)
print(match)
```

```
[('example', 'gmail', 'com')]
```

정규 표현식을 활용한 파이썬 코드

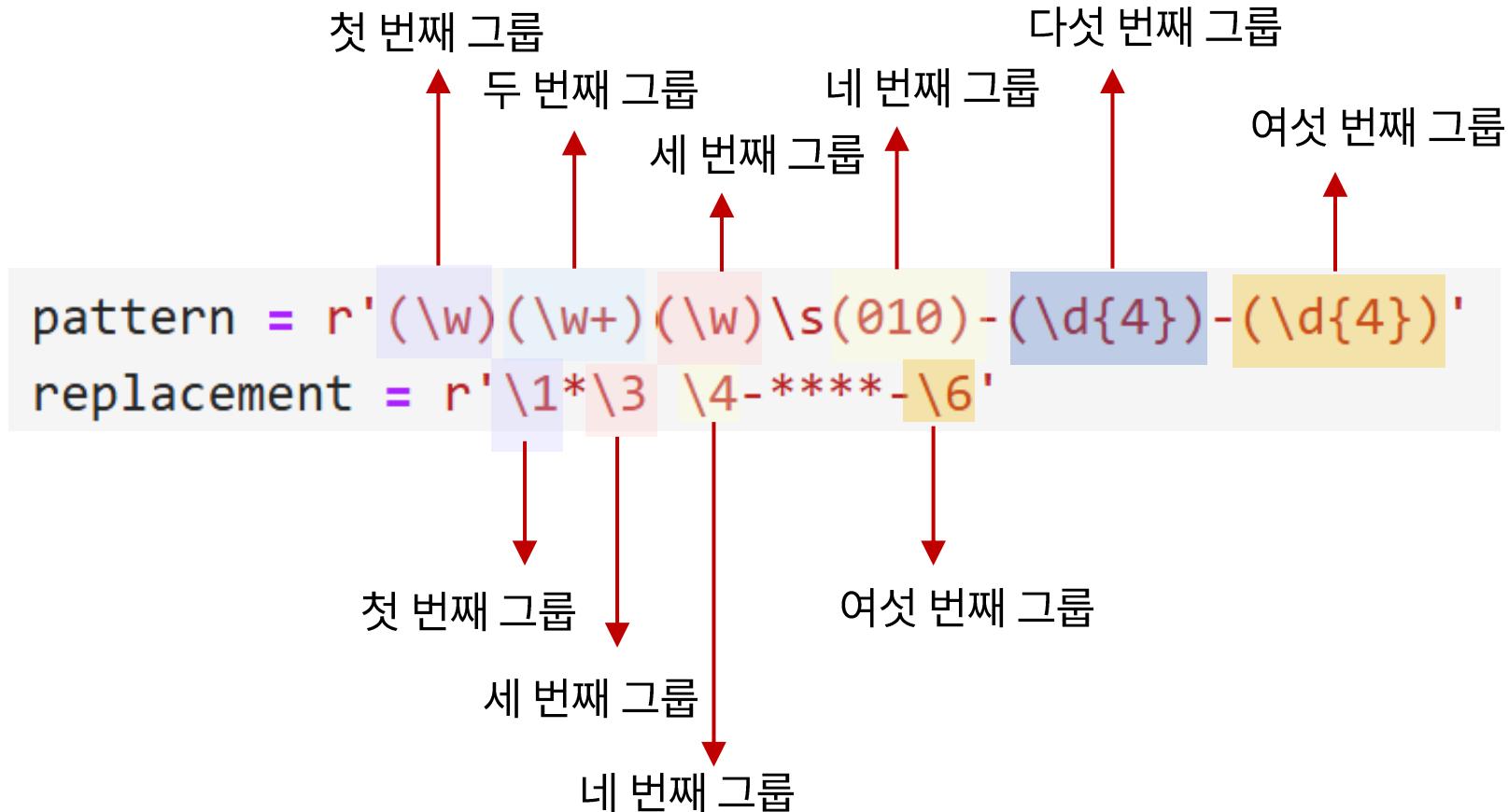
```
[17]: text = """홍길동 010-1234-5678  
임꺽정 010-4321-8765  
성춘향 010-2345-6543  
"""
```

```
[18]: import re  
  
pattern = r'(\w)(\w+)(\w)\s(010)-(\d{4})-(\d{4})'  
replacement = r'\1*\3 \4-****-\6'  
re_text = re.sub(pattern, replacement, text)  
  
# 결과 출력  
print(re_text)
```

홍*동 010-****-5678
임*정 010-****-8765
성*향 010-****-6543

이게 어떻게 가능할까?

정규 표현식을 활용한 파이썬 코드



감사합니다.

Q & A