4-4. 텍스트 데이터를 문자열로 저장한다는 것 (3) 정규 표현식

어떤 특정한 단어를 검색할 때 자주 사용하는 `Ctrl+F` , 거의 모든 운영 체제가 이 단어 검색 기능을 지원하고 있는데요. 이러한 기능이 바로 **정규 표현식** (regular expression, regex)에 근거해 만들어진 기능이랍니다. 정규표현식은 이처럼 대부분의 운영 체제, 다양한 프로그래밍 언어, 그리고 텍스트 편집기와 같은 프로그램들이 지원을 하고 있습니다.

문자열 vs 정규 표현식

자, `'I can do it!'` 이라는 문장에서 `I` 를 `You` 로 바꾸고 싶습니다. 어떻게 하면 될까요? 가장 간단한 방법으로 앞에서 다룬 `replace()` 메소드를 이용할 수 있습니다.

```
sent = 'I can do it!'
sent.replace("I", "You")
```

또 다른 방법이 바로 '정규 표현식'을 이용하는 것입니다. 한번 살펴 볼까요?

[Input]

```
import re
sent = 'I can do it!'
pattern = re.sub("I", "You", sent)
pattern
```

실행

[Output]

정규 표현식은 우리가 찾고자 하는 문자열 패턴을 정의하고 기존 문자열과 일치하는지를 비교하는 것입니다. 세상에는 패턴화된 문자열이 굉장히 많습니다. 이메일, 주민등록번호, 전화번호, 우편번호, URL 등등 … 정말 방대하죠. 이럴 때마다 문자열 메소드를 사용하면 굉장히 버거울 거예요.

위 그림에서 Source 문자열과 Pattern 문자열의 단어를 잘 기억해 보세요. 자, 이제부터 본격적으로 시작해 보겠습니다.

정규 표현식 시작하기

import re

파이썬에서는 표준 라이브러리인 `re` 모듈을 `import` 해서 정규 표현식을 사용할 수 있습니다.

```
import re
```

Compile()

정규 표현식의 사용법은 크게 1) 찾고자 하는 문자열의 패턴을 정의하고 2) 정의된 패턴과 매칭되는 경우를 찾아 다양한 처리를 하는 2단계로 나누어집니다. 이중 1)에 해당하는 과정을 컴파일(compile)이라고 합니다.

아래는 위 두 단계에 대한 간략한 예시코드입니다.

[Input]

```
#1단계 :  "the"라는 패턴을 컴파일한 후 패턴 객체를 리턴합니다.
pattern = re.compile("the")

# 2단계 : 컴파일된 패턴 객체를 활용하여 다른 텍스트에서 검색을 수행합니다.
pattern.findall('of the people, for the people, by the people')
```

실행

[Output]

하지만, 정규 표현식을 활용하기 위해 꼭 명시적으로 re.compile()으 호출해야 하는 것은 아닙니다. 아래 코드는 위에서 2단계로 수행했던 내역을 명시적 컴파일 과정 없이 한줄로 동일하게 처리하고 있습니다.

[Input]

```
re.findall('the', 'of the people, for the people, by the people')
```

실행

[Output]

아래쪽 경우가 간결해서 더 좋은 것 같습니다. 하지만 명시적으로 패턴 객체를 생성한 경우 해당 객체를 반복 사용 가능하다는 점이 장점이 될 때가 있습니다.

메소드

정규 표현식의 객체를 컴파일했으니 이제 위 2단계에서 패턴 객체를 활용해서 호출 가능한 메소드와 속성을 알아 볼 차례입니다. 많이 사용되고 있는 메소드들을 정리해 보았는데요. 한번 가볍게 살펴 봅시다.

- `search()` : 일치하는 패턴 찾기 * 일치 패턴이 있으면 MatchObject를 반환합니다.
- `match()` : `search()` 와 비슷하지만, 패턴이 검색대상에 처음부터 일치해야 합니다.
- `findall()` : 일치하는 모든 패턴 찾기 * 모든 일치 패턴을 리스트에 담아서 반환합니다.
- `split()` : 패턴으로 나누기
- `sub()` : 일치하는 패턴으로 대체하기

아래는 `search()`, `match()` 등이 리턴하는 MatchObject가 가진 메소드입니다.

- `group()` : 실제 결과에 해당하는 문자열을 반환합니다.

[Input]

```
src = "My name is..."
regex = re.match("My", src)
if regex:
   print(regex.group())
else:
   print("No!")
```

실행

[Output]

패턴 : 특수문자, 메타문자

패턴이야말로 정규 표현식을 강력하게 해주는 도구 인데요. 특수문자 혹은 메타문자라 불리는 `[]. -. . ? * + {} /` 등을 이용해 특수한 패턴을 만들 수 있습니다. 자, 어떤 것들이 있는지 볼까요?

- `[ ]` : 문자
- `-` : 범위
- `.` : 하나의 문자
- `?` : 0회 또는 1회 반복
- `*` : 0회 이상 반복
- `+` : 1회 이상 반복
- `{m, n}` : m ~ n
- `\d` : 숫자
- `\D` : 비숫자
- `\w` : 알파벳 문자
- `\W` : 비알파벳 문자
- `\s` : 공백 문자
- `\S` : 비공백 문자
- `\b` : 단어 경계
- `\B` : 비 단어 경계

훑어만 보아도 이 많은 것들을 당장 외우기는 힘들어 보이네요. 그래서 이 중 몇 가지를 실습해 볼 예제를 준비해 보았습니다. 아래 실습을 통해 정규 표현식의 패턴들을 차근차근 익혀보도록 해요!

예제

[Input]

```
#- 연도(숫자)
text = """
The first season of America Premiere League  was played in 1993.
The second season was played in 1995 in South Africa.
Last season was played in 2019 and won by Chennai Super Kings (CSK).
CSK won the title in 2000 and 2002 as well.
Mumbai Indians (MI) has also won the title 3 times in 2013, 2015 and 2017.
"""
pattern = re.compile("[1-2]\d\d\d")
pattern.findall(text)
```

실행

[Output]

[Input]

```
#- 전화번호(숫자, 기호)
phonenumber = re.compile(r'\d\d\d-\d\d\d-\d\d\d\d')
phone = phonenumber.search('This is my phone number 010-111-1111')
if phone:
  print(phone.group())
print('------')
phone = phonenumber.match ('This is my phone number 010-111-1111')
if phone:
  print(phone.group())
```

실행

[Output]

[Input]

```
#- 이메일(알파벳, 숫자, 기호)
text = "My e-mail adress is doingharu@aiffel.com, and tomorrow@aiffel.com"
pattern = re.compile("[0-9a-zA-Z]+@[0-9a-z]+\.[0-9a-z]+")
pattern.findall(text)
```

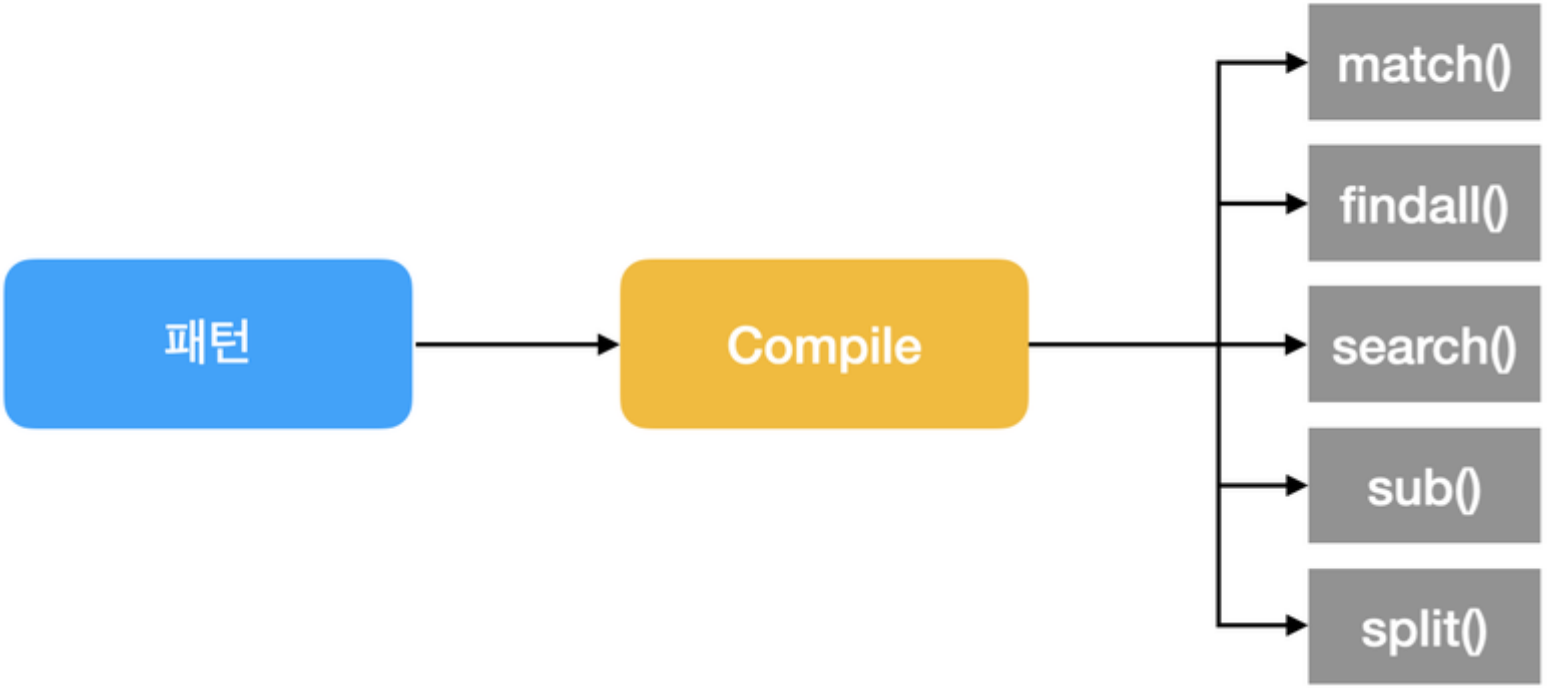
실행

[Output]

구현 순서

위 예제들에서 정규 표현식을 구현한 순서를 그림과 함께 간단히 정리하고 넘어가 봅시다. 패턴들이 덜 익숙한 것 뿐 정규식의 구현 과정은 매우 간단한 것을 확인하실 수 있을 거에요.

- `import re` 를 통해 정규식 모듈을 가져옵니다.
- `re.compile()` 함수로 Regex 객체를 만듭니다.
- 검색할 문자열을 Regex 객체의 `search()` , `findall()` 메소드로 전달합니다.


패턴
Compile
match()
findall()
search()
sub()
split()