파이썬 정규표현식
re 모듈

파이썬 re모듈

- 정규표현식(regular expression)
  - 특정한 규칙을 가진 문자열의 패턴을 표현하는 데 사용하는 표현식(Expression)
  - 복잡한 문자열을 처리할 때 사용
  - 텍스트에서 특정 문자열을 검색하거나 치환할 때 사용
  - 파이썬 뿐만 아니라 문자열을 처리하는 모든 곳에서 사용

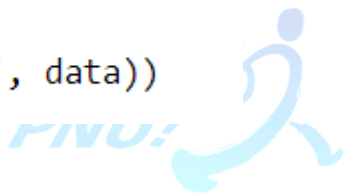
- 정규 표현식을 지원하기 위해 re모듈
  - re 모듈은 파이썬을 설치할 때 자동으로 설치되는 기본 라이브러리
  - 패턴과 매치하는 텍스트를 찾고 조작하는 기능을 제공

파이썬 정규 표현식

- **사용방법**
  - 컴파일 하는 방법
    - 검색할 패턴을 컴파일 하여 사용
    - 같은 문자를 여러 번 검색하는 경우에 사용하는 방법
    - 매번 검색 패턴을 지정하지 않아도 되기 때문에 검색 속도가 빠름
  - 컴파일 하지 않는 방법
    - 검색할 때마다 패턴을 설정하는 방법
    - 검색할 패턴이 매번 다른 경우에 사용
  - 검색 패턴에 raw문자인 r 사용
    - 문자열 안에 있는 백슬러시 문자를 특수한 의미를 가진(에스케이프) 문자가 아닌 백슬러시 문자 그대로 인식되기 때문

```
#컴파일하여 사용
pt = re.compile(r'10')
print(pt.search(data))
```

```
#컴파일하지 않고 사용
print(re.search(r'10', data))
```



파이썬 정규 표현식

- **검색 방법**
  - **match(pattern, string)**
    - 문자열의 시작부분부터 매칭이 되는지 검색.
    - matchObject 인스턴트로 반환
  - **search(pattern, string)**
    - 문자열에 패턴과 매칭되는 곳이 있는지 검색.
    - match() 함수와 차이점은 검색 대상 문자열의 시작 부분이 매칭되지 않아도 검색
    - 여러 개 매칭 시 처음에 검색된 부분을 반환
    - matchObject 인스턴트로 반환
  - **findall(pattern, string)**
    - 정규식과 매치되는 모든 문자열을 리스트로 반환.
  - **finditer(pattern, string)**
    - 정규식과 매치되는 모든 문자열을 iterator 객체로 반환.
    - 반환 값을 for문을 사용하여 모두 추출

```
#검색 방법
print(re.match('10', data))
print(re.search('10', data))
print(re.findall('10', data))
print(re.finditer('10', data))
```

```
None
<re.Match object; span=(11, 13), match='10'>
['10', '10']
<callable_iterator object at 0x000002168EF609B0>
```

파이썬 정규 표현식

- **정보 추출 방법**
  - group()
    - 매칭된 문자열.
  - start()
    - 매칭된 문자열 시작 위치.
  - end()
    - 매칭된 문자열 종료 위치.
  - span()
    - 매칭된 문자열 시작과 종료위치를 튜플로 반환.

```
# 검색정보 추출
data10 = re.search('10', data)
print(data10)
print(data10.group())
print(data10.start())
print(data10.end())
print(data10.span())

data10s = re.finditer('10', data)
for d in data10s:
    print(d)
    print(d.group())
    print(d.start())
    print(d.end())
    print(d.span())
```

```
<re.Match object; span=(11, 13), match='10'>
10
11
13
(11, 13)
<re.Match object; span=(11, 13), match='10'>
10
11
13
(11, 13)
<re.Match object; span=(16, 18), match='10'>
10
16
18
(16, 18)
```

파이썬 정규 표현식

- 메타 문자
  - 원래 그 문자가 가진 뜻이 아닌 특별한 용도로 사용하는 문자
    - . ^ $ * + ? { } [ ] ₩ | ( )
  - 정규 표현식에 메타 문자를 사용하면 특별한 의미를 갖게 됨

정규 표현식 (Regular Expression)

- **메타문자 []**
  - 문자 클래스(character class)
  - [ ] 사이의 문자들과 매치
    - 예) [abc] : a, b, c 중 한 개의 문자와 매치
  - [ ] 안의 두 문자 사이에 하이픈(-)을 사용
    - 두 문자 사이의 범위(From - To)를 의미
    - 예) [a-zA-Z] : 알파벳 모두, [0-9] : 숫자

```
# 메타문자 : [] 문자열 클래스
pt = '[a-z0-9]'
print(re.findall(pt, data))
```

```
['1', '0', '1', '0', 'm', 'o', 'r', 'n', '2', '0', '0', '5', '2', '4']
```

정규 표현식 (Regular Expression)

- 자주 사용하는 문자 클래스
  - ₩d
    - 숫자와 매치, [0-9]와 동일한 표현식
  - ₩D
    - 숫자가 아닌 것과 매치, [^0-9]와 동일한 표현식
  - ₩s
    - whitespace 문자와 매치
    - [₩t₩n₩r₩f₩v]와 동일한 표현식 : 맨 앞의 빈 칸은 공백문자(space)를 의미
  - ₩S
    - whitespace 문자가 아닌 것과 매치
  - ₩w
    - 문자+숫자(alphanumeric)와 매치, [a-zA-Z0-9_]와 동일한 표현식
  - ₩W
    - 문자+숫자(alphanumeric)가 아닌 문자와 매치, [^a-zA-Z0-9_]와 동일한 표현식

정규 표현식 (Regular Expression)

- **메타문자 : ^**
  - 반대(not)
  - 예) [^0-9]라는 정규 표현식은 숫자가 아닌 문자만 매치
- **메타문자 : Dot(.)**
  - 줄바꿈 문자인 ₩n을 제외한 모든 문자와 매치됨을 의미
    - 예) a.b : a와 b라는 문자 사이에 어떤 문자가 들어가도 모두 매치
  - 문자 클래스([]) 내에 Dot(.) 메타 문자가 사용
    - "모든 문자"라는 의미가 아닌 문자 . 그대로를 의미
    - 예) a[.]b :

정규 표현식 (Regular Expression)

- 메타문자 : 반복 (*)
  - 예) ab*c :
    문자 b가 0부터 무한대로 반복될 수 있다는 의미로 b가 한번도 나오지 않는 ac도 매치
- 메타문자 : 반복 (+)
  - 예) ab+c : 최소 1번 이상 반복될 때 사용하므로 ac는 매치가 되지 않음
- 메타문자:반복 {m,n}
  - 반복 횟수를 고정
  - {m, n} 정규식을 사용하면 반복 횟수가 m부터 n까지 매치
  - m 또는 n을 생략하면 m은 0과 동일하며, 생략된 n은 무한대의 의미
  - 예) ab{2}c : b가 2회만 반복되면 매치
  - 예) ab{2,5}c : b가 2회에서 5회까지 반복되면 매치

정규 표현식 (Regular Expression)

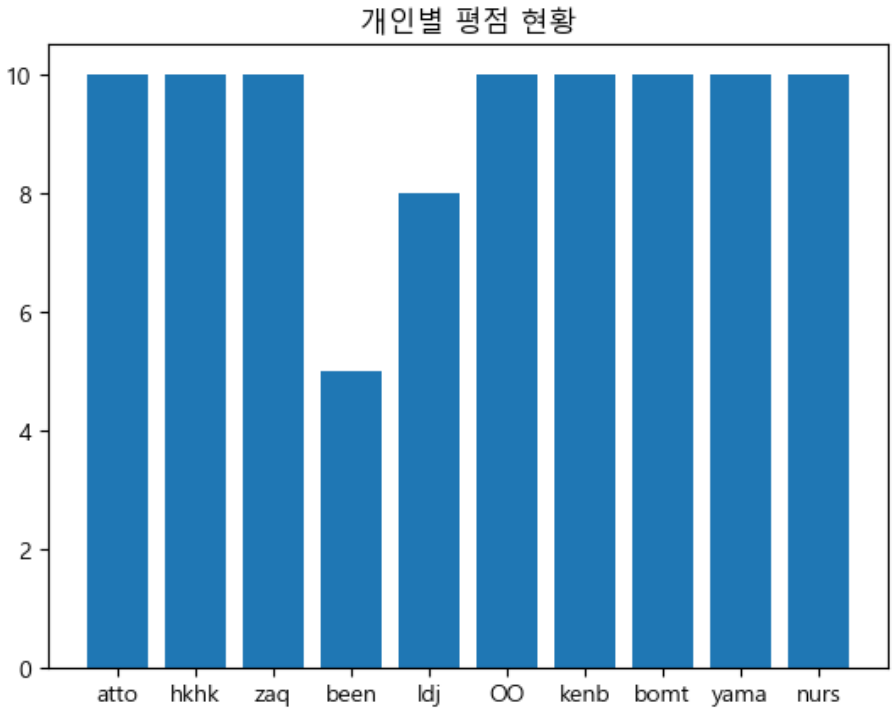
- **메타문자: ?**
  - 없거나 하나만 있으면 매치
  - 예) ab?c : b가 없거나 하나만 있는 경우 매치
- **메타문자:()**
  - 반환 값을 그룹 설정
  - group() 혹은 group(0)
    - 전체 가져오기
  - group(n)
    - 첫번째 그룹 가져오기

```
# 메타문자 반복
pt = r'm.'
print(re.search(pt, data).group(0))

pt = r'm[.]'
print(re.search(pt, data))

pt = r'm.r'
print(re.search(pt, data).group(0))

pt = r'[\d]+[.]'
print(re.search(pt, data).group())

pt = r'[\d]+[.]*'
print(re.search(pt, data).group())

pt = r'[\d]+[.]+'
print(re.search(pt, data).group())

pt = r'([a-z]+)([*]{1,4})'
print(re.search(pt, data).group())
print(re.search(pt, data).group(1))
print(re.search(pt, data).group(2))
```

```
mo
None
mor
20.
10
20.
morn****
morn
****
```

해결문제

- 영화 평점에 대한 정보가 있는 평점.txt 파일을 읽어서 개인별 평점 그래프를 작성하시오.



해결문제2

- 영화 평점에 대한 정보가 있는 평점.txt 파일을 읽어서 일자별 평점 평균 그래프를 작성하시오.