문제해결을 위한 데이터 분석 및 시각화

- JSON
- 정규 표현식

한성대학교 노은희 교수

"미래로 향하는 새로운 이정표"

오늘의 학습

학습목표

- JSON이해
- 정규 표현식 이해

JSON

JSON

JSON(Javascript Object Notation)으로 자바스크립트 객체 표기법으로 작성된 텍스트
Python에는 JSON 데이터 작업에 사용할 수 있는 내장 패키지가 있습니다

{}를 사용

예>표현
{“name” : “홍길동”, “age”:20}

파이썬으로 JSON 다루기

json 모듈을 가져오기

```
import json
```

```
1  import json
2
3  # JSON:
4  x = '{ "name":"홍길동", "age":20, "city":"서울"}'
5
6  print(x)
7
8  print(x["age"])
```

JSON객체는 키값으로 접근 불가

```
{ "name":"홍길동", "age":20, "city":"서울"}

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
C:\Users\Public\Documents\ESTsoft\CreatorTemp/ipykernel_3312/2367281842.py in <module>
      6 print(x)
      7
----> 8 print(x["age"])

TypeError: string indices must be integers
```

파이썬으로 JSON 다루기

json 모듈을 가져오기

```
import json
```

JSON 구문 분석 : JSON에서 Python으로 변환

`json.loads()` : JSON 문자열이 있는 경우 메서드를 사용하여 구문 분석

JSON

```
1  import json
2
3  # JSON:
4  x = '{ "name":"홍길동", "age":20, "city":"서울"}'
5
6  # parse x:
7  y = json.loads(x)
8
9  print(y)
10
11 # the result is a Python dictionary:
12 print(y["age"])
13
```

```
{'name': '홍길동', 'age': 20, 'city': '서울'}
20
```

파싱(parsing) :
구문 분석이라고 한다. 문장이 이루고 있는 구성 성분을 분해하고 분해된 성분의 위계 관계를 분석하여 구조를 결정하는 것

`json.loads()` : JSON 문자열이 있는 경우 메서드를 사용하여 구문 분석

키값으로 접근 가능

[참고] 인코딩과 디코딩 (Encoding & Decoding)]

문자 코드를 기준으로 문자를 코드로 변환하는 것을 **문자 인코딩(encoding)** 이라하고
코드를 문자로 변환하는 것을 **문자 디코딩(decoding)**

UTF-8 : 하나의 문자를 **1~4**바이트의 가변길이로 표현. **1**바이트 영역은 **ASCII**코드와 하위 호환되며 **ASCII**코드의 **128**개 문자 집합은 **UTF-8**과 동일하게 호환됨.
현재 인터넷에서 가장 많이 쓰이는 인코딩이며 뛰어난 크로스플랫폼 호환성도 갖고 있음.

정규 표현식

정규식 표현식(Regular Expressions)

- 정규식 모듈
- `re` Python에는 정규식으로 작업하는 데 사용할 수 있는 내장 패키지
- 정규식은 검색 패턴을 형성하는 일련의 문자
- 정규표현식은 복잡한 문자열을 처리하는데 사용하는 기법으로 문자열을 처리하는 곳에서 모두 사용

정규식 표현식(Regular Expressions)

정규표현식의 기초, 메타문자(메타 문자는 특별한 용도로 사용되는 문자)

```
.^*+?{}[]\|()
```

문자클래스 : []
- 문자클래스로 만들어진 정규식은 []안에 들어있는 문자들과 매치라는 의미
- []안의 두 문자 사이에 하이픈(-)이 사용되면 두 문자 사이의 범의를 의미(예[a-d]는[abcd]와 동일

<예> 정규표현식이 [abc]라면 'a,b,c'중 한 개의 문자와 매치

정규식	문자열	매치여부	설명
[abc]	a	Yes	'a'는 정규식과 일치하는 "a"가 존재함으로 매치
	bee	Yes	'bee'는 정규식과 일치하는 "b"가 존재함으로 매치
	dude	no	'dude'는 정규식과 일치하는 "a,b,c"중 하나도 포함하지 않기 때문에 매치되지 않음

- [a-zA-Z]:알파벳 모두
- [0-9] : 숫자

정규식 표현식(Regular Expressions)

정규표현식의 기초, 메타문자

```
.^*+?{}[]\|()
```

Dot(.)
줄바꿈 문자인 \n를 제외한 모든 문자와 매치
<예> a.b
설명 : a와 b사이에 줄바꿈 문자를 제외한 어떤 문자가 들어가도 모두 매치

정규식	문자열	매치여부	설명
a.b	aab	Yes	‘aab’는 가운데 문자“a”가 모든 문자를 의미함으로 정규식과 매치
	a3b	Yes	‘a3b’는 가운데 문자“3”가 모든 문자를 의미함으로 정규식과 매치
	abd	no	‘abd’ 가운데 문자“a”문자와 “b”문자 사이에 어떤 문자라도 하나가 있어야 하는 정규식과 일치하지 않기 때문에 매치하지 않음

정규식 표현식(Regular Expressions)

정규표현식의 기초, 메타문자

```
.^*+?{}[]\|()
```

반복(*)
- 반복의미, *바로 앞에 있는 문자가 0부터 무한 반복

정규식	문자열	매치여부	설명
be*t	bt	Yes	“b”가 0번 반복되어 매치
	beet	Yes	“b”가 0번 이상 반복되어 매치(2번 반복)
	bbbbbbt	yes	“b”가 0번 이상 반복되어 매치(5번 반복)

정규식 표현식(Regular Expressions)

정규표현식의 기초, 메타문자

```
.^*+?{}[]\|()
```

반복(+)
- 반복의미, +바로 앞에 있는 문자가 1부터 무한 반복

정규식	문자열	매치여부	설명
Be+t	bt	No	“b”가 0번 반복되어 매치되지 않음
	beet	Yes	“b”가 0번 이상 반복되어 매치(2번 반복)
	bbbbbbt	yes	“b”가 0번 이상 반복되어 매치(5번 반복)

정규식 표현식(Regular Expressions)

정규표현식의 기초, 메타문자

```
.^*+?{}[]\|()
```

반복({m,n},?)
{} 사용법

정규식	문자열	매치여부	설명
be{2}t	bet	no	“e”가 1번 반복되어 매치되지 않음
	beet	Yes	“e”가 2번 반복되어 매치

반복({m,n},?)
- 반복 횟수가 m부터 n까지, m과 n생략 가능(m은 생략하면 0, n은 생략하면 무한대)

정규식	문자열	매치여부	설명
be{2,5}t	bet	no	“e”가 1번 반복되어 매치되지 않음
	beet	Yes	“e”가 2번 반복되어 매치
	beeeeet	yes	“e”가 5번 반복되어 매치

```
remove_tag = re.compile('<.*?>')
```

정규식 표현식(Regular Expressions)

정규표현식의 기초, 메타문자

```
.^*+?{}[]\|()
```

?

- {0,1}의미(0~1번 사용되면 매치)

정규식	문자열	매치여부	설명
ab?c	ac	yes	“b”가 0번 사용되어 매치
	abc	Yes	“b”가 1번 사용되어 매치

Python에서 정규 표현식 사용하기

re모듈 가져오기

```
import re
p = re.compile('abc*') # re.compile을 이용하여 표현식('abc*')을 컴파일 한다.
```

Python에서 정규 표현식을 이용한 문자열 검색

- 컴파일된 패턴 객체는 문자열 검색을 위한 4가지 메서드 제공

메서드	
match()	문자열의 처음부터 정규식과 매치되는지 조사
search()	문자열 전체를 검색하여 정규식과 매치되는지 조사
findall()	정규식과 매치되는 모든 문자열을 리스트로 리턴
finditer()	정규식과 매치되는 모든 문자열을 반복 가능한 객체로 리턴

match, search는 정규식과 매치될때 match객체를 리턴하고, 매치가 되지 않으면 none리턴

match 객체의 매서드

메서드	
group()	매치된 문자열을 리턴
start()	매치된 문자열의 시작 위치를 리턴
end()	매치된 문자열의 끝 위치 리턴
span()	매치된 문자열의(시작,끝)에 해당되는 튜플을 리턴

[실습하기]

정규식 표현 사용

```
#1. 패턴 만들기
import re
p = re.compile('[a-z]+')    # 패턴 만들기
```

```
#2. 매치하기
m = p.match("python")       # match()메서드는 문자열의 처음부처 정규식과 매치되는지 조사.
print(m)                    # "python" 문자열은 [a-z]+ 정규식에 부합되므로 matct 객체 리턴
```

```
<re.Match object; span=(0, 6), match='python'>
```

```
m = p.match("7 python")
print(m)                  # "7 python" 문자열은 [a-z]+ 정규식에 부합되지 않으므로 None 리턴
```

```
None
```

```
# 1. 패턴 만들기
p2 = re.compile('[0-9a-z]+')    # 패턴 만들기
```

```
# 2. 매치하기
m = p2.match("7 python")
print(m)                    # "7 python" 문자열은 [0-9a-z]+ 정규식에 부합되지 않으므로 matct 객체 리턴
```

```
<re.Match object; span=(0, 1), match='7'>
```

문자열 바꾸기 : sub메서드

sub(substitute)
파이썬에서 정규 표현식을 사용하여 문자열을 치환
- sub메서드를 이용하면 정규식과 매치되는 부분을 다른 문자로 변경할 수 있다.

re.sub(정규 표현식, 대상 문자열, 치환 문자)

정규 표현식 – 검색 패턴을 지정
대상 문자열 – 검색 대상이 되는 문자열
치환 문자 – 변경하고 싶은 문자

```
1  import re
2  p = re.compile('[0-9]{3}-[0-9]{4}-[0-9]{4}')
3
4  text = """
5  010-1234-5678 Lee
6  010-5432-6543 Kim
7  010-4321-2365 Park
8  """
9  # 정규 표현식 사용 치환
10 text_mod = re.sub(p,"***-****-****",text)
11 print (text_mod)
```

```
***-****-**** Lee
***-****-**** Kim
***-****-**** Park
```

메타문자 [참고]

Character	Description	Example
[]	[]안에 있는 모든 문자	"[a-m]"
\	특수 시퀀스 신호(특수 문자를 이스케이프하는 데 사용할 수도 있음)	"\d"
.	모든 문자(except newline character)	"he..o"
^	시작	"^hello"
$	끝	"planet$"
*	0회 이상 발생	"he.*o"
+	하나 이상의 발생	"he.+o"
?	0또는 1회 발생	"he.?o"
{}	정확히 지정된 발생 횟수	"he.{2}o"
|	하나 또는	"falls|stays"
()	캡처 및 그룹화	