Fastcampus Web Programming SCHOOL

Python

Regular Expressions

Regular Expressions

- 특정한 규칙을 가진 문자열의 집합을 표현하는데 사용하는 형식언어
- Python은 re 모듈로 정규표현식 사용 가능
- 기본적인 문법은 비슷하나 언어별로 사용법이나 문법이 조금씩 달라 외우기 보다는 필요할 때 Reference를 참조하는 것이 낫습니다.

Regular Expressions - match()

```
import re
a = 'penpineapple'
b = 'applepen'

m = re.match('^pen', a)
n = re.match('^pen', b)

>>> m
<_sre.SRE_Match object; span=(0, 3), match='pen'>
>>> n
None
```

Regular Expressions - match object

```
>>> m.group()
'pen'
>>> m.start()
0
>>> m.end()
3
>>> m.span()
(0,3)

>>> a[m.start():m.end()]
'pen'
```

Regular Expressions - Meta Characters

- Dot(`.`): 개행문자(₩n)을 제외한 모든 문자를 의미합니다.
  ex) a.ple == a(모든문자)ple == aaple, abple, acple, ..
- Repetition(`*`): 앞 문자의 반복을 의미합니다.(0개 ~ 무한대)
  ex) ap*le == ale, aple, apple, appple, ..
- Repetition(`+`): 앞 문자의 반복을 의미합니다.(1개 ~ 무한대)
  ex) ap+le == aple, apple, appple, ..
- Caret(`^`): 정의한 문자열 패턴으로 시작함을 의미합니다.
  ex) ^pen == penpineapple (!= applepen)
- Dollar sign(`$`): 정의한 문자열 패턴으로 끝남을 의미합니다.
  ex) [a-z]+pen$ == applepen (!= penpineapple)

Regular Expressions - Meta Characters

- Question mark(`?`): 앞 문자의 존재유무(있거나 없거나)를 의미합니다.
  ex) appl?e == appe 또는 apple
- Curly brackets(`{m,n}`): 반복횟수를 정의할 때 사용합니다. 중괄호 안의 숫자(m,n)은 반복횟수를 정의하며, 정확히 일치할 경우 1개의 숫자(m)를, 범위를 정의할 경우 2개의 숫자를 사용합니다.(m개~n개)
  ex1) ba(na){2} == banana
  ex2) ba(na){0,2} == ba, bana, banana

Regular Expressions - Meta Characters

- Square brackets(`[]`): 문자클래스를 의미합니다. 대괄호 사이에 나열한 문자 중 하나의 값과 일치할 경우를 의미합니다. 이 문자클래스와 함께 쓰이는 메타문자는 하이픈(-)과 캐럿(^) 입니다.
  - Hyphen(-): 두 문자 사이의 범위를 뜻합니다.
    ex1) [A-Za-z] == 대문자와 소문자 알파벳 모두
    ex2) [a-g0-5] == a,b,c,d,e,f,g 와 0,1,2,3,4,5 중 하나의 문자
  - Caret(^): 부정(Not)을 의미합니다.
    ex1) [^0-9]: 숫자가 아닌 문자
    ex2) [^A-Za-z]: 알파벳이 아닌 모든 문자

Regular Expressions - Meta Characters

- Parentheses(`()`): 소괄호 안의 문자열의 그룹을 의미합니다. 문자열 패턴의 반복을 검사하고 싶을 때 사용합니다.
  ex) (apple)+ == penpineappleapplepen
- Vertical bar(`|`): | 의 좌우 중 하나의 값과 일치하는 경우를 의미합니다.
  ex1) (apple|apqle) == apple 또는 apqle
  ex2) ap[p|q]le == apple 또는 apqle
- Backslach(`\`): 정규표현식을 만들 때 메타문자가 메타문자가 아닌 그 문자 그대로 사용하게 하기 위해 사용합니다.
  ex) `https://www.google.co.kr/` => URL의 구두점(.)이 메타문자로 인식될 수 있음
  -> `https://www\.google\.co\.kr/`

Regular Expressions - HOW TO USE?

```
>>> p = re.compile('^pen')
>>> m = p.match(a)
>>> m
<_sre.SRE_Match object; span=(0, 3), match='pen'>

>>> m.group()
'pen'
```

- compile을 해두면 regular expression object를 재사용 할 수 있어 훨씬 효율적으로 운영할 수 있습니다.

re module functions

```
re.search(patterns, string[, flags])
re.match(patterns, string[, flags])
re.findall(patterns, string[, flags])
re.compile(pattern[, flags])
re.split(patterns, string[,maxsplit=0])
re.sub(pattern, repl, string[, count])
```

Regular Expressions - Extract text

`fake_info.csv`를 읽어 결과물을 리스트에 저장한 뒤, 정규표현식을 활용하여 다음 문제를 해결하세요.(str.split(',')을 사용하지 말 것)

```
name,email,age,state
```

1. 유저의 평균 나이를 조사하세요.
2. 이메일 도메인의 분포를 조사하세요.
   2-1. 도메인 점유율
   2-2. com과 net의 비율
3. 주거 주(state) 분포를 조사하세요.(주거 주의 리스트와 인원 수)

Regular Expressions - with Hangul

```
ㄱ-ㅎ
```

```
ㅏ-ㅣ
```

```
가-힣
```

Regular Expressions - with Hangul

다음 문자열 `이름:김패캠 전화번호:010-이삼사오-6789 사는곳:서울강동구 이메일:facampkim@gmail.com` 에서 이름과 전화번호, 사는곳, 이메일을 정규표현식을 활용해 추출하세요.

다음에 대해서도 동작하도록 구성하세요

`이름:박패캠 전화번호:0IO-2345-육칠팔구 사는곳:부산광역시 이메일:facamppark at gmail dot com`

lambda

lambda

- 익명함수(이름이 없는 함수)
- 간단한 수식을 함수로 지정해 한 두번 쓸 용도로 사용할 때
- 두 줄 이상 실행될 함수는 그냥 함수로 정의하는게 나음!

lambda

- python은 모든 것이 객체로 존재
- 간단한 연산 함수 조차 객체로 존재하여 리소스를 점유

그러나 남발하면..

- Code 자체의 Readibility를 해칠 뿐 아니라, 재사용에 대한 고민없이 사용하다 lambda를 반복사용하여 Heap을 괴롭힐 수 있습니다.

lambda - traditional function

```
def get_next_integer(a):
        return a + 1
```

lambda - lambda function

```
lambda a: a+1
```

lambda - usual example

```
>>> (lambda a: a+1)(12)
13
```

lambda

- only expression, not statement

```
lamb =
```

map, filter, reduce

map

- map(function, iter)
- list의 각 element에 대해 특정한 함수를 적용

map - example

```
def get_squared(num_list):
    squared = []
    for num in num_list:
        squared.append(num**2)
    return squared
```

map - example

```
def squared_lambda(x):
    return x ** 2

list(map(squared_lambda, [1,2,3,4]))
```

map with lambda - example

```
list(map(lambda x: x**2, [1,2,3,4]))
```

map is rather than for

```
def print_with_sleep(x):
    time.sleep(1)
    return x ** 2
```

```
m = map(print_with_sleep, [1,2,3])
next(m)
next(m)
next(m)
..
```

```
for i in range(1,3+1):
    print_with_sleep(i)
```

timing is perfect!

```
m = map(print_with_sleep, [1,2,3])
for i in m:
    print(i)
```

```
m = map(print_with_sleep, [1,2,3])
list(m)
```

map is rather than for

- map은 제너레이터를 생성해 함수와 인자를 바인딩만 하고, 필요할 때 순회하며 값을 처리
- for는 한번에 처리하므로 for 수행 중 다른 일을 할 수 없음

filter

- filter(function, iter)
- 특정함수를 만족하는 요소만 남기는 필터

filter - example

```
def even_selector(x):
    if x % 2 == 0:
        return True
    else:
        return False

filter(even_selector, range(1,10+1))
```

filter with lambda - example

```
filter(lambda x: x%2==0, range(1,10+1))
```

reduce

- reduce(function, iter[, initializer])
- iterable object의 모든 element에 대하여 연산결과를 출력
- python3 기본 내장함수에서 제외되어 functools에서 import함
    - why? Rossum은 map, filter, reduce에 대해 readibility가 떨어진다는 이유로 제외하고자 하였음

```
from functools import reduce
```

before reduce

3min
1부터 100까지 모든 숫자의 합을 반복문을 활용하여 연산하시오.

reduce example

```
result = 0
for i in range(1,100+1):
    result += element
```

reduce example

```
def adder(a,b):
    return a+b

reduce(adder, range(1,100+1))
```

reduce with lambda

```
reduce((lambda x,y:x+y), range(1,100+1)
```

reduce with initializer

```
default = 10
for i in range(1,10+1):
    dafault += i
```

```
reduce(lambda x,y:x+y, range(1,10+1), 10)
```

Deep dive into reduce

```
reduce(lambda x,y:y+x, range(1,10+1)
```

```
reduce(lambda x,y:y+x, 'fastcampus')
```

Do it yourself

```
recycle_bin = [
1, 2, "Fastcampus", ['dog', 'cat', 'pig'], 5, 4, 5.6, False
"패스트캠퍼스", 100, 3.14, 2.71828, {'name':'Kim'}, True,
]
```

1. 위 리스트의 요소 중 정수와 실수인 요소만 리스트로 구성하기
2. 위 리스트의 요소 중 정수만 각각 제곱하여 리스트로 구성하기
3. 위 리스트의 요소 중 정수만 각각 제곱한 수들의 합계 출력하기

> Hint: isinstance(1, int)

Do it yourself

Order ID	Quantity	Unit Price
181121001	2	2400
181121002	12	9800
181121003	5	124800
181121004	10	76000

- 앞서 배운 개념들을 활용해 아래 문제를 해결하세요.

1. 튜플과 리스트를 활용해 위 테이블을 변환하세요
2. 각 주문 별 총 주문가격을 산출하세요
3. 11월 21일에 발생한 총 매출, 평균 구매금액을 산출하세요