

문자열

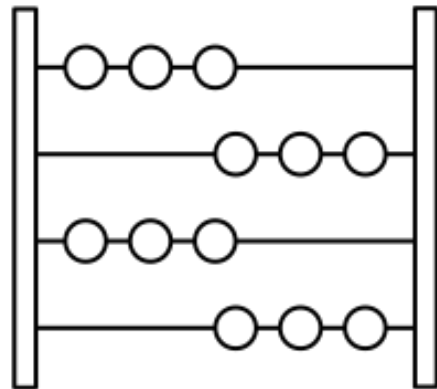
다음 두 가지가 익숙하게 생각이 든다면 프로그래밍을 잘하게 된거라고 감히 말씀드릴 수 있겠습니다.

1. 집합 데이터를 잘 다루는 것
2. 문자열 데이터를 잘 다루는 것

문자열을 다루는 것에 집중해서 프로그래밍을 배워봅시다.

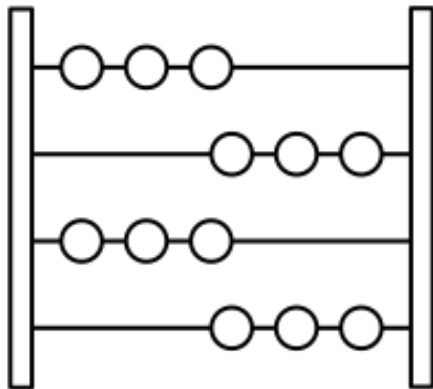


미션 1. 영문 문장에서 사용된 단어 개수 세기
어떤 전략이 있을까?



미션 1. 영문 문장에서 사용된 단어 개수 세기 어떤 전략이 있을까?

1. 문장을 단어로 분리
 - 특수문자는 제외
 - 단어들은 소문자로
2. 단어별로 카운팅
 - word dict를 만들고 카운트



문자열에 사용되는 주요 함수

- print: 문자열 출력
- len: 문자열 길이
- list: 각 문자의 리스트 생성

```
[60] 1 print("let it be")
```

```
✕ let it be
```

```
[61] 1 len("let it be")
```

```
↳ 9
```

```
[62] 1 list("let it be")
```

```
↳ ['l', 'e', 't', ' ', 'i', 't', ' ', 'b', 'e']
```

문자열 주요 메소드 1

- split: 문자열 분리
- join: 문자열 병합
- strip: 공백 문자열 제거

```
[50] 1 "let it be".split(" ")
```

```
↳ ['let', 'it', 'be']
```

```
▶ 1 " ".join(['let', 'it', 'be'])
```

```
↳ 'There is still a light that'
```

```
[20] 1 "_".join(['let', 'it', 'be'])
```

```
↳ 'There_is_still_a_light_that'
```

```
[51] 1 " let it be ".strip()
```

```
↳ 'let it be'
```

문자열 주요 메소드 2

- lower: 소문자로
- upper: 대문자로
- capitalize: 첫음절 대문자로

```
[26] 1 'LET IT BE'.lower()
```

```
↳ 'let it be'
```

```
▶ 1 'let it be'.upper()
```

```
↳ 'LET IT BE'
```

```
[27] 1 'let it be'.capitalize()
```

```
↳ 'Let it be'
```

문자열 주요 메소드 3

- find: 위치 찾기
- replace: 문자열 교체

```
[34] 1 'let it be'.find('it')
```

```
↳ 4
```

```
[35] 1 'let it be'.find('go')
```

```
↳ -1
```

```
[36] 1 'let it be'.replace('be', 'go')
```

```
↳ 'let it go'
```


미션 1. 영문 문장에서 사용된 단어 개수 세기

도전

1. 공백을 기준으로 단어를 분리
2. 단어별 개수 세기

추가 고민

- 특수문자들
- 대소문자

```
[74] 1 # 공백을 기준으로 단어들 분리  
      2 words = lyrics.split(' ')
```



```
1 # 단어별 개수 세기  
2 word_dict = {}  
3 for w in words:  
4     if w in word_dict:  
5         word_dict[w] += 1  
6     else:  
7         word_dict[w] = 1
```

미션 1. 영문 문장에서 사용된 단어 개수 세기

도전

1. 소문자로 변경
2. 특수문자 처리
3. 공백을 기준으로 단어를 분리
4. 단어별 개수 세기



```
1 # 모두 소문자로
2 lyrics = lyrics.lower()
3 # 특수 문자 확인
4 chars = set(lyrics) - set('abcdefghijklmnopqrstuvwxyz ')
5 # 특수 문자 제거
6 for c in chars:
7     lyrics = lyrics.replace(c, '')
8
9 # 공백을 기준으로 단어들 분리
10 words = lyrics.split(' ')
```

미션 2. 가장 많이 쓰이는 단어 10개 추출

전략

1. 단어들을 횃수로 정렬
2. 상위 10개만 추출



lambda 함수

- 잠시 쓰고 버리는 함수
- 문법
 - lambda 인자: 표현식
- sorted, map, filter 함수와 함께 사용하는 방법을 익혀두자
- lambda가 익숙해지면 함수형 언어에 대해 공부하자.

```
[97] 1 members = [{'id': 'a', 'point': 80},  
2             {'id': 'b', 'point': 70},  
3             {'id': 'c', 'point': 90},  
4             {'id': 'd', 'point': 60}]
```

```
[98] 1 list(map(lambda x: x['point'] * 2, members))
```

```
↳ [160, 140, 180, 120]
```

```
[99] 1 list(filter(lambda x: x['point'] > 70, members))
```

```
↳ [{'id': 'a', 'point': 80}, {'id': 'c', 'point': 90}]
```

```
▶ 1 sorted(members, key=lambda x: x['point'])
```

```
↳ [{'id': 'd', 'point': 60},  
    {'id': 'b', 'point': 70},  
    {'id': 'a', 'point': 80},  
    {'id': 'c', 'point': 90}]
```

미션 2. 가장 많이 쓰이는 단어 10개 추출

도전

1. 단어들을 리스트 형태로 변형
2. 단어들을 횟수로 정렬
3. 상위 10개만 추출

```
[118] 1 # 아래의 형태로 변형
      2 # [{'word': 단어1, 'count': 갯수},
      3 #    {'word': 단어2, 'count': 갯수},
      4 #    {'word': 단어3, 'count': 갯수}]
      5
      6 trans = [{'word': k, 'count': v} for k,v in word_dict.items()]
      7 trans
```

```
☞ [{'count': 3, 'word': 'when'},
   {'count': 1, 'word': 'i'},
   {'count': 1, 'word': 'find'},
   {'count': 1, 'word': 'myself'},
   {'count': 4, 'word': 'in'},
   {'count': 1, 'word': 'times'},
   {'count': 6, 'word': 'of'},
   {'count': 1, 'word': 'troublemother'},
   {'count': 1, 'word': 'marv'}
```

미션 2. 가장 많이 쓰이는 단어 10개 추출

도전

1. 단어들을 리스트 형태로 변형
2. 단어들을 횟수로 정렬
3. 상위 10개만 추출



```
1 # 정렬
2 trans = sorted(trans, key=lambda x: x['count'], reverse=True)
3 # 10개 추출
4 trans[:10]
```



```
[{'count': 10, 'w': 'it'},
 {'count': 8, 'w': 'let'},
 {'count': 7, 'w': 'be'},
 {'count': 6, 'w': 'of'},
 {'count': 4, 'w': 'in'},
 {'count': 4, 'w': 'will'},
 {'count': 3, 'w': 'when'},
 {'count': 3, 'w': 'words'},
 {'count': 3, 'w': 'wisdom'},
 {'count': 3, 'w': 'beand'}]
```

library konlpy 소개

미션 1. 한글 문장에서 사용된 단어 개수 세기

한글은 공백으로 나누어도 단어가 아닌데...
어떤 전략이 있을까?

<http://konlpy.org/ko/latest/>

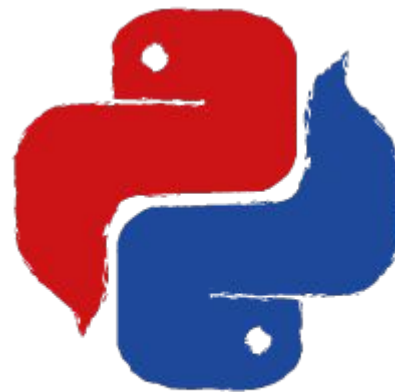
```
1 !pip install konlpy
```



Konlpy 알고리즘 종류 및 비교

<https://konlpy-ko.readthedocs.io/ko/v0.4.3/morph/>

- [Kkma](#)
- [Komoran](#)
- [Hannanum](#)
- [Twitter](#)
- [Mecab](#)



KoNLPy

Konlpy 주요 메소드

- sentences: 문장 분리
- nouns: 단어 분리
- pos: 품사 분석

형태소 분석 품사 태그표

```
[3]: from konlpy.tag import Kkma  
     kkma = Kkma()
```

```
[4]: kkma.sentences('네, 안녕하세요. 반갑습니다.')
```

```
[4]: ['네, 안녕하세요.', '반갑습니다.']
```

```
[7]: kkma.nouns('질문이나 건의사항은 깃헙 이슈 트래커에 남겨주세요.')
```

```
[7]: ['질문', '건의', '건의사항', '사항', '깃헙', '이슈', '트래커']
```

```
[8]: kkma.pos('오류보고는 실행환경, 에러메시지와함께 설명을 최대한상세히!^^')
```

```
[8]: [('오류', 'NNG'),  
      ('보고', 'NNG'),  
      ('는', 'JX'),  
      ('실행', 'NNG'),  
      ('환경', 'NNG'),  
      (',', 'SP'),  
      ('에러', 'NNG'),  
      ('메시지', 'NNG'),  
      ('와', 'JKM'),  
      ('함께', 'MAG'),  
      ('설명', 'NNG'),  
      ('을', 'JKO'),  
      ('최대한', 'NNG'),  
      ('상세히', 'MAG'),  
      ('!', 'SF'),  
      ('^^', 'EMO')]
```

미션 1. 한글 문장에서 사용된 단어 개수 세기

도전

1. konlpy를 이용하여 단어를 분리
2. 특수문자 처리
3. 단어별 개수 세기

```
from konlpy.tag import Kkma
```

```
# 형태소 분리
```

```
kkma = Kkma()
```

```
words = kkma.pos(lyrics)
```

```
# 특수 문자 제거
```

```
chars = ['SF', 'SP', 'SS', 'SE', 'SO', 'SW'] # 부호 태그
```

```
words = [w for w in words if w[1] not in chars]
```

```
# 단어별 개수 세기
```

```
word_dict = {}
```

```
for w in words:
```

```
    if w in word_dict:
```

```
        word_dict[w] += 1
```

```
    else:
```

```
        word_dict[w] = 1
```

미션 2. 가장 많이 쓰이는 단어 10개 추출

도전

1. 단어들을 리스트 형태로 변형
2. 단어들을 횡수로 정렬
3. 상위 10개만 추출

```
[25]: # 아래의 형태로 변형
      # [{'word': 단어1, 'count': 갯수},
      #    {'word': 단어2, 'count': 갯수},
      #    {'word': 단어3, 'count': 갯수}]

      trans = [{'word': k, 'count': v} for k, v in word_dict.items()]

[26]: # 정렬
      trans = sorted(trans, key=lambda x: x['count'], reverse=True)
      # 10개 추출
      trans[:10]

[26]: [{'word': ('이', 'VCP'), 'count': 11},
      {'word': ('꽃', 'NNG'), 'count': 10},
      {'word': ('어야', 'ECD'), 'count': 5},
      {'word': ('는', 'ETD'), 'count': 5},
      {'word': ('에서', 'JKM'), 'count': 4},
      {'word': ('여기', 'NP'), 'count': 3},
      {'word': ('서', 'JKM'), 'count': 3},
```

정규표현식

지금까지 배운 방법으로
문자열을 찾고, 바꾸고, 변환하는 것이
충분히 가능합니다. 그러나

곧, 다음의 불편함이 찾아오게 됩니다.

“문서에서 전화번호들을 추출하고 싶어.”

“문서에서 email들을 추출하고 싶어.”



정규표현식을 위한 도구

- regexone.com <https://regexone.com/>

정규 표현식 튜토리얼

- regexper <http://www.regexper.com/>

정규 표현식을 시각화해서 보여주는 도구

- regexr <http://www.regexr.com/>

정규 표현식에 대한 도움말과 각종 사례들을 보여주는 서비스로
정규표현식을 라이브로 만들 수 있는 기능도 제공하고 있다.

패턴 기본과 이스케이프

- 문자들 - ABCs
- 숫자들 - 123s
- a로 시작 - `^a`
- z로 종료 - `z$`
- `^`, `$` 검색 - `W^`, `W$` (`W`는 이스케이프 문자)

Task	Text	
Match	<code>cat.</code>	✓
Match	<code>896.</code>	✓
Match	<code>?=+. </code>	✓
Skip	<code>abc1</code>	
<input +\\."\$"="" type="text" value="."/>		Continue >

모든 문자 그룹

- 숫자 모두 - `\d`
- 숫자 제외한 모든 문자 - `\D`
- 알파벳과 숫자 모두 - `\w`
- 알파벳과 숫자 제외한 모든 문자 - `\W`
- 모든 문자 (특수문자 포함) - `.`

Task	Text	
Match	<code>abc123xyz</code>	✓
Match	<code>define "123"</code>	✓
Match	<code>var g = 123;</code>	✓

[Continue >](#)

특정 문자와 범위

- a, b, c 중 하나 - [abc]
- a, b, c 제외한 문자 하나 - [^abc]
- a부터 m까지 문자 중 하나 - [a-m]
- 0부터 9까지 숫자 중 하나 - [0-9]
- 한글 전체; [가-힣]

Task	Text	
Match	Ana	✓
Match	Bob	✓
Match	Cpc	✓
Skip	aax	
Skip	bby	
Skip	ccz	

[Continue >](#)

수량자

- a 10개 연속 - $a\{10\}$
- a 5개~10개 연속 - $a\{5, 10\}$
- a가 0개 이상 연속 - a^*
- a가 1개 이상 연속 - a^+
- a가 1개 있거나 없거나 - $a?$

Task	Text	
Match	aaaabcc	✓
Match	aabbbbc	✓
Match	aacc	✓
Skip	a	

Continue >

서브패턴과 매칭그룹

- 문자열 매칭 그룹 - () 소괄호
- 문자열 중 하나 매칭 그룹 - (abc|def)
- 하위 매칭 그룹 - (a(bc))

Exercise 11: Matching Groups

Task	Text	Capture Groups
Capture	file_record_transcript.pdf	file_record_transcript ✓
Capture	file_07241999.pdf	file_07241999 ✓
Skip	testfile_fake.pdf.tmp	

(.+).pdf\$

Continue >

전방탐색, 후방탐색

- 긍정형 전방탐색 - (?=)
- 부정형 전방탐색 - (?!)
- 긍정형 후방탐색 - (?<=)
- 부정형 후방탐색 - (?<!)

Example

```
\d(?:=px)
```

```
1pt 2px 3em 4px
```

Example

```
\d(?:!px)
```

```
1pt 2px 3em 4px
```

library re

정규 표현식을 지원하기 위해 기본 라이브러리로 re 모듈을 제공

주요 메소드

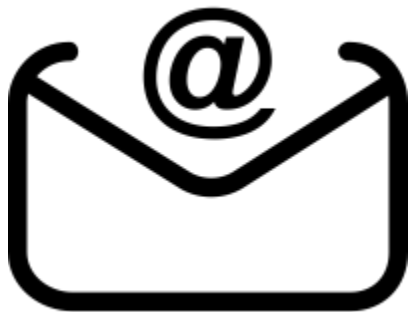
- match: 문자열 전체가 정규식과 매치되는지 확인
- search: 정규식과 매치되는 문자열이 있는지 확인
- findall: 정규식과 매치되는 모든 문자열(substring)을 리스트로 반환
- sub: 패턴을 찾아서 교체함.

```
import re  
p = re.compile('ab*')
```

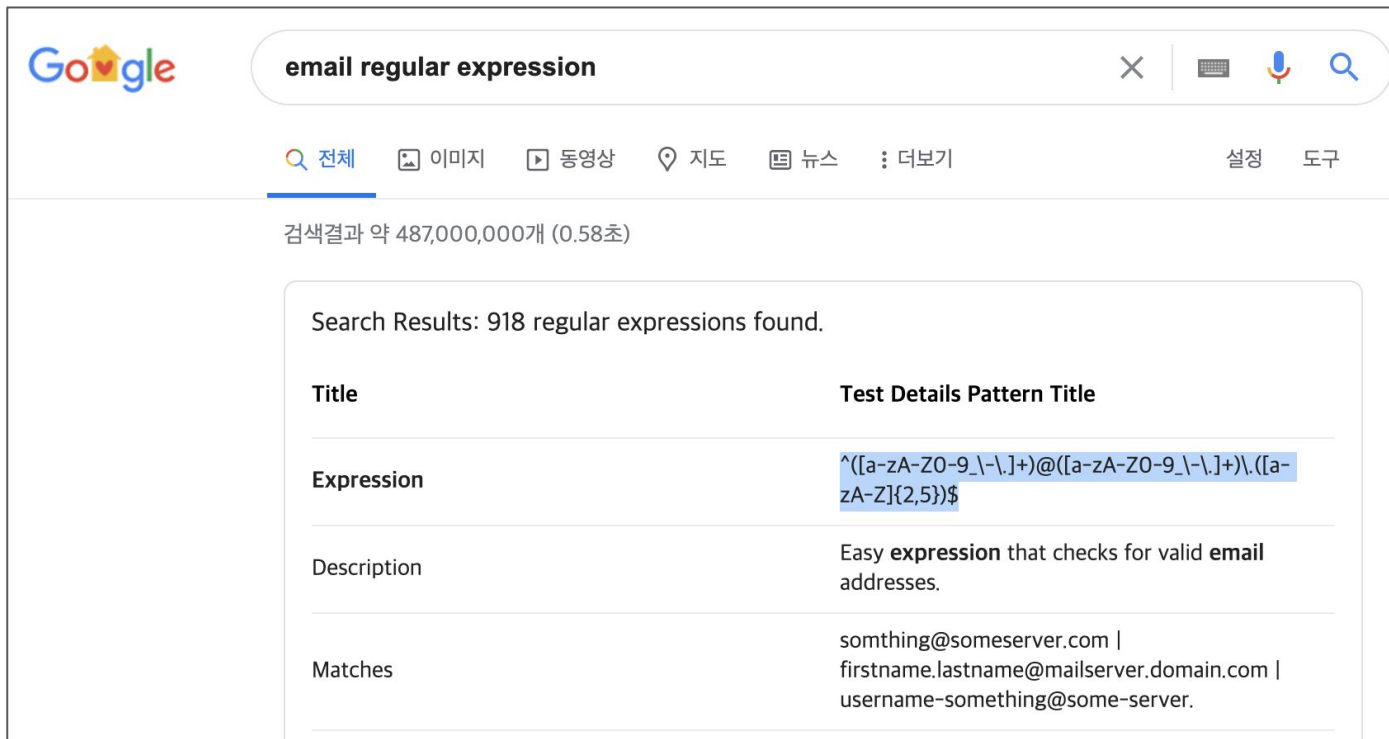
미션 1. 문서에서 email을 추출해보자

전략

1. email 정규표현식 패턴 생성
2. re를 이용하여 문자열 추출
3. email에서 email id와 도메인 분리 추출



검색을 통해 email 정규표현식 패턴을 찾아보자



The screenshot shows a Google search interface. The search bar contains the text "email regular expression". Below the search bar, there are tabs for "전체" (All), "이미지" (Images), "동영상" (Videos), "지도" (Maps), "뉴스" (News), and "더보기" (More). The "전체" tab is selected. Below the tabs, it says "검색결과 약 487,000,000개 (0.58초)". The search results are displayed in a table format. The table has two columns: "Title" and "Test Details Pattern Title". The first row shows the title "Search Results: 918 regular expressions found." and the pattern title "^[a-zA-Z0-9_\\-\\.]+@[a-zA-Z0-9_\\-\\.]+\\.([a-zA-Z]{2,5})\$". The second row shows the title "Expression" and the pattern title "^[a-zA-Z0-9_\\-\\.]+@[a-zA-Z0-9_\\-\\.]+\\.([a-zA-Z]{2,5})\$". The third row shows the title "Description" and the description "Easy **expression** that checks for valid **email** addresses." The fourth row shows the title "Matches" and the matches "something@someserver.com | firstname.lastname@mailserver.domain.com | username-something@some-server."

Google

email regular expression

전체 이미지 동영상 지도 뉴스 더보기

설정 도구

검색결과 약 487,000,000개 (0.58초)

Search Results: 918 regular expressions found.

Title	Test Details Pattern Title
Expression	<code>^[a-zA-Z0-9_\\-\\.]+@[a-zA-Z0-9_\\-\\.]+\\.([a-zA-Z]{2,5})\$</code>
Description	Easy expression that checks for valid email addresses.
Matches	something@someserver.com firstname.lastname@mailserver.domain.com username-something@some-server.

미션 1. 문서에서 email을 추출해보자

전략

1. email 정규표현식 패턴 생성
2. re를 이용하여 문자열 추출
 - email에서 email id와 도메인 분리 추출

```
text = """
당선을 축하드립니다.
당선된 분들 email 목록입니다.
```

```
blackdew7@gmail.com
egoing@opentutorials.org
hanaro@finecode.kr
sookbun@dplus.company
"""
```

```
import re

regex_email = re.compile('([a-zA-Z0-9_\-\.]+)@([a-zA-Z0-9_\-\.]+\.[a-zA-Z]{2,5})')
regex_email.findall(text)

[('blackdew7', 'gmail', 'com'),
 ('egoing', 'opentutorials', 'org'),
 ('hanaro', 'finecode', 'kr'),
 ('sookbun', 'dplus', 'compa')]
```

미션 2. email에서 domain을 'opentutorials.com'으로 통일

전략

1. email 정규표현식 패턴 생성
2. re를 이용하여 문자열 바꾸기
 - sub 메소드를 통해서 변환
 - '\g<num>' 사용법 이해

```
import re

regex_email = re.compile('([a-zA-Z0-9_\-\.]+)@([a-zA-Z0-9_\-\.]+\.[a-zA-Z]{2,7})')
print(regex_email.sub('\g<1>@opentutorials.com', text))
```

당선을 축하드립니다.

당선된 분들 email 목록입니다.

blackdew7@opentutorials.com
egoing@opentutorials.com
hanaro@opentutorials.com
sookbun@opentutorials.com

실습

HTML에서 a 태그의 href 링크 추출하기.