article_raw

- 파일을 read()로 읽어와 하나의 str로 받아옵니다.
- re.split("\n\n", file_data)로 논문별로 구분합니다.
- 구분된 각각의 item은 가공되지 않은 순수한 논문이라는 의미에서 'article_raw'라고 부릅니다.

```
def read_file(file_name):
    with open(file_name, "rt") as file:
        return file.read()
```

```
# file_data : str - 파일을 통째로 read() 로 읽어옴.
file_data = FileManager.read_file("final_report.txt")

# data_sep : list - "\n\n" 로 구분되어 있는 논문들을 각각 리스트에 담음.
data_sep = re.split("\n\n",file_data)
# print(len(data_sep)) # 논문의 개수 7653

Article_list = list()

# article_raw : str - 논문 1개에 대한 날것의 정보 ( 아무 과정을 거치지 않음 )
for article_raw in data_sep:
    Article_list.append(Article(article_raw))
```

article_labeled & article_dict

- article_raw 를 이용하기 쉽게 가공하는 과정입니다.
- article_labeling(article_raw):
  - 내용의 길이에 따라 줄구분이 되어있는 자료를 헤더에 따라 줄바꿈이 되어있는 자료로 바꿉니다.
  - 예) AB - line1\nline2 => AB - line1line2
- article_dicting(article_raw):
  - labeling이 된 데이터를 헤더를 key로 가지는 dict로 가공합니다.
  - 하나의 article은 여러개의 AU를 가질수 있는데 이때 AU에 해당하는 데이터를 다음과 같이 가공합니다.
  - 예) AU - data1, AU - data2 => {"AU":[data1, data2], ... }
- 활용
  - article_dict를 활용하면 원하는 정보를 쉽게 가져올 수 있습니다.

```python
# input article_raw
# output article_labeled : list - ["PMID- ...", "DP  - ...", ... ]
def article_labeling(article_raw):
    article_labeled = list()
    article_lines = article_raw.split("\n")

    # 처음 6글자가 공백이면 기존영역
    # 처음 6글자가 "....- " 이면 새로운 영역
    header = str()
    for line in article_lines:

        if line[:6] == "      ":
            # 기존 영역에 속하므로 헤더에 line 더하기.
            # 줄맞춤을 위해 있던 6칸짜리 공백 지우기.
            header += line[6:]
        else:
            # 새로운 영역이므로, 전영역이 있었다면 저장 후 새로운 헤더 지정
            if header:
                article_labeled.append(header)
            header = line

    return article_labeled
```

```python
def article_dicting(article_raw):
    article_labeled = article_labeling(article_raw)
    article_dict = dict()
    for item in article_labeled:
        # item[:4].strip() : "AB  " -> "AB"
        # item[:4] item[6:] 사이 잘리는 부분은 "- "
        key, value = item[:4].strip(), item[6:]
        # key가 이미 존재한다면, [기존 value] + [새로운 value] = [기존1, 기존2, 새로운]
        # key가 존재하지 않았다면, [] + [새로운 value] = [새로운]
        article_dict[key] = article_dict.get(key, list()) + [value]
    return article_dict
```

```python
def get_tag(self, tag):
    return self.data.get(tag)
```

```
태그의 값을 획득할 수 있는 기능
Article.get_tag("LR") => ['20200325']
Article.get_tag("DP") => ['Kannan S', 'Shaik Syed Ali P', 'Sheeza A', 'Hemalatha K']
```

Article

- Article 클래스입니다.
- 객체의 개수를 셀 수 있게 클래스 변수인 counter를 만들고, 생성자가 실행될때마다 counter += 1합니다.
- Article(article_raw) 새로운 객체를 만들면, article_raw를 입력받고 문제해결에 필요한 7가지 데이터를 가져옵니다.
- 데이터의 존재 유무, 개수에 따라 가져오는 방법이 조금씩 다릅니다.
- print(Article) == print(Article.__str__) == print(f"PMID : {self.PMID} (Title : {self.TI})")

```python
class Article:

    # Article의 개수
    counter = 0

    def __init__(self, article_raw):
        Article.counter += 1
        self.data = article_dicting(article_raw)

        # article_dict_processing()는 모든 라벨 처리
        # Article의 생성자는 문제에 필요한 7개의 값만 처리
        # PMID DP TI AB AU AD MH

        # 항상 존재하며 1개만 존재 (1)
        self.PMID = self.data["PMID"][0]
        self.DP = self.data["DP"][0]
        # 1개이거나 존재하지 않을 수 있는 값들. (0~1)
        self.TI = self.data.get("TI", [None])[0]
        self.AB = self.data.get("AB", [None])[0]
        # 복수로 존재할 수도 있는 값들. (0~ )
        self.AU = self.data.get("AU")
        self.AD = self.data.get("AD")
        self.MH = self.data.get("MH")

    def __str__(self):
        return f"PMID : {self.PMID} (Title : {self.TI})"
```

```python
#############
# 존재 테스트 #
#############

# PMID DP TI AB 를 복수로 가지는 article 은 없으며,
# AB가 없는 논문 2개, TI가 없는 논문 6개, 둘다 없는 논문은 없다.
# AD, AU, MH 가 없는 논문은 많다.
# cnt = 0
# for Article in Article_list:
#     if len(Article.PMID) != 1:
#         print(f"PMID {len(Article.PMID)}")
#
#     if len(Article.DP) != 1:
#         print(f"DP   {len(Article.DP)}")
#
#     if Article.TI:
#         if len(Article.TI) != 1:
#             print(f"TI   {len(Article.TI)}")
#     else:
#         print(f"TI   None")
#         if not Article.AB:
#             print("both")
#
#     if Article.AB:
#         if len(Article.AB) != 1:
#             print(f"AB   {len(Article.AB)}")
#     else:
#         print("AB   None")
#
#     if not Article.AU:
#         print("AU None")
#     if not Article.AD:
#         print("AD None")
#     if not Article.MH:
#         print("MH None")
#
#
#     if (Article.AB):
#         if (Article.TI):
#             cnt += 1
#
# print(cnt) # 7645 + 2 + 6 = 7653
```

실행

```
print(f"클래스 메서드를 정의하여 Article 객체의 수를 관리"
      f"\n객체의 수(list len()) : {len(Article_list)}\n"
      f"객체의 수(cls method) : {Article.counter}")
print()

print(f"print 함수로 Article 객체를 출력하는 형식\n"
      f"print(Article) => "
      f"{Article_list[0]}")
print()

print(f"태그의 값을 획득할 수 있는 기능\n"
      f"Article.get_tag(\"LR\") => "
      f"{Article_list[0].get_tag('LR')}")
print(f"Article.get_tag(\"DP\") => "
      f"{Article_list[0].get_tag('AU')}")
```

```
클래스 메서드를 정의하여 Article 객체의 수를 관리
객체의 수(list len()) : 7653
객체의 수(cls method) : 7653
```

```
print 함수로 Article 객체를 출력하는 형식
print(Article) => PMID : 32141569 (Title : COVID-19 (Novel Coronavirus 2019) - recent trends.)
```

```
태그의 값을 획득할 수 있는 기능
Article.get_tag("LR") => ['20200325']
Article.get_tag("DP") => ['Kannan S', 'Shaik Syed Ali P', 'Sheeza A', 'Hemalatha K']
```