나의 첫 파이썬 특강 시리즈

파이썬으로 만드는 우리들의 메신저 대화 분석기

강사 한지훈

나의 첫 파이썬: 파이썬으로 만드는 우리들의 메신저 대화 분석기

- 우리는 지금부터 파이썬으로 메신저 대화 분석기를 만드는 방법에 대하여 학습합니다.
- 코드를 작성하기 위한 기초 파이썬 문법을 학습하고, 이를 적용하는 방법에 대하여 학습합니다.
- 실제 프로젝트 코드를 통하여 기초 파이썬 문법의 요소들이 어떻게 적용될 수 있는지에 대하여 학습합니다.

1. 프로젝트를 위한 사전 작업

- 프로젝트 진행을 위하여 몇가지의 구성 작업을 진행하여야 합니다.
- 아래의 셀들을 설명에 따라 실행하여 주시기 바랍니다.

1-1. 차트 및 워드 클라우드에서 한글 사용을 위한 폰트 설치 작업

- 결과물에서 정상적으로 한글을 보여주기 위해서 한글 폰트를 설치하여야 합니다.
- 아래의 셀을 실행시킨 후, 화면 상단 메뉴의 `런타임` - `세션 다시 시작`을 클릭하여 주시기 바랍니다.

In [ ]:
```
# 이 셀의 코드를 실행하세요

!sudo apt-get install -y fonts-nanum
!sudo fc-cache -fv
!rm ~/.cache/matplotlib -rf
```

1-2. 강의에서 사용되는 라이브러리 설치를 위한 작업

- 우리는 아래의 추가 기능과 코드를 활용하기 위하여 라이브러리를 설치합니다.
  - 표 데이터를 다루기 위한 Panda라는 이름의 라이브러리를 사용합니다.
  - 그래프를 만들기 위한 Matplotlib라는 이름의 라이브러리를 사용합니다.
  - 한글의 형태소를 분석하기 위하여 KoNLPy라는 이름의 라이브러리를 사용합니다.
  - 단어 구름을 생성하기 위하여 WordCloud라는 이름의 라이브러리를 사용합니다.

In [ ]:
```
# 이 셀의 코드를 실행하세요

%pip install pandas
%pip install matplotlib
%pip install konlpy
%pip install wordcloud
```

기초 파이썬 문법: `라이브러리`

- 라이브러리란, 다른 이가 만든 코드 혹은 코드의 모음입니다.

- 우리는 컴퓨터에 특정 기능의 프로그램을 설치하고 실행하여 사용합니다.
- 파이썬에서도 라이브러리를 설치하고 불러와 다른 사람이 만든 기능을 나의 코드에서 사용할 수 있습니다.
- 라이브러리에는 별도의 설치가 필요한 라이브러리가 있으며, 별도의 설치가 필요없이 사용할 수 있는 라이브러리가 있습니다.

- 라이브러리의 설치를 위해서는 `pip` 명령을 사용합니다.
- 아래의 코드는 `pip` 프로그램을 통하여 `pandas` 라는 이름의 라이브러리를 설치 (`install`)하는 코드입니다.

In [ ]:
```
%pip install pandas
```

- 우리가 프로그램을 `설치` 하고 실행까지 해주어야 하듯이, 우리가 설치한 라이브러리를 사용할 때에는 해당 라이브러리를 불러오기(import) 해주어야 합니다.

- pandas 라이브러리 사용을 위하여 불러오기 해줍니다.
- 아래처럼 라이브러리를 불러오기 위하여 `import` 이후에 라이브러리의 이름을 기입해 줍니다.

In [ ]:
```
# 이 셀의 코드를 실행하세요

import pandas
```

- 라이브러리를 불러올 때 별칭을 지정할 수도 있습니다.
- 이름이 길거나 복잡한 라이브러리의 경우에는 별칭(`alias`)을 지정하여 간단한 이름으로 바꾸어 사용할수도 있습니다.
- 아래는 각각 그래프를 그리기 위한 `Matplotlib` 의 pyplot이라는 요소를 `plt` 라는 별칭으로 불러오고 있습니다.
- `pandas` 라이브러리도 `pd` 라는 별칭으로 불러오고 있습니다.
- `as` 로 별칭을 지정하고 불러오는 라이브러리는 실제 코드를 작성할 때 해당 별칭을 이용하면 됩니다.

In [ ]:
```
# 이 셀의 코드를 실행하세요

import matplotlib.pyplot as plt
import pandas as pd
```

- 라이브러리의 특정 요소를 불러오기 위해서는 `from` 구문과 함께 사용하여 특정 요소만을 효율적으로 불러올 수도 있습니다.
- 아래 코드는 각각 `wordcloud` 라이브러리에서 `WordCloud` 라는 요소를 불러오고 있으며, `konlpy` 라이브러리에서 `Komoran` 이라는 요소를 불러오고 있습니다.

In [ ]:
```
# 이 셀의 코드를 실행하세요

from wordcloud import WordCloud
from konlpy.tag import Komoran
```

1-3. 차트 및 워드 클라우드에서 한글 사용을 위한 폰트 설정 작업

- 위에서 설치한 한글 폰트를 그래프를 그리는 Matplotlib 라이브러리에서 사용하기 위한 작업입니다.
- 위에서 학습한 방식처럼 라이브러리를 불러와 설정하여 줍니다.

In [ ]:
```
# 이 셀의 코드를 실행하세요

import matplotlib.pyplot as plt

plt.rc('font', family='NanumGothic')
```

1-4. 구글 드라이브와 연결을 위한 작업

- 아래의 셀을 실행하고 나타나는 `노트북에서 Google Drive 파일에 액세스하도록 허용하시겠습니까?` 라는 제목의 대화상자에서 `Google Drive에 연결` 을 클릭합니다.
- 이후 나타나는 창에서 본인의 계정을 선택하여 줍니다. 계정을 선택하고 `Google Drive for desktop 서비스로 로그인` 에서 `계속` 을 클릭합니다.
- 계속을 클릭 후 이동되는 `Google Drive for desktop에서 Google 계정에 대한 추가 액세스를 요청합니다.` 의 창에서 `계속` 을 클릭합니다.

In [ ]:
```
# 이 셀의 코드를 실행하세요

from google.colab import drive

drive.mount("/content/drive")
```

2. 메신저 데이터 불러오고 살펴보기

2-1. 메신저 PC버전 프로그램에서 대화내역 내보내기

- 메신저에서 대화내용을 내보내시기 바랍니다.
- 내보낸 대화내용 `csv` 파일을 드라이브에 업로드하여 주시기 바랍니다.
- 혹은 제공되는 가상의 대화 `csv` 파일을 사용할 수 있습니다.

2-2. 파이썬에서 Pandas를 이용하여 표 데이터 사용하기

- 아래 코드는 `pandas` 라이브러리를 이용하여 `csv` 파일을 읽는 코드입니다.
- pandas 라이브러리를 사용하기 위해 `import pandas as pd` 로 불러왔으므로, `pd` 라는 이름으로 사용하고 있습니다.
- pandas의 `read_csv` 함수를 사용하고 있습니다.
- `read_csv` 함수의 인자로 csv 파일의 `경로` 가 주어집니다.

In [ ]:
```
# 이 셀에 코드를 작성하세요.

pd.read_csv("{대화 기록 파일 경로}")
```

기초 파이썬 문법: `함수`

- 함수는 특정 작업을 수행하는 코드의 묶음입니다.
- 함수는 반복적으로 실행되는 코드를 매번 치지 않고 함수의 이름만으로 실행할 수 있게 만들어 줍니다.
- 따라서 함수를 사용하면 의미구조가 명확해지고, 재사용성이 높아집니다. 효율적으로 코드를 관리할 수 있게 해줍니다.
- 함수는 인자와 함수가 수행할 코드, 반환값으로 구성됩니다.

In [ ]:
```
# 이 셀의 코드를 실행하세요

def sum(a, b):
    result = a + b
    return result
```

In [ ]:
```
# 이 셀의 코드를 실행하세요

print(sum(1, 2))
```

- 위의 함수를 보면 def라는 키워드가 작성되어 있습니다. `def` 라는 키워드를 통하여 함수를 만들 수 있습니다.
- `def` 키워드 다음에는 함수의 이름이 작성됩니다.
  - 위 함수의 이름은 `sum` 입니다.
- 함수의 이름 다음 괄호에는 건네받을 `인자의 이름` 을 작성합니다.
  - 위 함수에서는 `(a, b)` 에서 볼 수 있듯이, 각각 `a` 라는 이름과 `b` 라는 이름의 2개의 인자를 받고 있습니다.
- 해당 건네받은 인자는 함수 코드 안에서 해당 이름으로 사용할 수 있습니다.
  - 함수 안쪽의 코드를 보면 2번째 줄에서 `a + b` 로, 인자를 이용하여 값을 계산하고 있음을 확인할 수 있습니다.
- 함수에서 실행할 코드를 작성할 때에는 **`들여쓰기를 반드시 해주어야 합니다.`**
  - 들여쓰기는 `tab 한 번` 또는 `띄어쓰기 4칸` 으로 지정할 수 있습니다.
- `return` 키워드 뒤에는 함수의 실행이 끝나고 다시 되돌려 줄 반환값을 지정합니다.
  - 위에서는 `result 값` 을 함수를 실행한 곳으로 반환하여 주고 있습니다.
- 함수를 사용할 때에는 `함수의 이름` 과 `인자에 전달하길 원하는 값` 이나 변수를 입력하면 됩니다.
  - 위에서는 `sum(1, 2)` 이라는 코드를 통하여 `sum` 이라는 이름의 함수를 사용하고 있습니다.
  - `(1, 2)` 를 통해 `a와 b에 각각 1과 2라는 값` 을 전달하고 있습니다.
  - sum(1, 2)은 함수의 실행이 끝나면 그 결과값 `3` 을 반환합니다. `print 함수는 인자로 전달된 값을 출력` 하므로, `3이 출력` 됩니다.

연습문제

- 아래의 셀에 2개의 인자를 받고, `첫번째 인자에서 두번째 인자의 값을 뺀 값을 반환하는 함수` 를 작성하시기 바랍니다.

- 만든 함수의 반환값을 출력하여 보기 바랍니다.

In [ ]:
```
# 이 셀에 코드를 작성하세요.

def sub(a, b):
    result = a - b
    return result

print(sub(1, 2))
```

기초 파이썬 문법: `변수`

In [ ]:
```
# 이 셀의 코드를 실행하세요

a = 10
```

In [ ]:
```
# 이 셀의 코드를 실행하세요

print(a)
```

- `변수`는 `데이터를 저장하는 공간`입니다.
- 위의 코드에서 `대입연산자 =`를 이용하여 변수에 값을 저장하고 있습니다.
- 대입연산자 =의 `좌변에는 변수의 이름`이, `우변에는 해당 이름의 변수에 저장할 값`을 입력합니다.
- 변수에 저장된 값을 사용할 때에는 변수의 이름을 통해서 사용할 수 있습니다.

In [ ]:
```
# 이 셀의 코드를 실행하세요

b = "a"
```

In [ ]:
```
# 이 셀의 코드를 실행하세요

print(b)
```

- 변수에는 10과 같은 `정수(int)`만 저장할 수 있는 것은 아닙니다.
- 파이썬에서 사용할 수 있는 모든 값을 저장할 수 있습니다.
- 위의 코드는 "a"라는 `문자열(str)` 형태의 `글자, 단어, 문장과 같은 값`을 저장하는 코드입니다.
- 문자열은 `큰따옴표` 혹은 `작은따옴표`로 감싸 만들어낼 수 있습니다.

해결하기

- 위에서 우리는 pandas를 통하여 csv 파일을 열고, 표 형태의 자료를 얻는 방법에 대해서 배웠습니다.
- `pandas`에서 사용되는 이러한 표 형태의 자료를 `DataFrame`이라고 부릅니다.

- 아래의 셀에 `read_csv` 함수를 통해 반환된 DataFrame 형태의 값을 `df` 라는 이름의 변수에 저장하는 코드를 작성하여 보시기 바랍니다.
- 해당 변수값을 `출력` 하시기 바랍니다.

In [ ]:
```
# 이 셀의 코드를 수정하세요.

df = pd.read_csv("{대화 기록 파일 경로}")
```

In [ ]:
```
# 이 셀에 코드를 작성하세요.

df
```

2-2. Pandas를 이용하여 필요한 표 데이터 선택하고 필요한 형태의 데이터로 만들기

- Pandas에서 사용하는 표 형태의 자료형인 `DataFrame` 에는 `여러 유용한 함수와 기능이 포함` 되어 있습니다.
- 각 참여자의 메시지수를 조회하는 방법에 대해서 알아봅시다.
  - `df` 의 `User` 열에는 `참여자의 이름` 이 존재합니다.
  - `User` 열에 나타나는 `참여자 이름의 수` 를 세어, 각 `참여자의 메시지 수` 를 파악하여 봅시다.

- 아래 코드는 `df` 에서 `User라는 열` 을 선택하는 코드입니다.
- 표 형태의 자료형 `DataFrame` 에서 `열을 선택할 때` 에는 `대괄호 안에 열의 이름` 을 적어 열을 추출할 수 있습니다.

In [ ]:
```
# 이 셀의 코드를 실행하세요

user_column = df["User"]
```

In [ ]:
```
# 이 셀의 코드를 실행하세요

user_column
```

연습문제

- 위에서 열을 선택한 것처럼 `Message` 라는 열을 선택하고, `message_column 변수` 에 저장하여 보세요.
- `message_column` 변수를 출력해보시기 바랍니다.

In [ ]:
```
# 이 셀에 코드를 작성하세요.

message_column = df["Message"]
message_column
```

- 아래 코드는 선택한 열에 대해서 `value_counts` 함수를 사용하는 코드입니다.
- `value_counts` 함수는 해당 열에 대해서 `반복적으로 나오는 값의 수` 를 세어 반환하여 주는 함수입니다.

In [ ]:
```
# 이 셀의 코드를 실행하세요

user_count = user_column.value_counts()
```

In [ ]:
```
# 이 셀의 코드를 실행하세요

user_count
```

2-3. 만들어낸 데이터를 통하여 파이 차트 만들기

- `Matplotlib` 을 통하여 얻어낸 값을 통하여 파이 차트를 만들어 보겠습니다.
- `pie` 차트를 만들기 위하여 `pie` 함수를 사용하겠습니다.
- 필요한 인자를 알기 위해서는 `도큐먼트` 를 확인하여야 합니다.
- `? 명령` 을 통하여 인자 및 함수의 `도큐먼트` 를 확인할 수 있습니다.

In [ ]:
```
# 이 셀의 코드를 실행하세요

plt.pie?
```

- 파이 차트 각 부분에 `해당 참여자 이름을 함께 표기` 하기 위해서 `labels` 인자에 `참여자 이름을 전달` 하여 주어야 합니다.
- `참여자 이름의 목록` 을 얻어내기 위해서 `index 속성` 을 사용합니다.
- 속성은 함수와 비슷하게 값을 반환합니다.
- 하지만 인자를 전달하여 값을 조정할 수는 없으며 단순히 정해진 값을 반환합니다.

In [ ]:
```
# 이 셀의 코드를 실행하세요

user_index = user_count.index
```

In [ ]:
```
# 이 셀의 코드를 실행하세요

user_index
```

- `pie` 함수에 위에서 얻어낸 `user_count 데이터와 user_index 데이터를 전달` 합니다.
- 차트에 퍼센트 값을 함께 표시하기 위하여 `autopct` 인자를 전달해줍니다.
- `title` 함수를 통해 차트의 이름을 표시합니다.
- `show` 함수를 통해 차트를 출력합니다.

In [ ]:
```
# 이 셀의 코드를 실행하세요

plt.pie(user_count, labels=user_index, autopct="%1.1f%%")
plt.title("채팅방 대화 점유율")
plt.show()
```

3. 메신저 대화 분석하고 시각화하기

- 우리는 위에서 간단한 파이썬 문법과 라이브러리의 도움으로 참여자들의 메시지 전송 비율을 파이 차트로 만들어 냈습니다.

- 아래에서는 메신저의 대화 내용을 분석하고, 각 참여자 당 많이 사용한 단어들을 `단어 구름`으로 만들어보겠습니다.

- 많은 데이터 분석에서는 문제를 해결하기 위하여 다음과 같은 과정을 진행합니다.

1. **분석을 위한 설계**
   - 각 참여자의 메시지의 목록을 확인한 후 메시지에 포함된 단어(명사)를 통하여, **해당 사용자가 자주 사용하는 단어를 확인**합니다.
   - 우리는 이러한 설계를 기반으로 어떻게 코드를 작성할지 계획을 세울 수 있습니다.
2. **분석을 위한 데이터 준비**
   - 위에서 추출한 **메신저 대화 기록**을 사용합니다.
   - 표 형태의 메신저 대화 기록을 위에서 사용한 `Pandas`의 `DataFrame`을 통하여 표현하여 낼 겁니다.
3. **분석을 위한 전처리와 데이터 가공**
   - 우리가 가지고 있는 데이터에서 **분석할 수 없거나, 필요하지 않는 데이터를 정제**합니다.
   - 이를 통하여 우리가 필요한 데이터가 더 잘 드러나게 됩니다.
   - 데이터에는 분석을 진행할 수 없는 `이모지`나, 우리 눈에는 직접적으로 보이지 않지만, `특수한 기능을 하는 문자`들이 포함되어 있습니다.
   - 위와 같은 문자를 `제거`한 후 우리가 필요한 `명사 단어`만 추출할 수 있습니다.
   - 우리는 이미 정제된 데이터를 가지고 형태소 분석을 진행하는 방법만을 진행하여 보겠습니다.
   - 명사 추출을 위하여 `한국어 형태소 분석기`인 `KoNLPy`를 활용하겠습니다.
4. **가공한 데이터를 통한 데이터 분석**
   - 위에서 가공한 데이터를 통해서 **시각화**를 진행합니다.
   - 우리는 `WordCloud` 라이브러리를 이용하여, `단어 구름`을 만들어 보겠습니다.

3-1. 데이터를 불러오고 필요한 데이터 선택하기

- 우리는 위에서 `df`에 대화기록 데이터를 `DataFrame`의 형태로 불러왔습니다.
- 대괄호 안에 열 이름을 작성하여, 특정 열을 선택하여 사용한 바 있습니다.
- 우리는 `각 참여자에 대한 대화기록`이 필요합니다.
- 아래에서 필요한 데이터를 찾아봅시다.

In [ ]:
```
# 이 셀의 코드를 실행하세요

df
```

- 위의 `df`에서 우리는 User가 동일한 메시지 행만을 선택하고 싶습니다.
- `Pandas`의 `Boolean Indexing`을 통하여 이러한 기능을 수행할 수 있습니다.
- `Boolean Indexing`은 `특정 열에 대하여 조건에 해당하는 행만을 추출`하고자 할 때 사용하게 됩니다.

1. 기준이 될 열(Column)을 선택합니다.
   - 우리는 `User 열에 대해서 주어진 이름과 일치하는지 여부`를 확인할 것입니다.

- `User열을 선택` 합니다.

In [ ]:
```
# 이 셀에 코드를 작성하세요.

df["User"]
```

2. 해당 열에 대해서 원하는 조건을 부여합니다.
   - `User의 값이 내가 원하는 참여자의 이름과 같은지 판단` 합니다.
   - `자료형` 에 주의하세요.

In [ ]:
```
# 이 셀에 코드를 작성하세요.

df["User"] == "김가람"
```

3. 조건으로 만들어진 결과물을 열을 선택하듯, 열의 이름자리에 넣습니다.
   - `열의 이름이 들어가는 자리` 에 `2에서 만든 코드를 삽입` 합니다.

In [ ]:
```
# 이 셀에 코드를 작성하세요.

df[df["User"] == "김가람"]
```

4. 우리에게 필요한 `Message 열만 선택` 합니다. 그리고 이를 `messages` 라는 이름의 변수에 저장해둡니다.

In [ ]:
```
## 이 셀에 코드를 작성하세요.

messages = df[df["User"] == "김가람"]["Message"]
messages
```

기초 파이썬 문법: 비교연산자와 논리자료형

- 위에서 우리는 `True` , `False` 라는 값을 확인한 바 있습니다.
- 파이썬에서 이러한 `True` 와 `False` 라는 값을 `논리자료형(Boolean)` 이라고 합니다.
- 각각의 값은 참과 거짓을 의미하며, 논리자료형에는 이 두 값만이 존재합니다.
- 이러한 `논리자료형` 과 조건을 기술할 수 있는 `비교연산자` 는 밀접한 관계를 가집니다.

- 아래의 조건은 거짓이므로 `Fasle` 값이 연산되어 나옵니다.

In [ ]:
```
print(1 > 2)
```

- 아래의 조건은 참이므로 `True` 값이 연산되어 나옵니다.

In [ ]:
```
print(1 < 2)
```

- 파이썬에서 `같음`을 의미하는 비교연산자는 `==` 입니다.

In [ ]:
```
print(1==1)
```

- 이러한 `같음을 비교하는 연산자`는 `문자열`에서도 사용할 수 있습니다.

연습문제

- 아래의 각각의 `결과값`이 무엇이 나올지 예측하여 보시기 바랍니다.
- 각각의 `자료형`이 무엇일지 예측하여 보시기 바랍니다.
- 파이썬에서 자료형을 확인할 때에는 `type` 함수를 사용합니다.

In [ ]:
```
print("Python" == "Python")
print(type("Python" == "Python"))
```

In [ ]:
```
print("Python" == "Not Python")
print(type("Python" == "Not Python"))
```

3-2. 필요한 데이터로 가공하기

- 우리는 가져온 데이터, 즉 `특정 참여자의 메시지에서 명사 단어`만을 추출해야 합니다.
- 이러한 작업을 `KoNLPy`라는 라이브러리의 도움을 받아 진행해보도록 합시다.
- `NLP(Natural Language Processing)`란 컴퓨터를 이용하고 텍스트 데이터를 처리하고 해석하는 기술을 의미합니다.
- `KoNLPy`는 파이썬에서 활용할 수 있는 `한글 NLP 라이브러리`로, 이를 통하여 각 단어의 품사나 형태소를 분석해낼 수 있습니다.

- `KoNLPy`에는 다양한 분석기가 존재합니다. 우리는 그 중 `Komoran`이라는 분석기를 활용할 것입니다.
- 아래의 코드는 Komoran 분석기를 생성하는 과정입니다.
- `Komoran` 분석기를 생성하여 `analyzer`란 `변수`에 저장하고 있습니다.
- 우리는 앞으로 이 만들어진 분석기를 `analyzer`라는 이름의 `변수`를 통하여 사용하면 됩니다.

In [ ]:
```
# 이 셀의 코드를 실행하세요

analyzer = Komoran()
```

- 문장의 형태소를 분리하기 위해서는 `morphs 함수`를 사용합니다.
- 함수의 인자로 분석을 원하는 문장을 `문자열` 형태로 전달합니다.
- 함수의 반환값은 `리스트(list)`로 반환됩니다.

In [ ]:
```
# 이 셀의 코드를 실행하세요

text = "이것은 한국어 형태소 분석의 예시입니다."
```

```
print(analyzer.morphs(text))
```

- 문장에 존재하는 명사만을 분리하기 위해서는 `nouns 함수` 를 사용합니다.
- 함수의 인자로 분석을 원하는 문장을 `문자열` 형태로 전달합니다.
- 함수의 반환값은 `리스트(list)` 로 반환됩니다.

In [ ]:
```
# 이 셀의 코드를 실행하세요

text = "이것은 한국어 형태소 분석의 예시입니다."

print(analyzer.nouns(text))
```

기초 파이썬 문법: `리스트 자료형`

- `리스트(list)` 는 우리가 이전에 배웠던 `정수형(int)` , `문자열(str)` , `논리형(boolean)` 과 같은 자료의 형태를 의미합니다.
- 리스트는 `여러 개의 데이터를 하나의 이름으로 통합적으로 관리` 하는데 그 목적을 가집니다.
- 리스트는 `여러 개의 다양한 자료형을 담아서 그 순서` 를 붙여 데이터를 관리합니다.

- 아래처럼 나열을 원하는 데이터를 `대괄호 안에 쉼표 구분` 하여 입력하여 주면 리스트를 만들어 낼 수 있습니다.

In [ ]:
```
# 이 셀의 코드를 실행하세요

even_numbers = [2, 4, 6, 8, 10]
print(even_numbers)
print(type(even_numbers))
```

- 리스트에는 하나의 자료형만 들어가지 아니하여도 좋습니다.
- 아래는 `다양한 형태의 자료형이 하나의 리스트에 포함` 되어 있는 형태의 예시입니다.

In [ ]:
```
# 이 셀의 코드를 실행하세요


mess_list = [1, True, "Hello", [1, 2, 3]]
print(mess_list)
print(type(mess_list))
```

- 이러한 리스트에서 각 자료를 조회하거나 수정하고 싶다면 그 순서를 통하여 접근할 수 있습니다.
- 이때의 순서를 `인덱스` 라고 이야기 합니다.
- 이러한 인덱스는 몇가지 역사적, 물리적 이유로 `0` 부터 시작합니다.
- 아래는 `mess_list` 에 포함되어 있는 각각의 데이터를 `인덱스를 통하여 출력` 하고, 그 자료형을 조회하는 코드입니다.

In [ ]:
```
# 이 셀의 코드를 실행하세요

print(mess_list[0])
print(type(mess_list[0]))
```

In [ ]:
```
# 이 셀의 코드를 실행하세요

print(mess_list[2])
print(type(mess_list[2]))
```

- 리스트에 값을 추가하거나 리스트끼리 합칠 수 있습니다.
- 리스트에 `값을 추가` 할때에는 `append` 함수를 사용합니다.
  - 가장 마지막에 값이 추가됩니다.

In [ ]:
```
even_numbers.append(12)
```

In [ ]:
```
print(even_numbers)
```

- 리스트끼리 `합칠 때` 에는 `extend` 함수를 사용합니다.
  - 아래 코드는 `mess_list` 리스트 끝에 `even_numbers` 리스트의 요소들을 끝에 합치는 코드입니다.
  - 리스트가 하나의 요소로 추가되는 것이 아닌, `실제로 각 요소들이 합쳐짐` 에 주의하세요.

In [ ]:
```
mess_list.extend(even_numbers)
```

In [ ]:
```
print(mess_list)
```

연습문제

- 아래의 셀에 `odd_numbers` 라는 이름의 변수에 `1이상 10이하의 수 중 홀수 정수` 를 담은 리스트를 만드시기 바랍니다.

In [ ]:
```
odd_numbers = [1, 3, 5, 7, 9]
```

연습문제

- `odd_numbers 리스트 마지막` 에 `11이라는 정수 값을 추가` 하시기 바랍니다.

In [ ]:
```
odd_numbers.append(11)
```

연습문제

- `mess_list 리스트 마지막` 에 `odd_numbers 리스트 자체` 를 추가하시기 바랍니다.

In [ ]:
```
mess_list.append(odd_numbers)
```

연습문제

- `mess_list 리스트 마지막` 에 `odd_numbers 리스트 요소를 모두 추가` 하시기 바랍니다.

In [ ]:
```
mess_list.extend(odd_numbers)
```

3-3. 필요한 모든 데이터로 가공하기

- 우리는 위에서 `KoNLPy` 라이브러리의 `Komoran 분석기`를 통하여 `명사 단어`만을 `리스트`로 얻어내는 방법을 알아보았습니다.
- 하지만 위 코드는 `하나의 메시지에 대해서만 수행` 됩니다.
- `모든 메시지에 반복적으로 분석기의 nouns 함수`를 적용하여 봅시다.
- 최종적으로 `모든 명사 단어가 하나의 리스트`로 만들어지도록 유도하여 봅시다.
- 파이썬으로 반복적으로 동작하는 코드를 작성하기 위해서는 `For 반복문`을 사용합니다.
  - For 반복문을 위해서 `3개의 요소`가 필요합니다.
  - For 반복문을 위한 `여러 데이터가 나열된 요소`가 필요합니다.
  - For 반복문은 `리스트와 같이 여러 개의 데이터가 나열된 요소`에 대해서, `각 요소를 자동적으로 변수`로 가져옵니다.
  - 자동적으로 `가져온 변수의 이름`이 필요합니다.
  - 자동적으로 `가져온 변수에 어떠한 반복적인 작업을 할 것인가에 대한 코드`가 필요합니다.

1. 여러 개의 데이터가 나열된 요소가 필요합니다.
   - 이 요소는 전혀 의미없는 요소가 아닌, `우리가 필요한 데이터`가 되어야 합니다.
   - 우리는 `모든 메시지에 대해서, 각각의 메시지에 분석기의 nouns 함수`를 사용해야 합니다.
   - 이 때 `필요한 여러 개의 데이터가 나열된 형태의 요소`는 우리가 위에서 선택한 `messages` 변수입니다.
2. 우리는 `전체 메시지에 대해서 자동적으로 가져오는 메시지 하나 하나`를 `message`라는 변수 이름으로 가져오겠습니다.
3. 이제 각각의 메시지는 `message`에 저장될 겁니다. 우리는 이 `message`에 `nouns 함수`를 적용하면 되겠습니다.

In [ ]:
```
# 이 셀의 코드를 실행하세요

for message in messages:
    print(analyzer.nouns(message))
```

해결하기

- 우리는 위 파이썬 기초 문법 중 리스트에서 `리스트끼리 합치는 extend 함수`에 대해서 배웠습니다.
- 우리는 최종적으로 `모든 명사 단어가 하나의 리스트`로 만들어져야 합니다.
- 이 리스트를 `nouns 라는 변수 이름`으로 아래 셀에 만들어 두었습니다.
- 여러분들은 위에서 만든 `For 반복문의 코드를 수정`하여 `각 리스트가 출력되는 것이 아닌 , nouns 리스트에 합쳐지도록` 코드를 수정하세요.

In [ ]:
```
# 이 셀의 코드를 수정하세요

nouns = []

for message in messages:
    nouns.extend(analyzer.nouns(message))
```

3-4. 가공이 끝난 데이터를 차트를 만들 수 있는 자료형으로 바꾸기

- 위에서 우리는 `모든 메시지에 대해서 nouns 함수` 를 사용하고, `모든 메시지에서 나타나는 명사를 nouns 라는 이름의 리스트 변수에 저장` 했습니다.
- 이제 우리는 각각의 명사 단어가 몇 번 등장했는지 개수를 세어야 합니다.
- 이전 우리는 `User` 열의 이름을 셀 때 `DataFrame` 의 `value_counts` 함수를 통하여 이러한 문제를 해결했습니다.
- 동일한 방법을 사용하기 위하여 우리가 가지고 있는 리스트 형태의 데이터를 `DataFrame` 으로 변환하여 봅시다.

- `pd.DataFrame` 을 통하여 우리가 가지고 있는 `파이썬 자료형을 데이터프레임` 으로 만들 수 있습니다.
- 이때 열의 이름을 지정하기 위해서 `columns 인자` 에 `열의 이름을 리스트의 형태` 로 넣어 줍니다.

In [ ]:
```
# 이 셀의 코드를 실행하세요

noun_df = pd.DataFrame(nouns, columns=["noun"])
```

In [ ]:
```
# 이 셀의 코드를 실행하세요

noun_df
```

해결하기

- `2-3. 만들어낸 데이터를 통하여 파이 차트 만들기` 에서 진행하였던 것처럼 `value_counts 함수` 를 위에서 만든 `noun_df` 의 `noun` 열에 대해서 실행하여 보세요.
- 그 반환값을 `noun_count 변수에 저장` 하여 보세요.
- 그리고 `noun_count 변수` 를 출력하여 보세요.

In [ ]:
```
# 이 셀에 코드를 작성하세요.

noun_count = noun_df['noun'].value_counts()
noun_count
```

3-5. 차트 만들기

- 우리는 위에서 각 명사 단어 수를 얻어 냈습니다.
- 이 자료를 통하여 단어 구름을 만들어 봅시다.
- `WordCloud` 를 통하여 우리는 `단어 구름` 을 만들어 낼 수 있습니다.
- `WordCloud 요소` 는 단어 구름을 만들거나 그릴 수 있는 요소입니다.

- 우리는 이 WordCloud 요소에 우리가 `원하는 형태로 단어 구름을 생성하도록 인자를 넣어 설정` 을 해주어야 합니다.
- 그리고 이 요소를 `wordcloud_generator 변수` 에 담아서 실제 단어 구름을 만드는 함수를 실행시킵니다.

In [ ]:
```
wordcloud_generator = WordCloud(width=800,
                                height=800,
                                background_color='white',
                                font_path='{폰트 파일 경로}')
```

- 아래의 `generate_from_frequencies` 함수를 통하여 단어 구름을 생성시킬 수 있습니다.
- 인자로 우리가 만든 `noun_count` 를 전달하여 줍니다.

In [ ]:
```
wordcloud = wordcloud_generator.generate_from_frequencies(noun_count)
```

- `파이 차트` 를 출력했던 것과 같이 `단어 구름 그림을 출력` 하기 위하여 `plt` 를 활용합니다.

In [ ]:
```
plt.figure()
plt.imshow(wordcloud)
plt.axis('off')
plt.title("김가람")
plt.show()
```