

07 문서 업무 자동화



경고 : 본 강의자료는 연세대학교 학생들을 위해 수업 목적으로 제작.게시된 것이므로, 수업목적 이외의 용도로 사용할 수 없으며, 다른 사람과 공유할 수 없습니다. 위반에 따른 법적 책임은 행위자 본인에게 있습니다.



연세대학교
YONSEI MIRAE
CAMPUS

- ◆ **openpyxl**과 **xlsxwriter** 를 이용해 엑셀 파일을 생성하고 수정하는 방법
- ◆ **pyautogui** 패키지로 사람이 마우스와 키보드로 컴퓨터를 다루듯이 자동화하는 방법

7.1 엑셀 업무 자동화

7.2 **openpyxl(R/W)** 과 **XlsxWriter(W)**로 엑셀 파일 만들기

7.3 **pyautogui**로 시스템 제어하기

엑셀 업무 자동화를 위한 openpyxl

- ◆ 엑셀, 파워포인트, 워드 파일과 같이 office 프로그램에서 만들어지는 문서 파일은 모두 표준 규약인 **Office Open XML(OOSML)** 규약에 맞춰 저장되므로
- ◆ MS Office 프로그램 없이도 문서의 내용을 변경하고 저장할 수 있다.
- ◆ **openpyxl** 은 OOXML의 이러한 점을 활용해 만들어진 라이브러리다.
- ◆ **pandas, numpy** 등 파이썬의 데이터 처리 모듈과 연계하여 효율적인 작업을 할 수 있다.

◆ openpyxl의 주요 기능

- 엑셀 파일(워크북) **생성, 수정, 삭제 및 기존 엑셀 파일 로드**
- 워크시트 생성 및 수정, 행/열(셀) 생성 및 수정
- 셀에 들어있는 값과 표시 형식 추가, 수정 및 셀 메모 생성
- 텍스트 서식, 셀 테두리/배경, 셀 크기 등 서식 수정
- 그래프(차트), 피벗 테이블 생성 및 수정

엑셀 업무 자동화를 위한 openpyxl



➤ openpyxl 설치 : 주피터 노트북에서 아래 pip 명령어 입력으로 설치

!pip install openpyxl

```
In [1]: 1 !pip install openpyxl
```

```
Requirement already satisfied: openpyxl in c:\programdata\anaconda3\lib\site-packages (3.0.9)
```

```
Requirement already satisfied: et-xmlfile in c:\programdata\anaconda3\lib\site-packages (from openpyxl) (1.1.0)
```

● 엑셀 파일 생성 및 저장

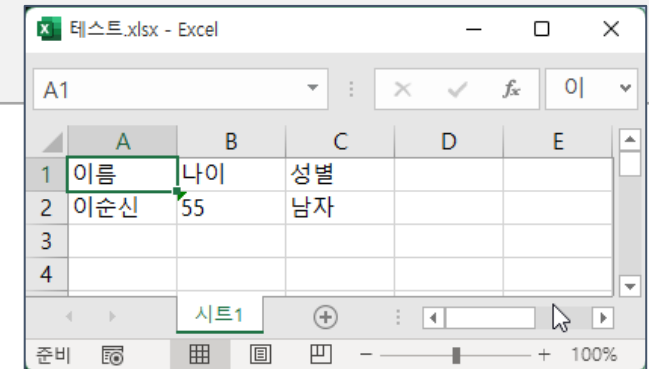
- openpyxl에 있는 **Workbook** 클래스의 다양한 함수들을 활용해 엑셀 파일 핸들링
- 엑셀 파일의 생성과 저장은 Workbook을 통해 이루어진다.
- 단, sheet를 사용할 때 주의할 점은 해당 엑셀 파일의 객체(Workbook) 변수를 통해 접근해야 한다.

```
from openpyxl import Workbook # Workbook 모듈(클래스) 불러오기

wb = Workbook() # wb 객체 생성. 동시에 워크시트 하나가 기본적으로 추가
ws = wb.active # wb 객체를 통해 워크시트를 활성화하고 ws 객체 생성. 현재 Active Sheet 얻기
ws.title="시트1" # ws 시트의 시트명 변경
ws.append(["이름","나이","성별"]) # ws 시트의 첫 번째 행(A1)부터 차례로 내용 추가
ws.append(["이순신","55","남자"]) # 두 번째 행에 추가
#wb["시트1"].append(["이순신", "55", "남자"]) # wb 객체의 시트를 직접 지정해서 두 번째 행에 추가하는 방법

wb.save(filename="chapter07/테스트.xlsx") # workbook(엑셀 파일) 저장
```

- 클래스를 복제해서 생성한 것을 **객체(인스턴스)**라고 하며, 객체에서는 클래스에서 정의한 **함수(메소드)**, **멤버변수(속성)** 등을 사용할 수 있다.

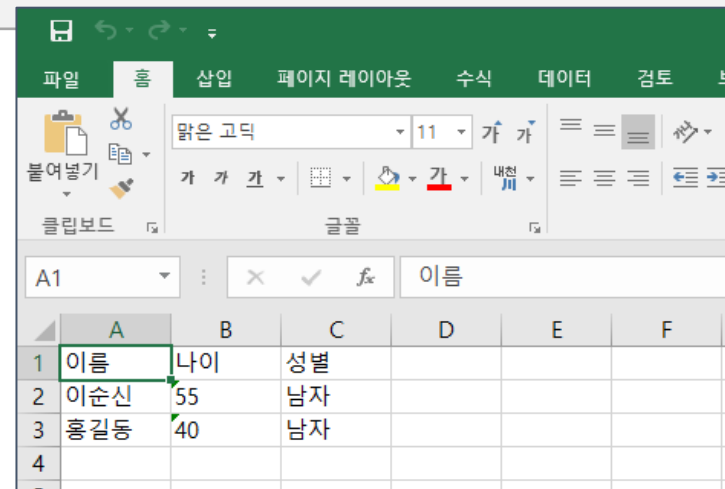


● 엑셀 파일 생성 및 저장

- 엑셀 파일을 불러와 기존 워크시트에 내용을 수정한 후 다시 저장
- **load_workbook()** 함수는 **Workbook** 클래스와 **PC에 저장된 엑셀 파일을 불러온 후 이를 객체로 저장**

```
from openpyxl import load_workbook      # Workbook 클래스와 엑셀 파일을 불러와 객체로 저장할 수 있는 함수
wb = load_workbook(filename = "chapter07/테스트.xlsx")  # "테스트.xlsx" 불러 오기
ws = wb.active                          # ws 활성화
ws.append(["홍길동", "40", "남자"])      # 기존 워크시트의 자료 행 다음에 새로운 행 추가

wb.save(filename = "chapter07/테스트.xlsx")          # wb를 "테스트.xlsx" 파일로 저장
```



The screenshot shows an Excel window with a worksheet named '테스트.xlsx'. The active cell is A1, which contains the text '이름'. The formula bar shows '이름'. The worksheet contains the following data:

	A	B	C	D	E	F
1	이름	나이	성별			
2	이순신	55	남자			
3	홍길동	40	남자			
4						

● 엑셀 시트 생성 및 저장

➤ 시트 생성하기 : **create_sheet()** 함수

```
from openpyxl import Workbook    # Workbook 모듈(클래스) 불러오기
wb = Workbook()                  # wb 객체 생성, wb에 시트 하나가 기본 추가됨
wb.create_sheet("시트2")          # wb에 두 번째 시트 추가
wb.create_sheet("시트3")          # wb에 세 번째 시트 추가
```

➤ 시트 정보 출력하기 : **sheetnames** 멤버변수

```
print(wb.sheetnames)             # wb 객체에 있는 모든 시트 정보 출력 → ["Sheet", "시트2", "시트3"]
```

➤ 시트 선택하기 : 시트에 대한 정보는 wb 객체에 **딕셔너리** 형태로 저장되어 있다.

```
ws = wb["시트2"]                 # wb 객체에 딕셔너리 형태로 저장되어 있는 "시트2"를 선택
```

➤ 선택한 시트에 내용 입력하고 저장하기

```
ws.append(["첫 번째로 추가된 시트입니다."])    # 선택된 시트에 내용 추가 "시트2"
wb["시트3"].append(["세 번째 시트에 내용을 추가합니다."]) # 시트를 직접 지정해서 내용 추가 "시트3"
wb.save(filename = "chapter07/샘플_시트 추가.xlsx") # 엑셀 파일로 저장
```

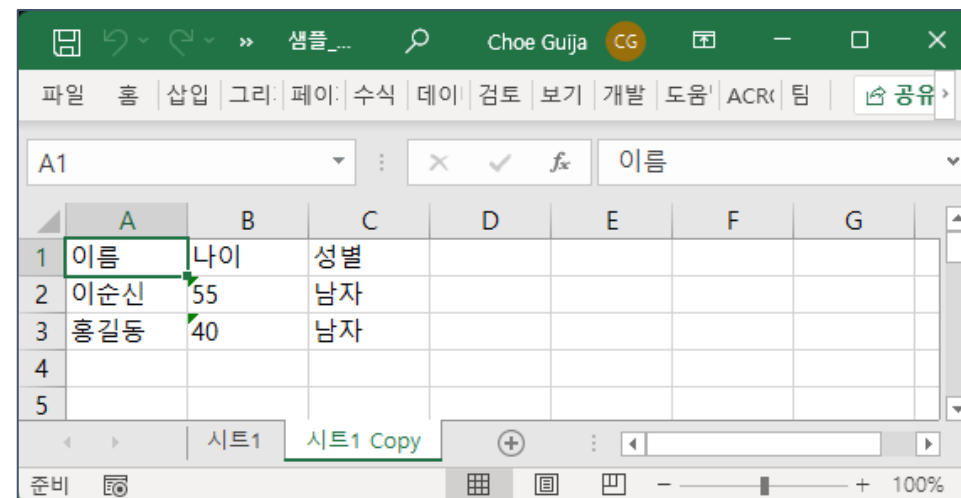
● 엑셀 시트 복사

- 시트 복사하기 : **copy_worksheet()** 함수
- 복사된 시트는 "**시트이름 Copy**" 로 생성된다

```
from openpyxl import load_workbook
wb = load_workbook("chapter07/테스트.xlsx")
wb.copy_worksheet(wb["시트1"])
wb.active = wb["시트1 Copy"]
wb.save(filename = "chapter07/샘플_시트 복사.xlsx")
print(wb.sheetnames)
```

Workbook 모듈과 load_workbook 불러오기
"테스트.xlsx" 파일을 불러와 wb 객체에 저장
"시트1"을 복사하여 wb 객체에 저장
파일을 열 때 "시트1 Copy"를 활성화
파일 저장
시트명 출력

- **copy_worksheet()** 함수를 통해 시트를 복사하면, 복사된 시트 이름에는 자동으로 '**Copy**'라는 접미어가 붙는다.
- **title** 멤버변수를 이용해 시트 이름을 변경할 수 있다.



● 엑셀 시트 수정/변경

➤ 시트 이름 바꾸기 : **.title**

```
wb = load_workbook("chapter07/샘플_시트 복사.xlsx") # "샘플_시트 복사.xlsx" 파일 불러오기
print(wb.sheetnames)                               # 시트명 출력 → ["시트1", "시트1 Copy"]

wb["시트1 Copy"].title = "시트1 복사"               # "시트1 Copy" 시트명을 "시트1 복사" 로 변경
wb.save(filename = "chapter07/샘플_시트 복사.xlsx") # 파일 저장
print(wb.sheetnames)                               # 시트명 출력 → ["시트1", "시트1 복사"]
```

➤ 시트 순서 변경하기 : **move_sheet()**

```
wb = load_workbook("chapter07/샘플_시트 복사.xlsx") # "샘플_시트 복사.xlsx" 파일 불러오기
print(wb.sheetnames)                               # 시트명 출력 → ["시트1", "시트1 복사"]

wb.move_sheet(wb["시트1 복사"], -1)                 # "시트1 복사" 시트를 왼쪽(-1)/오른쪽(1)으로 이동
wb.save(filename = "chapter07/샘플_시트 순서 변경.xlsx") # 저장
print(wb.sheetnames)                               # 시트명 출력 → ["시트1 복사", "시트1"]
```

● 엑셀 시트 삭제

➤ 시트 삭제하기 : **remove()**

```
from openpyxl import load_workbook
wb = load_workbook("chapter07/샘플_시트 순서 변경.xlsx")
print(wb.sheetnames)
```

```
wb.remove(wb["시트1 복사"])
wb.save(filename = "chapter07/샘플_시트 삭제.xlsx")
print(wb.sheetnames)
```

```
# Workbook모듈과 load_workbook 불러오기
# 샘플_시트 순서 변경 파일 불러오기
# 시트명 출력 → ["시트1 복사", "시트1"]
```

```
# 시트1 복사 시트 삭제하기
# 파일 저장
# 출력결과 → ["시트1"]
```

● 셀 내용 수정 및 삽입

- openpyxl에서 불러온 엑셀 파일의 모든 데이터는 Workbook 객체에 딕셔너리나 튜플 형태로 저장된다.
- 셀 내용 확인 : value 멤버변수
 - 인덱스 넘버는 **행은 1**부터, **열은 0**부터 시작
 - `ws["A1"].value` 와 `ws[1][0].value` 는 동일한 코드

```
from openpyxl import load_workbook
wb = load_workbook("chapter07/테스트.xlsx")
ws = wb["시트1"]
```

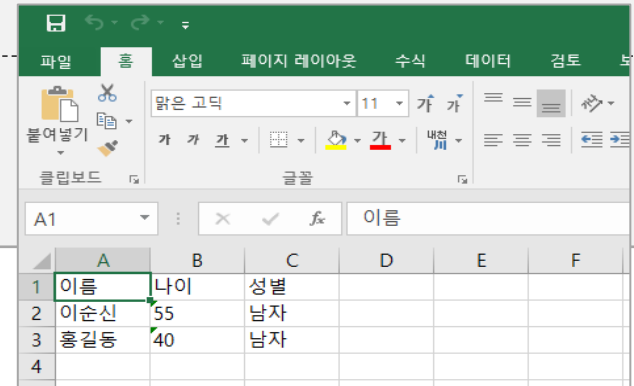
```
# Workbook모듈과 load_workbook 불러오기
# 테스트.xlsx 파일을 불러와 wb 객체에 저장
# 시트 선택
```

```
print(ws["A1"].value)
print(ws[1][0].value)
```

```
# [A1] 셀의 값을 출력 → 이름
# 1행의 0번째 컬럼의 값 출력 (A1 셀) → 이름
```

출력결과

이름
이름



	A	B	C	D	E	F
1	이름	나이	성별			
2	이순신	55	남자			
3	홍길동	40	남자			
4						

● 셀 내용 수정 및 삽입

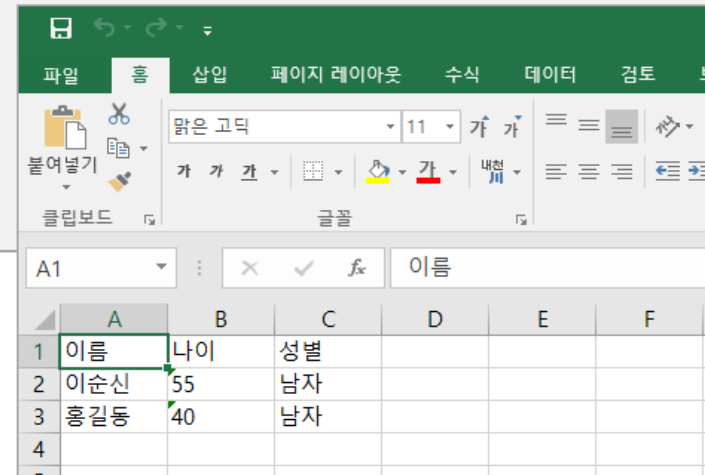
➤ 반복문을 통해 셀 정보에 접근하기 : 행 정보에 접근

```
from openpyxl import load_workbook          # Workbook모듈과 load_workbook 불러오기
wb = load_workbook("chapter07/테스트.xlsx")  # 테스트.xlsx 파일을 불러와 wb 객체에 저장
ws = wb["시트1"]                             # 시트 선택
```

```
for row in ws:                                # 반복문을 활용하여 ws 객체에 데이터가 있는 행을 차례 차례 불러옴
    print(row)                               # 행의 속성을 출력
```

출력결과

(<Cell "시트1".A1>, <Cell "시트1".B1>, <Cell "시트1".C1>)
(<Cell "시트1".A2>, <Cell "시트1".B2>, <Cell "시트1".C2>)
(<Cell "시트1".A3>, <Cell "시트1".B3>, <Cell "시트1".C3>)



	A	B	C	D	E	F
1	이름	나이	성별			
2	이순신	55	남자			
3	홍길동	40	남자			
4						

● 셀 내용 수정 및 삽입

- 반복문을 통해 셀 정보에 접근하기 : 반복문을 중첩하여 셀 정보 접근

```
for row in ws:                # 반복문을 활용하여 ws 객체에 데이터가 있는 행을 차례 차례 읽어 옴
    for cell in row:          # 반복문을 활용하여 변수 row에 있는 속성과 값을 읽어 cell에 저장
        print("셀 객체명 : {}, 셀 값 : {}".format(cell, cell.value))    # cell의 속성과 값을 출력
```

----- 출력결과 -----

```
셀 객체명 : <Cell "시트1".A1>, 셀 값 : 이름
셀 객체명 : <Cell "시트1".B1>, 셀 값 : 나이
셀 객체명 : <Cell "시트1".C1>, 셀 값 : 성별
셀 객체명 : <Cell "시트1".A2>, 셀 값 : 이순신
셀 객체명 : <Cell "시트1".B2>, 셀 값 : 55
셀 객체명 : <Cell "시트1".C2>, 셀 값 : 남자
셀 객체명 : <Cell "시트1".A3>, 셀 값 : 홍길동
셀 객체명 : <Cell "시트1".B3>, 셀 값 : 40
셀 객체명 : <Cell "시트1".C3>, 셀 값 : 남자
```

셀 내용 다루기



● 셀 내용 수정 및 삽입

➤ 셀 내용 수정하기

```
from openpyxl import load_workbook
wb = load_workbook("chapter07/테스트.xlsx")
ws = wb["시트1"]
```

```
ws["A1"] = "성명"
wb.save(filename = "chapter07/테스트_셀 내용 변경.xlsx")
```

Workbook모듈과 load_workbook 불러오기
"테스트.xlsx" 파일을 불러와 wb 객체에 저장
시트 선택

A1 셀의 내용을 "이름"에서 "성명"으로 변경
파일 저장

	A	B	C	D	E	F
1	이름	나이	성별			
2	이순신	55	남자			
3	홍길동	40	남자			
4						



	A	B	C	D	E	F
1	성명	나이	성별			
2	이순신	55	남자			
3	홍길동	40	남자			
4						

셀 내용 다루기



● 셀 내용 수정 및 삽입

➤ 반복문을 통해 셀 정보에 순차적으로 접근, 값 변경하기

```
wb = load_workbook("chapter07/테스트.xlsx")  
ws = wb["시트1"]  
new_data = ["신사임당", "45", "여자"]
```

```
row_no = 1  
for row in ws:  
    if row_no == 2:  
        for cell in row:  
            cell.value = new_data[cell.col_idx - 1]  
        row_no += 1
```

```
wb.save(filename = "chapter07/테스트_셀 내용 변경(반복문).xlsx") # 파일 저장
```

```
# 테스트.xlsx 파일을 불러와 wb 객체에 저장  
# 시트 선택  
# 변경할 데이터를 리스트로 생성
```

```
# 행 번호를 저장할 변수 설정  
# 행을 하나씩 읽어 row에 저장  
# row_no가 2일때(2행) 아래 코드 실행  
# 행의 셀을 하나씩 읽어 옴  
# 셀 값 변경  
# row_no 증가
```

	A	B	C	D	E	F
1	성명	나이	성별			
2	이순신	55	남자			
3	홍길동	40	남자			
4						

- `cell.value = new_data[cell.col_idx - 1]`
- `new_data`의 값을 하나씩 읽어와 `cell.value`의 값을 변경.
- 현재 [A2] 셀에 접근하고 있다면 `row`로 부터 받은 `cell`의 `col_idx`는 1 부터 시작하기 때문에 -1 을 해주어 리스트의 첫번째, 즉 인덱스 0 의 데이터 부터 `cell.value`에 입력

	A	B	C	D	E	F
1	이름	나이	성별			
2	신사임당	45	여자			
3	홍길동	40	남자			
4						

● 셀 내용 수정 및 삽입

➤ 셀에 내용 입력하기

```
from openpyxl import load_workbook
wb = load_workbook("chapter07/테스트.xlsx")
ws = wb["시트1"]
ws["A4"] = "신사임당"
ws["B4"] = "45"
ws["C4"] = "여자"
```

```
ws.append(["을지문덕", "60", "남자"])
```

```
new_data_list = [["유관순", "16", "여자"], ["세종대왕", "60", "남자"]]
```

```
for new_data in new_data_list:
    ws.append(new_data)
```

```
wb.save(filename = "chapter07/테스트_셀 내용 입력.xlsx")
```

```
# Workbook모듈과 load_workbook 불러오기
# "테스트.xlsx" 파일을 불러와 wb 객체에 저장
# 시트 선택
# 1) 셀에 데이터 직접 입력
```

2) 다음 행에 데이터 입력

리스트 생성

3) 반복문을 활용해 데이터 입력

파일 저장

	A	B	C	D	E	F
1	이름	나이	성별			
2	이순신	55	남자			
3	홍길동	40	남자			
4	신사임당	45	여자			
5	을지문덕	60	남자			
6	유관순	16	여자			
7	세종대왕	60	남자			

● 셀 내용 삭제

➤ 행/열 삭제하기 : **delete_rows(idx, amount)** / **delete_cols(idx, amount)**

- idx는 삭제할 행 또는 열 **번호**, amount는 idx 부터 삭제할 행 또는 열의 **개수**. **인덱스는 모두 1부터 시작**

```
from openpyxl import load_workbook
wb = load_workbook("chapter07/테스트.xlsx")
ws = wb["시트1"]
```

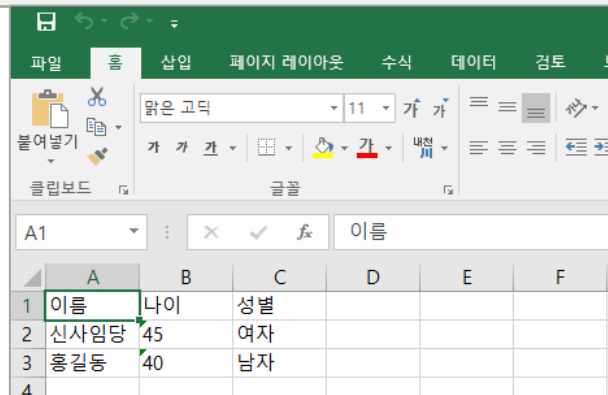
```
# Workbook모듈과 load_workbook 불러오기
# 테스트.xlsx 파일을 불러와 wb 객체에 저장
# 시트 선택
```

```
ws.delete_rows(idx=2, amount=1)
ws.delete_cols(idx=2, amount=2)
```

```
# 2행부터 1행만 삭제
# 두 번째 열(B열)부터 2열 삭제
```

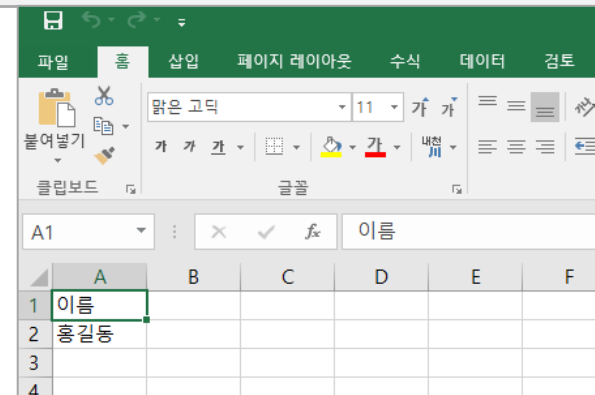
```
wb.save(filename = "chapter07/테스트_행과 열 삭제.xlsx")
```

```
# 파일 저장
```



Excel spreadsheet showing the initial state. Row 2 is selected. The data is as follows:

	A	B	C	D	E	F
1	이름	나이	성별			
2	신사임당	45	여자			
3	홍길동	40	남자			
4						



Excel spreadsheet showing the state after deleting row 2 and column B. The data is as follows:

	A	C	D	E	F
1	이름	성별			
2	홍길동	남자			
3					
4					

● 셀 복사하기 : 셀 하나의 값을 읽어 여러 셀에 복사해 넣기

```
from openpyxl import load_workbook
wb = load_workbook("chapter07/테스트.xlsx")
ws = wb["시트1"]
```

```
# Workbook모듈과 load_workbook 불러오기
# 테스트.xlsx 파일을 불러와 wb 객체에 저장
# 시트 선택
```

```
src = ws["A2"].value
ws["A5"] = src
```

```
# A2 셀 값을 변수 src에 저장
# A5 셀에 A2 셀 값을 복사
```

```
for row in ws["A6:C6"]:
    for cell in row:
        cell.value = src
```

```
# [A6], [B6], [C6] 셀을 선택해 반복문 실행
# 행의 각 셀에 접근
# 행의 각 셀을 src 변수 값으로 변경
```

```
wb.save(filename = "chapter07/테스트_셀 복사1.xlsx") # 파일 저장
```

	A	B	C	D	E	F
1	이름	나이	성별			
2	이순신	55	남자			
3	홍길동	40	남자			
4						
5	이순신					
6	이순신	이순신	이순신			
7						
8						

● 셀 복사하기 : 여러 셀의 내용을 읽어와 복사해 넣기

```
wb = load_workbook("chapter07/테스트.xlsx")  
ws = wb["시트1"]
```

테스트.xlsx 파일을 불러와 wb 객체에 저장
시트 선택

```
src_data = []  
for cell in ws[2]:  
    src_data.append(cell.value)
```

```
['이순신']  
['이순신', '55']  
['이순신', '55', '남자']
```

복사할 값을 저장할 변수 선언
원본이 될 2번째 행에 접근
2번째 행의 셀값들을 src_data 변수에 저장

```
for row in ws["A5:C5"]:  
    for cell in row:  
        cell.value = src_data[cell.col_idx - 1]
```

A5에서 C5까지 영역을 선택해 반복문 실행
행의 각 셀에 접근
셀의 값을 src 변수 값으로 변경

```
wb.save(filename = "chapter07/테스트_셀 복사2.xlsx") # 파일 저장
```

	A	B	C	D	E	F
1	이름	나이	성별			
2	이순신	55	남자			
3	홍길동	40	남자			
4						
5	이순신	55	남자			
6						
7						
8						

● 셀 내용 이동하기

- 셀 이동 : **move_range**(셀범위, row, col)
 - 특정 영역의 셀 내용을 상하좌우 원하는 방향으로 이동

셀 내용 이동하기

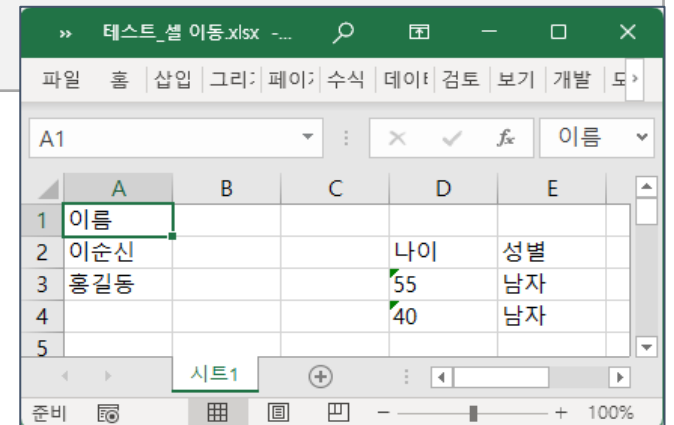
```
from openpyxl import load_workbook
wb = load_workbook("chapter07/테스트.xlsx")
ws = wb["시트1"]
```

```
# Workbook 모듈과 load_workbook 불러오기
# "테스트.xlsx" 파일을 불러와 wb 객체에 저장
# 시트 선택
```

[B1:C3] 영역에 있는 데이터를 아래로 한 칸, 오른쪽으로 2칸 이동

```
ws.move_range("B1:C3", rows = 1, cols = 2)
wb.save(filename = "chapter07/테스트_셀 이동.xlsx")
```

```
# 음수값을 사용해 위나 왼쪽으로도 이동 가능
# 파일 저장
```



● 셀 병합 및 해제하기

➤ 셀 병합 및 해제 : `merge_cells()`, `unmerge_cells()` 함수

```
from openpyxl import load_workbook
wb = load_workbook("chapter07/테스트.xlsx")
ws = wb["시트1"]
```

```
# Workbook모듈과 load_workbook 불러오기
# 테스트.xlsx 파일을 불러와 wb 객체에 저장
# 시트 선택
```

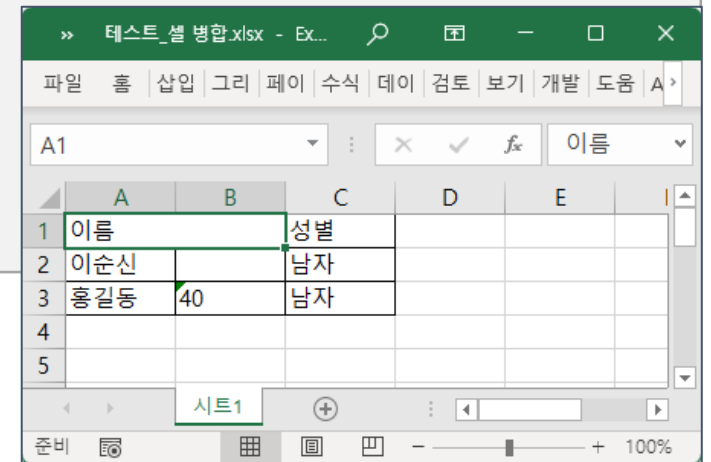
```
ws.merge_cells("A1:B1")
ws.merge_cells("A2:B2")
```

```
# A1, B1 셀 병합
# A2, B2 셀 병합
```

```
ws.unmerge_cells("A2:B2")
```

```
# A2, B2 셀 병합해제
```

```
wb.save(filename = "chapter07/테스트_셀 병합.xlsx") # 파일 저장
# A1:B1 셀은 병합 되어 A1셀 값만 보이게 되고,
# A2:B2 셀은 병합 후 다시 병합해제 → A2 셀의 값만 보이게 됨
```



● 텍스트 형식을 숫자 형식으로 바꾸고 엑셀 함수로 연산

➢ 텍스트로 설정된 셀 표시 형식을 숫자형으로 바꾸어 합계/평균 등의 계산을 할 수 있도록 함

- 정수형(**i**), 문자형(**s**), 실수형(**f**)

```
from openpyxl import load_workbook
wb = load_workbook("chapter07/테스트.xlsx")
ws = wb["시트1"]
```

```
# Workbook모듈과 load_workbook 불러오기
# "테스트.xlsx" 파일을 불러와 wb 객체에 저장
# 시트 선택
```

```
ws["B2"].data_type = "i"
```

```
# [B2]셀을 정수형으로 변환
```

```
ws["B3"].data_type = "i"
```

```
# [B3]셀을 정수형으로 변환
```

```
ws["A4"] = "나이 합계"
```

```
# [A4]셀에 데이터 입력
```

```
ws["B4"] = "=sum(B2:B3)"
```

```
# [B4]셀에 엑셀의 sum 함수 직접 입력
```

```
wb.save(filename = "chapter07/테스트_셀 연산.xlsx")
```

	A	B	C	D	E	F
1	이름	나이	성별			
2	이순신	55	남자			
3	홍길동	40	남자			
4	나이 합계	95				
5						
6						

● 셀 배경과 폰트 서식 변경하기

➤ openpyxl.styles 패키지의 **PatternFill**, **Font**, **Color** 클래스를 이용해 배경/폰트/색상 서식 변경

```
from openpyxl import load_workbook          # Workbook모듈과 load_workbook 불러오기
from openpyxl.styles import Font, PatternFill  # 폰트와 배경 설정 클래스 불러오기
wb = load_workbook("chapter07/테스트.xlsx")    # 테스트.xlsx 파일을 불러와 wb 객체에 저장
ws = wb["시트1"]                             # 시트 선택
# [A1:C1] 배경색은 검은색, 폰트는 크기 12, 볼드체, 색상은 흰색으로 변경(색 정보는 RGB HEX 코드로 지정)
fill_style = PatternFill(fill_type="solid", start_color="000000") # 셀 배경 설정
font_style = Font(color="FFFFFF", sz=12, bold=True)                # 폰트 설정(흰색, 12, 굵게)

for row in ws:
    for cell in row:
        if cell.row == 1:
            cell.fill = fill_style
            cell.font = font_style
wb.save(filename = "chapter07/테스트_배경 서식 변경.xlsx")
```

ws의 행을 하나씩 읽어 옴
각 행의 컬럼을 하나씩 읽어 옴
셀 행이 1일 때 True가 되어 속성 변경 가능
셀의 배경 속성을 fill_style로 설정
셀의 폰트 속성을 font_style로 설정

	A	B	C	D	E	F
1	이름	나이	성별			
2	이순신	55	남자			
3	홍길동	40	남자			
4						
5						

● 셀 높이와 너비 조정하기

- styles 패키지 로드 없이 워크시트 객체의 **row_dimensions** 와 **column_dimensions** 멤버변수로 셀 높이/너비 조정 가능

```
from openpyxl import load_workbook                    # 모듈 불러오기

wb = load_workbook("chapter07/테스트_배경서식변경.xlsx")  # 파일 불러와서 wb 객체에 저장
ws = wb["시트1"]                                         # 시트 선택

# 1~5행의 높이를 24로, 1열의 너비를 14로 변경
for i in range(1, 6):
    ws.row_dimensions[i].height = 24                    # row_dimensions[i] → i 행 전체를 선택
    ws.column_dimensions["A"].width = 14                # column_dimensions["A"] → A열 전체를 선택

wb.save(filename = "chapter07/테스트_너비와 높이 조정.xlsx")
```

	A	B	C	D
1	이름	나이	성별	
2	이순신	55	남자	
3	홍길동	40	남자	
4				
5				
6				
7				
8				

● 텍스트 정렬하기

- openpyxl.styles 패키지의 **Alignment** 클래스로 가로(왼쪽/중앙/오른쪽), 세로(위쪽/중앙/아래쪽) 정렬 가능

```
from openpyxl import load_workbook # 모듈 불러오기
from openpyxl.styles import Alignment # 텍스트 정렬 모듈 불러오기

wb = load_workbook("chapter07/테스트_너비와 높이 조정.xlsx") # 파일 불러와서 wb 객체에 저장
ws = wb["시트1"] # 시트 선택

#가로 정렬값과 세로 정렬값을 중앙으로 변경하는 객체 생성
alignment_style = Alignment(horizontal="center", vertical="center")

for row in ws: # ws의 행을 하나씩 읽어 옴
    for cell in row: # 각 행의 셀을 하나씩 읽어 옴
        cell.alignment = alignment_style # 셀의 정렬 속성을 설정

wb.save(filename = "chapter07/테스트_텍스트 정렬.xlsx")
```

	A	B	C	D	E	F
1	이름	나이	성별			
2	이순신	55	남자			
3	홍길동	40	남자			
4						

● 셀 테두리 서식 변경하기

➤ openpyxl.styles 패키지의 **Side** 클래스와 **Border** 클래스를 사용해 테두리 서식 설정

```
from openpyxl.styles import Side, Border          # 테두리 설정 모듈 불러오기
wb = load_workbook("chapter07/테스트_텍스트 정렬.xlsx")  # 파일 불러와서 wb 객체에 저장
ws = wb["시트1"]                                   # 시트 선택

side_style = Side(style="medium", color="000000")      # 테두리를 중간 굵기에 검정색으로 설정

# 위 서식(side_style)을 적용할 테두리 영역을 설정 후, border_styles 객체로 저장
border_styles = Border(left=side_style, right=side_style, top=side_style, bottom=side_style)

for row in ws:                                     # ws의 행을 하나씩 읽어 옴
    for cell in row:                                # 각 행의 셀을 하나씩 읽어 옴
        cell.border = border_styles                 # 셀의 테두리 속성을 설정

wb.save(filename = "chapter07/테스트_셀 테두리 변경.xlsx")
```

	A	B	C	D	E	F
1	이름	나이	성별			
2	이순신	55	남자			
3	홍길동	40	남자			
4						

- ◆ XlsxWriter는 엑셀 파일을 **만드는 기능(쓰기)**만 있음
 - 그러나 pandas의 read_excel() 을 이용하면 엑셀 파일을 읽을 수 있다.

- ◆ xlsxwriter 설치하기

```
!pip install xlsxwriter
```

◆ 기본 코드 구조

```
import xlsxwriter
```

```
# 1) 생성할 엑셀 파일 이름을 지정해 워크북 객체 생성
```

```
workbook = xlsxwriter.Workbook("엑셀파일")
```

```
# 2) 워크북 내에 사용할 워크시트 생성
```

```
worksheet = workbook.add_worksheet("시트이름")
```

```
# 3) 워크시트의 셀에 쓰기 작업 수행
```

```
worksheet.write(row, col, "셀 데이터")
```

```
# 셀 행과 열의 위치로 지정
```

혹은

```
worksheet.write("셀 주소", "셀 데이터")
```

```
# 셀 주소로 지정
```

```
# 4) 워크북 객체를 닫고 엑셀 파일 생성
```

```
workbook.close()
```

◆ 엑셀 파일 생성하기

```
import xlsxwriter
```

```
# 엑셀 파일 경로
```

```
folder = 'chapter07/'
```

```
file = folder + 'Xlsx_01.xlsx'
```

```
wb = xlsxwriter.Workbook(file)
```

```
ws = wb.add_worksheet()
```

```
ws.write(0, 0, 100)
```

```
ws.write('A2', 200)
```

```
wb.close()
```

```
print("생성한 엑셀 파일:", file)
```

```
# 워크북 객체 생성
```

```
# 워크시트 생성
```

```
# 셀의 행과 열의 위치를 지정해 셀에 데이터 쓰기
```

```
# 셀의 주소 지정 후 셀에 데이터 쓰기
```

```
# 워크북 객체를 닫고 엑셀 파일 생성
```

```
# 생성한 파일 이름 출력
```

	A	
1	100	
2	200	
3		

◆ 리스트 자료형 데이터 쓰기

```
import xlsxwriter
```

```
wb = xlsxwriter.Workbook("chapter07/Xlsx_02.xlsx")
```

```
ws = wb.add_worksheet()
```

```
# 리스트 자료형 쓰기
```

```
list_num = [10, 20, 30, 40]
```

```
for col_num, value in enumerate(list_num):
```

```
    ws.write(0, col_num, value)
```

```
list_num2 = [50, 60, 70, 80]
```

```
ws.write_row(1, 0, list_num)
```

```
ws.write_column(2, 0, list_num2)
```

```
wb.close()
```

```
# 워크북 객체 생성
```

```
# 워크시트 생성
```

	A	B	C	D
1	10	20	30	40
2	10	20	30	40
3	50			
4	60			
5	70			
6	80			

```
# 행 번호 1(인덱스 0) 은 고정, 열 번호는 증가
```

```
# 셀 A2에서 시작해 행 방향으로 쓰기
```

```
# 셀 A3에서 시작해 열 방향으로 쓰기
```

```
# 워크북 객체를 닫고 엑셀 파일 생성
```

◆ 딕셔너리 자료형 데이터 쓰기

딕셔너리 데이터 생성

```
dict_data = { '제품ID': ['P1001', 'P1002', 'P1003', 'P1004'],  
              '판매가격': [5000, 7000, 8000, 10000],  
              '판매량': [50, 93, 70, 48] }
```

dict_data

```
{'제품ID': ['P1001', 'P1002', 'P1003', 'P1004'],  
 '판매가격': [5000, 7000, 8000, 10000],  
 '판매량': [50, 93, 70, 48]}
```

◆ 딕셔너리 자료형 데이터 쓰기

```
import xlsxwriter
```

```
wb = xlsxwriter.Workbook("chapter07/Xlsx_03.xlsx")  
ws = wb.add_worksheet()
```

```
list_keys = list(dict_data.keys())  
list_values = list(dict_data.values())
```

```
ws.write_row(0, 0, list_keys)
```

```
# 두 번째 행에 리스트 데이터를 열 방향으로 쓰기  
for col, list_value in enumerate(list_values):  
    ws.write_column(1, col, list_value)
```

```
wb.close()
```

워크북 객체 생성

워크시트 생성

딕셔너리 키를 추출해 리스트로 변환

딕셔너리 값을 추출해 리스트로 변환

첫 번째 행에 키를 행 방향으로 쓰기

	A	B	C
1	제품ID	판매가격	판매량
2	P1001	5000	50
3	P1002	7000	93
4	P1003	8000	70
5	P1004	10000	48
6			

워크북 객체를 닫고 엑셀 파일 생성

◆ 판다스 DataFrame 데이터 쓰기 기본 구조

```
df.to_excel("엑셀파일"  
            [, index = True(기본)/False,  
              header = True(기본)/False,  
              sheet_name = "시트이름"(기본: 'Sheet1') 혹은 시트_번호(기본: 0),  
              startrow = 숫자(기본: 0),  
              startcol = 숫자(기본: 0)] [,options])
```

- ◆ 여러 개의 DataFrame 데이터를 하나의 엑셀 파일에 쓰거나, 서식 지정, 그림 추가 등의 작업을 진행하려면 ExcelWriter로부터 객체를 생성해 to_excel()을 사용

```
import pandas as pd
```

```
df = pd.read_csv("chapter07/korea_rain.csv") # CSV 파일을 읽어와서 DataFrame 데이터 생성  
df
```

	연도	봄	여름	가을	겨울
0	2014	215.9	599.8	293.1	76.9
1	2015	223.2	387.1	247.7	109.1
2	2016	312.8	446.2	381.6	108.1
3	2017	118.6	609.7	172.5	75.6
4	2018	368.1	586.5	351.2	66.5

◆ 여러 개의 DataFrame 데이터

```
# 1) 쓰기 엔진을 xlsxwriter로 지정해 판다스의 ExcelWriter로부터 객체(excel_writer) 생성  
excel_writer = pd.ExcelWriter("chapter07/Xlsx_DF_01.xlsx", engine='xlsxwriter')
```

```
# 2) 생성한 엑셀 객체에 DataFrame 데이터(df)를 쓰기(시트이름 지정)  
df.to_excel(excel_writer, sheet_name='Sheet1')
```

```
# 3) 객체를 닫고 엑셀 파일로 저장  
excel_writer.save()
```

	A	B	C	D	E	F
1		연도	봄	여름	가을	겨울
2	0	2014	215.9	599.8	293.1	76.9
3	1	2015	223.2	387.1	247.7	109.1
4	2	2016	312.8	446.2	381.6	108.1
5	3	2017	118.6	609.7	172.5	75.6
6	4	2018	368.1	586.5	351.2	66.5
7						

◆ 셀 서식 지정하는 코드 구조

```
import xlsxwriter
```

```
# 1) 생성할 엑셀 파일 이름을 지정해 워크북 객체 생성  
workbook = xlsxwriter.Workbook("엑셀파일")
```

```
# 2) 워크북 내에 사용할 워크시트 생성  
worksheet = workbook.add_worksheet("시트이름")
```

```
# 3) 셀 서식 지정을 위한 서식 객체 생성 및 서식 지정  
cell_format = workbook.add_format([properties])  
[cell_format.method()]
```

```
# 4) 워크시트의 셀에 서식을 지정해 쓰기 작업 수행  
worksheet.write(row, col, "셀 데이터", cell_format)  
        혹은  
worksheet.write("셀 주소", "셀 데이터", cell_format))
```

```
# 5) 워크북 객체를 닫고 엑셀 파일 생성  
workbook.close()
```

생략시 "Sheet1" 부터 자동 생성

속성을 지정
메서드 서식 추가 지정

셀 행과 열의 위치로 지정

셀 주소로 지정

◆ 셀 서식 지정

```
import xlsxwriter
# 1) 생성할 엑셀 파일이름을 지정해 워크북 객체 생성
wb = xlsxwriter.Workbook('chapter07/Xlsx_cellFormat.xlsx')

# 2) 워크북 내에 사용할 워크시트 생성
ws = wb.add_worksheet()

# 3) 셀 서식 지정을 위한 서식 객체 생성
# 속성: 텍스트 굵게, 글꼴 색은 파란색으로 속성을 지정해 서식 객체 생성
cell_format = wb.add_format({'font_name': '바탕', 'bold': True, 'font_color': 'blue'})

# 메서드를 이용해 서식 추가 지정(텍스트를 기울임꼴로 설정)
cell_format.set_italic()

# 메서드를 이용해 서식 추가 지정(글꼴 크기 지정)
cell_format.set_font_size(20)

# 4) 워크시트의 셀에 쓰기 작업 수행
ws.write('A1', "셀 서식 미지정")          # 서식 미지정
ws.write('A2', "셀 서식 지정", cell_format) # 서식 지정

# 5) 워크북 객체를 닫고 엑셀 파일 생성
wb.close()
```

	A	B	C
1	셀 서식 미지정		
2	<i>셀 서식 지정</i>		
3			

◆ 셀에 그림 삽입 형식

4) 워크시트의 셀에 쓰기와 이미지 삽입 작업 수행

```
worksheet.insert_image(row, col, image_file[, options])
```

혹은

offset : 지정한 셀의 좌측 상단에서 그림의 좌측 상단 까지의 가로와 세로 위치 차이

```
worksheet.insert_image(cell_address, image_file {'x_offset': 25, 'y_offset': 10})
```

◆ 셀에 그림 삽입

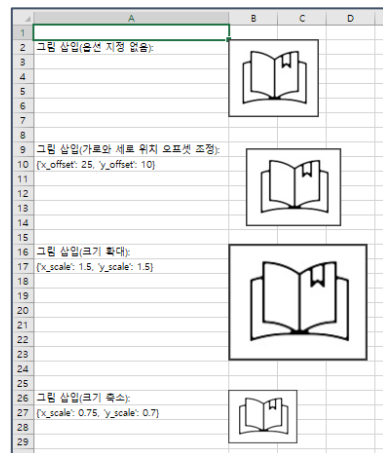
```
import xlsxwriter
```

```
image_file = 'images/book_image.png'    # 그림 파일
```

```
# 1) 생성할 엑셀 파일이름을 지정해 워크북 객체 생성  
wb = xlsxwriter.Workbook('chapter07/Xlsx_image.xlsx')
```

```
# 2) 워크북 내에 사용할 워크시트 생성  
ws = wb.add_worksheet()
```

```
# 3) 필요 시 행과 열의 높이 지정  
ws.set_column(0, 0, 35) # 0열 (A열)의 너비 지정
```



```
# 4) 워크시트의 셀에 쓰기와 이미지 삽입 작업 수행
```

```
ws.write(1, 0, "그림 삽입(옵션 지정 없음):")
```

```
ws.insert_image(1, 1, image_file)
```

```
ws.write(8, 0, "그림 삽입(가로와 세로 위치 오프셋 조정):")
```

```
ws.write(9, 0, "{ 'x_offset': 25, 'y_offset': 10}")
```

```
ws.insert_image(8, 1, image_file, { 'x_offset': 25, 'y_offset': 10})
```

```
ws.write(15, 0, "그림 삽입(크기 확대):")
```

```
ws.write(16, 0, "{ 'x_scale': 1.5, 'y_scale': 1.5}")
```

```
ws.insert_image(15, 1, image_file, { 'x_scale': 1.5, 'y_scale': 1.5})
```

```
ws.write(25, 0, "그림 삽입(크기 축소):")
```

```
ws.write(26, 0, "{ 'x_scale': 0.75, 'y_scale': 0.7}")
```

```
ws.insert_image(25, 1, image_file, { 'x_scale': 0.75, 'y_scale': 0.7})
```

```
# 5) 워크북 객체를 닫고 엑셀 파일 생성
```

```
wb.close()
```

시스템 제어를 위한 pyautogui

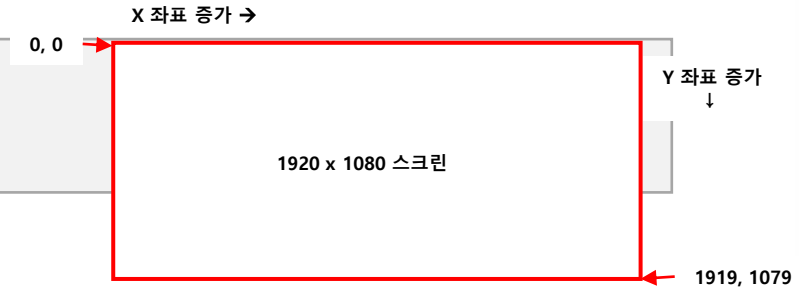
- ◆ 업무를 자동화하려면 마우스나 키보드를 컨트롤하거나 마우스의 좌표 인식 등으로 실행 중인 프로그램을 제어할 수 있어야 한다.
 - 이것이 업무 자동화(**RPA, Robotic Process Automation**)의 핵심 기술
- ◆ **pyautogui**는 마우스와 키보드를 제어하여 다른 응용 프로그램과의 상호작용을 자동화할 수 있도록 해주며 윈도우, 리눅스, 맥 OS에서도 작동
- ◆ **pyautogui**의 주요 기능
 - 마우스를 이동하고 다른 응용 프로그램의 창을 클릭
 - 응용 프로그램에 키보드로 내용 입력
 - 스크린 샷을 찍고 이미지가 주어지면 화면에서 탐색(예: 버튼 또는 확인란)
 - 응용 프로그램의 창을 찾아 이동, 크기 조정, 최대화, 최소화 또는 닫기
 - GUI 자동화 스크립트가 실행되는 동안 사용자 상호작용을 위한 메시지 상자 표시
- ◆ **pyautogui** 설치

```
!pip install pyautogui
```


● 마우스의 현재 좌표를 파악하기

- size() 함수 : 화면 해상도 크기를 알아볼 수 있으며 결과값은 두 정수의 튜플로 반환

```
import pyautogui          # pyautogui 패키지 불러오기
pyautogui.size()          # 출력결과 → Size(width=1920, height=1080)
```



- position() 함수 : 마우스 커서의 **현재** X 및 Y 좌표를 반환

```
pyautogui.position()      # 출력결과 → Point(x=2270, y=106)
```

- 코드 실행 시 마우스가 에디터의 실행 버튼을 클릭하기 위해 이동하면, 사용자가 원하는 위치의 좌표를 얻을 수 없게 된다.
- 이를 보완하기 위해 **time.sleep()** 함수로 대기 시간을 주면, 코드 실행 후 사용자가 원하는 위치까지 이동하는 동안 대기 시간을 벌기 때문에 원하는 마우스 위치의 좌표를 얻을 수 있다.

```
import time                # time 패키지 로드
time.sleep(3)              # 3초 동안 대기(원하는 위치로 마우스를 이동)
pyautogui.position()       # 마우스 포지션 반환
```

● 마우스 이동/드래그 하기

- moveTo() 함수 : 마우스의 커서가 지정한 X, Y 좌표로 즉시 이동
 - 시간을 두어 천천히 이동하려면 이동 시간을 지정

```
pyautogui.moveTo(100, 200)      # 마우스 커서가 x, y 좌표(100, 200)로 이동
pyautogui.moveTo(100, 400, 3)    # 마우스 커서가 x, y 좌표(100, 400)로 3초 동안 이동
```

- move() 함수 : **현재 위치를 기준으로** 입력한 좌표만큼 마우스 커서를 이동

```
pyautogui.move(0, 200)          # 현재 위치에서 아래로(y좌표만) 200 이동
pyautogui.move(100, 200)        # 현재 위치에서 우측으로(x좌표) 100, 아래로(y좌표) 200 이동
pyautogui.move(100, 200, 3)     # 현재 위치에서 우측으로 100, 아래로 200, 3초 동안 이동
```

- dragTo(), drag() 함수 : 드래그 기능으로, button 속성으로 마우스 버튼(left, right, middle)을 지정 가능
 - dragTo() : 현재 위치에서 특정 위치로 드래그하면서 이동
 - drag() : 현재 위치에서 상대적으로 특정 픽셀만큼 그냥 이동

```
pyautogui.dragTo(500, 500, button="left")  # 현재 위치에서 x, y(500, 500) 좌표로 드래그
pyautogui.dragTo(300, 300, 2, button="left") # 현재 위치에서 x, y(300, 300) 좌표로 2초 동안 드래그
pyautogui.drag(100, 0, 2, button="left")   # 현재 위치에서 우측으로 100, 2초 동안 드래그
```

● 마우스로 클릭하기

- click() 함수 : 현재 위치에서 마우스 왼쪽/오른쪽 버튼을 한 번 클릭하는 기능
 - 이 함수를 실행하기 전에 마우스를 클릭해야 하는 메뉴 또는 버튼 위치로 이동한 후 실행하면 자동으로 클릭 확인

pyautogui.click()	#현재 위치에서 마우스 버튼 클릭
pyautogui.click(x=100, y=200)	# x, y(100, 200) 좌표를 클릭
pyautogui.click(button="right")	# 우측 마우스 버튼 클릭
pyautogui.click(clicks=2)	# 2번 클릭으로 더블클릭과 동일
pyautogui.click(clicks=2, interval=0.25)	# 더블 클릭을 하되 클릭 사이에 0.25초 간격 지정
pyautogui.click(button="right", clicks=3, interval=0.25)	# 우측 버튼을 0.25초 간격으로 3번 클릭

※**doubleClick()**, **tripleClick()** 함수로 더블클릭/세번 클릭 대체 가능

- mouseDown(), mouseUp() 함수 : 버튼을 누른 후 다시 놓지 않고 누르거나 놓는 동작 하나만 수행

pyautogui.mouseDown()	# 현재 위치에서 좌측 마우스 버튼 누르기
pyautogui.mouseUp()	# 현재 위치에서 눌려진 좌측 마우스 버튼 놓기
pyautogui.mouseDown(button="right")	# 현재 위치에서 우측 마우스 버튼 누르기
pyautogui.mouseUp(button="right", x=200, y=300)	# 지정 좌표에서 우측 마우스 버튼 놓기

● 마우스로 스크롤하기

➤ `scroll()` 함수 : 마우스의 스크롤을 동작시켜 화면을 이동시키는 기능

```
pyautogui.scroll(10)
```

```
# 10 클릭만큼 위로 스크롤
```

```
pyautogui.scroll(-15)
```

```
# 15 클릭만큼 아래로 스크롤
```

```
pyautogui.scroll(10, x=200, y=200)
```

```
# x, y(200, 200) 좌표에서 10 클릭만큼 위로 스크롤
```

● 문자열과 특정 키 입력

- 이 슬라이드 내용은 새로운 노트북 파일을 열어 테스트 한다
- write() 함수 : 키보드로 전달되는 문자열을 입력하는 함수로 각 문자 키를 누르는 시간의 간격을 두려면 interval 옵션을 사용

```
import pyautogui  
pyautogui.write("Hello")
```

입력 후 [▶ Run] 을 눌러 실행 시킨다



- 다음 셀에 "hello" 가 출력된다

```
1 import pyautogui  
2 pyautogui.write("Hello")
```



```
1 Hello
```

```
pyautogui.write("Good morning", interval=0.25) # 0.25초의 간격을 두고 각 문자가 입력됨
```

```
1 pyautogui.write("Good Morning", interval=0.25)
```

```
1 Good Morning
```

● 문자열과 특정 키 입력

- pyautogui 는 한글이 적용되지 않으므로 복사하여 붙여 넣기를 이용한다.

```
import pyperclip
pyperclip.copy("안녕하세요")      # 클립보드에 텍스트를 복사
pyautogui.hotkey("ctrl", "v")      # 붙여 넣기
```

```
1 import pyperclip
2 pyperclip.copy("안녕하세요")
3 pyautogui.hotkey("ctrl", "v")
```



```
1 안녕하세요|
```

- hotkey() 함수 : 여러 키를 동시에 입력해야 할 때, 윈도우의 단축키를 사용할 수 있는 함수

```
pyautogui.hotkey("ctrl", "c")      # [ctrl + c] 키를 입력
```

키보드 제어함수에 사용할 수 있는 키의 공식 문서

<https://pyautogui.readthedocs.io/en/latest/keyboard.html>

● 문자열과 특정 키 입력

- `keyDown()`, `keyUp()` 함수 : 특정 키를 누르고 있거나 떼는 기능으로 `[shift]`, `[alt]`, `[ctrl]` 키를 누를 때 많이 사용

- 실행 시키지 말것

```
pyautogui.keyDown("shift")    # [shift] 키를 누른 상태를 유지
pyautogui.keyUp("shift")      # 누른 [shift] 키를 땀
```

- `press()` 함수 : 키보드의 키를 눌렀다가 떼는 함수로 `keyDown()`과 `keyUp()`을 함께 사용한 것과 동일

```
pyautogui.press("shift")      # [shift] 키를 눌렀다가 땀
```

#아래와 같이 `keyDown()`, `keyUp()`, `press()`함수를 조합해 단축키를 구현하는 것도 가능

```
pyautogui.keyDown("ctrl")     # [ctrl] 키를 누르고 있음
pyautogui.press("c")          # [c] 키를 한 번 눌렀다가 땀
pyautogui.keyUp("ctrl")       # 누르고 있던 [ctrl] 키를 땀
```

● 문자열과 특정 키 입력

- keyDown(), keyUp(), press() 함수의 조합 사용

```
pyautogui.press("shift")
```

```
# [shift] 키를 누름(눌렀다가 땀)
```

- [Shift] 키를 누른 상태에서 [←] 키를 3번 누른 경우

```
pyautogui.keyDown("shift")
```

```
# [shift] 키를 누르고 있음
```

```
pyautogui.press("left")
```

```
# [←] 키를 한 번 눌렀다가 땀
```

```
pyautogui.press("left")
```

```
# [←] 키를 한 번 눌렀다가 땀
```

```
pyautogui.press("left")
```

```
# [←] 키를 한 번 눌렀다가 땀
```

```
pyautogui.keyUp("shift")
```

```
# 누르고 있던 [shift] 키를 땀
```

- 한번에 해결하는 방법

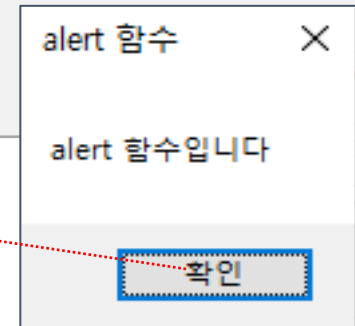
```
pyautogui.press(["left", "left", "left"])
```

```
# 왼쪽 화살표 키 입력을 세 번 반복
```


- 메시지를 표시하거나 승인/취소, 데이터 입력 받기 등 사용자와 상호작용하는 대화상자 사용 가능

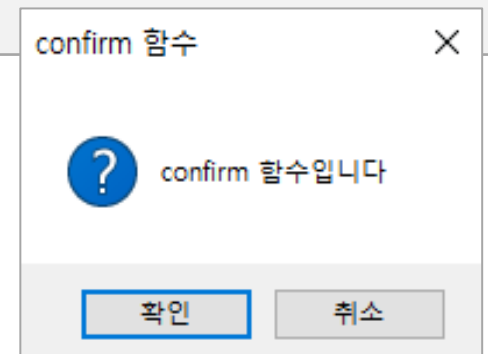
- alert() 함수 : [확인] 버튼이 있는 간단한 **경고** 메시지 박스를 표시하며 클릭한 **버튼의 텍스트를 반환**

```
pyautogui.alert(text="alert 함수입니다", title="alert 함수", button="OK")  
# [확인]을 클릭하면 'OK'가 출력됨
```



- confirm() 함수 : [확인] 과 [취소] 버튼이 있는 **확인** 메시지 박스를 표시

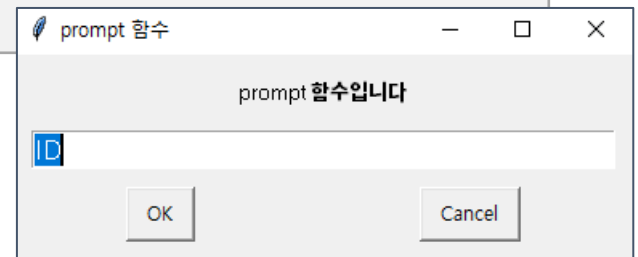
```
pyautogui.confirm(text="confirm 함수입니다", title="confirm 함수", buttons=["OK", "Cancel"])  
# [확인]을 클릭하면 'OK'가, [취소]를 클릭하면 'Cancel'이 출력됨
```



- 메시지를 표시하거나 승인/취소, 데이터 입력 받기 등 사용자와 상호작용하는 대화상자 사용 가능
 - prompt() 함수: 텍스트 입력과 [확인] 및 [취소] 버튼이 있는 메시지 박스를 표시
 - [ok] 버튼은 입력한 텍스트를 반환, [cancel] 버튼을 클릭한 경우 None을 반환

텍스트를 입력하면 입력한 텍스트를 반환함

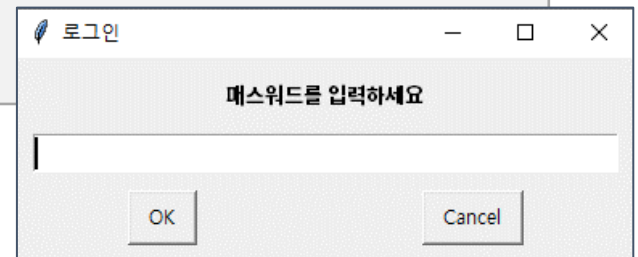
```
pyautogui.prompt(text="prompt 함수입니다", title="prompt 함수", default="ID")
```



- password() 함수 : 입력한 문자는 *로 나타나며, 패스워드 등을 입력할 때 주로 사용

패스워드 입력 후 OK 클릭하면 입력내용 출력 됨

```
pyautogui.password(text="패스워드를 입력하세요", title="로그인", mask="*")
```



- ◆ 컴퓨터 화면을 캡처하거나 코드에서 지정한 이미지를 화면에서 찾을 수 있으며, 마우스/키보드 기능과 함께 사용하여 마치 눈으로 보고 필요한 것을 클릭하는 것처럼 자동으로 작업이 가능

➤ **pillow** 설치하기 : 파이썬에서 이미지 입/출력 기능을 사용하기 위해 이미지 패키지인 pillow 설치

```
!pip install pillow
```

➤ pillow 패키지 불러오기

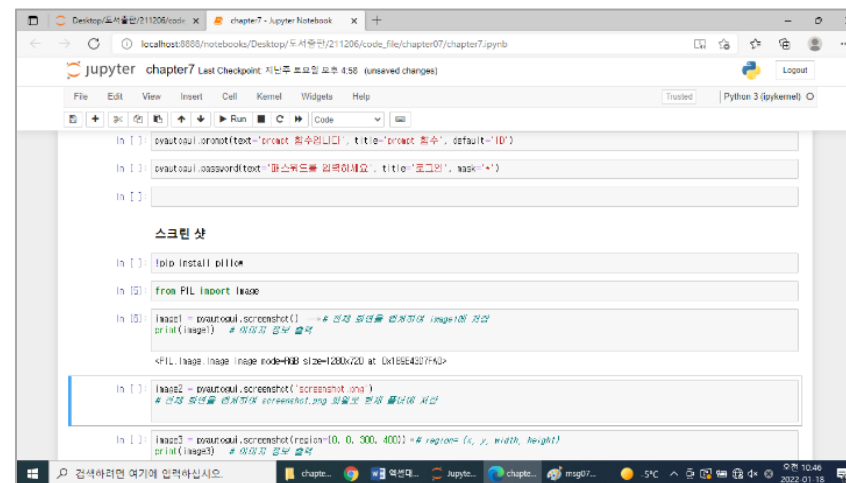
```
from PIL import Image
```

#Image의 'I'가 대문자임에 주의

- screenshot() 함수 : 화면 전체 또는 일부를 캡처하고 파일로 저장 가능

```
image1 = pyautogui.screenshot()    # 전체 화면을 캡처하여 image1에 저장
print(image1)                      # 이미지 정보 출력
#출력결과 → <PIL.Image.Image image mode=RGB size=1920x1080 at 0x1CE18460790>
```

```
# 전체 화면을 캡처하여 지정한 폴더에 'screenshot.png' 파일로 저장
image2 = pyautogui.screenshot("chapter07/screenshot.png")
```



- screenshot() 함수 : 특정 영역의 스크린 샷을 원하는 경우에는 **region** 옵션을 사용

```
image3 = pyautogui.screenshot(region=(0, 0, 300, 400))    # region= (x, y, width, height)
print(image3)                                              # 이미지 정보 출력
#출력결과 → <PIL.Image.Image image mode=RGB size=300x400 at 0x1CE1D5539A0>

#변수에 저장된 이미지는 matplotlib 패키지를 불러와 plt.imshow() 함수로 확인 가능
import matplotlib.pyplot as plt                          # matplotlib 패키지 불러오기
plt.imshow(image3)                                         # plt.imshow 함수로 이미지 출력하기
```

