

01. Python 개요



연세대학교
YONSEI MIRAE
CAMPUS

목차	학습내용
사전 작업	파이썬 및 편집 프로그램 설치
1. 파이썬 개요	파이썬 개요 및 특징
2. 자료형	변수와 상수, 숫자, 문자열, 리스트, 튜플, 딕셔너리 자료형
3. 제어문과 함수	조건문과 반복문, 함수와 클래스 이해하기
4. 엑셀을 위한 기본함수 구현하기	pandas 기초, 텍스트 함수, 수학 및 통계 함수 구현
5. 업무에 자주 쓰는 실무 함수 구현하기	동적 배열 함수, 찾기 및 참조 함수, 논리 및 정보 함수 구현
6. 그래프 함수로 시각화하기	matplotlib, pandas 로 그래프 그리기
7. 문서 업무 자동화	openpyxl, xlsxwriter 로 엑셀 파일 다루기, pyautogui 로 자동화하기
8. 웹 크롤링	BeautifulSoup, selenium 으로 웹 데이터 자동 수집하기
9. 실전 자동화 프로젝트	청구서 자동 발행 프로젝트 따라하기

- 1989년 네덜란드의 귀도 반 로섬(Guido van Rossum)이 개발
- 직관적이고 쉬운 언어
 - 다른 프로그래밍 언어에 비해 배우기 쉬운 문법 체계
 - 간단한 영어 문장을 읽듯이 쉽게 이해 가능
- 오픈 소스 소프트웨어
 - 누구나 무료로 사용 가능하고, 소프트웨어 복사본 배포 가능
 - 공개된 소스 코드 활용 및 수정 가능
- 확장 가능한 라이브러리
 - 다양한 명령어, 함수들을 해석하고 실행하는 표준 라이브러리 제공
 - 특정 기능(데이터 분석, 머신러닝, 게임 개발 등)을 수행하는 다양한 외부 라이브러리 설치 가능
- 우수한 이식성
 - 변환과정 없이 다양한 플랫폼에서 이용 가능
(즉, 윈도우에서 작성한 파이썬 프로그램은 리눅스나 맥 OS에서도 동일하게 사용 가능)

● 파이썬으로 엑셀기능 구현

- 파이썬의 **pandas**나 **openpyxl** 패키지를 활용하면 엑셀 대부분의 기능들을 쉽게 구현할 수 있고, 속도도 더 빠름

● 파이썬과 엑셀 비교

- 사례) 학생 20명의 5과목 성적 분석

엑셀

VS

파이썬

구분	국어	영어	수학	사회	과학
갯 수	20.00	20.00	20.00	20.00	20.00
평균	75.4	69.75	71.15	74.3	75
표준편차	15.76	14.70	16.05	13.22	12.88
최소값	52.00	51.00	51.00	54.00	52.00
1사분위	61.25	56.5	58.75	61.75	65
2사분위	79.00	67.00	65.00	77.00	74.00
3사분위	88.00	80.25	86.50	85.00	85.25
최대값	96.00	97.00	97.00	91.00	97.00

	국어	영어	수학	사회	과학
count	20.00	20.00	20.00	20.00	20.00
mean	75.40	69.75	71.15	74.30	75.00
std	15.76	14.70	16.05	13.22	12.88
min	52.00	51.00	51.00	54.00	52.00
25%	61.25	56.50	58.75	61.75	65.00
50%	79.00	67.00	65.00	77.00	74.00
75%	88.00	80.25	86.50	85.00	85.25
max	96.00	97.00	97.00	91.00	97.00

```
개수 = COUNT(C4:C23)
평균 = AVERAGE(C4:C23)
표준편차 = STDEV(C4:C23)
최소값 = MIN(C4:C23)
제1사분위 = QUARTILE(C4:C23,1)
제2사분위 = QUARTILE(C4:C23,1)
제3사분위 = QUARTILE(C4:C23,1)
최대값 = MAX(C4:C23)
```

```
df.describe()
```

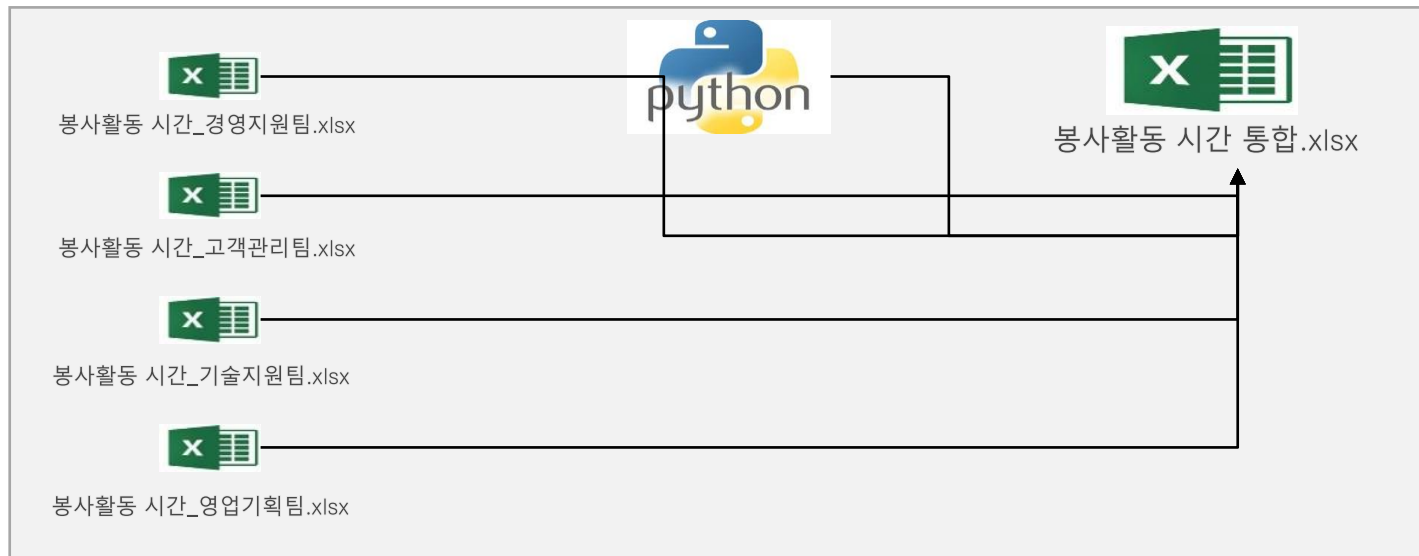
※ 엑셀: 매번 함수, 범위 지정 필요 vs 파이썬: 명령어 한 줄

● 업무 자동화(RPA: Robotic Process Automation)가 각광받고 있음

- 반복적인 업무프로세스를 소프트웨어를 통해 자동화 → 업무 효율 증가
- 값비싼 소프트웨어 없이 파이썬으로 업무 자동화 가능

● 파이썬으로 구현한 엑셀 파일 통합 자동화 사례

- 코드 12줄로 1~2초 이내 자동으로 업무 수행 구현 가능



```
files = listdir(r"d:\work")
path = (r"d:\work")
n=0
for myfile in files:
    n = n + 1
    if n == 1:
        te = pd.read_excel(path+"\"+myfile)
        result = te.drop(len(te.index)-1)
    t1 = pd.read_excel(path+"\"+myfile)
    test = t1.drop(len(t1.index)-1)
    result = result.append(test)
result.to_excel('봉사활동 시간 통합.xlsx')
```

◆ 데이터 분석

▪ NumPy (Numerical Python)

- 고성능 과학 계산을 패키지
- Matrix, Vector 와 같은 Array 연산을 할 때 표준 라이브러리처럼 사용
- 다차원 배열 생성이 가능하지만 주로 2차원 사용

▪ Pandas (Panel Data Analysis)

- numpy 기반으로 만든 라이브러리
- panel : 다차원 구조화된 데이터
- 엑셀 데이터, DBMS의 관계형 테이블 형태와 같은 데이터를 쉽게 다룰 수 있고, 분석을 용이하게 해주는 오픈 소스 데이터 분석 라이브러리

◆ 데이터 수집(크롤링)

▪ 분석 도구 : 크롬 개발자 도구(Chrome DevTools)

- 크롬에 기본 탑재된 웹 개발 및 디버깅 도구

▪ BeautifulSoup

- HTML 정보로부터 원하는 데이터를 가져오기 쉽게, 비슷한 분류의 데이터별로 parsing 하는 라이브러리

▪ Requests

- 특정 웹 사이트에 HTTP 요청을 보내 HTML 문서를 받아올 수 있는 라이브러리
- 단순한 String으로 받아오고, BeautifulSoup 을 사용해 HTML 문서로 바꾼다.

▪ Selenium

- 웹 애플리케이션을 위한 testing framework. 직접적으로 웹 사이트에 접근할 수 있게 해준다.
- 크롬 버전에 맞는 **Web Driver**를 설치해야 한다.

◆ 시각화 도구

▪ Matplotlib

- 데이터를 차트나 플롯(Plot)으로 그려주는 데이터 시각화(Data Visualization) 라이브러리
- numpy, Pandas 에서 사용되는 자료구조를 쉽게 시각화할 수 있음
- Line Plot, Bar Chart, Pie Chart, Histogram, Box Plot, Scatter Plot 등

◆ 엑셀 자동화

▪ OpenPyXL

- 엑셀을 다루기 위한 라이브러리
- 엑셀이 없어도 엑셀 파일을 읽고 쓸 수 있다.

▪ XlsxWriter

- 엑셀 파일을 만드는 기능만 있음

- ◆ 명령줄 인터페이스(CLI : Command Line Interface)
 - Terminal
 - **Command Prompt**(CMD:윈도우), Bash(리눅스), Terminal(MacOS)
 - **IDLE**
 - 파이썬 기본 탑재 콘솔. 순수 파이썬으로 작성
 - 파이썬 스크립트를 직접 편집할 수 있으며
 - 문법 강조와 디버깅 등을 지원
- ◆ 기타 코드 편집기
 - 메모장
 - Notepad++
 - Sublime Text
 - Atom

◆ 통합개발환경(IDE : Integrated Development Environment)

- Visual Studio Code : MS사
- PyCharm
 - JetBrains사. 각종 프레임워크 지원이 뛰어남
- Spyder
 - PyCharm 프로를 쓰기 힘든 개인 개발자가 사용할 만 함
- Anaconda
 - 데이터 분석, 기계학습 등에 필요한 1,400개 이상의 데이터 과학 패키지 포함

◆ 노트북 편집기

- 스타일 있는 문서 양식을 접목시킬 수 있다.
- 완성된 시스템을 만들기보다 데이터를 이리저리 돌려보며 코딩 → 결과 확인 → 코드 수정 → 결과 확인 등의 반복적인 작업을 할 수 있음
- **SymPy, NumPy, SciPy, Matplotlib** 같은 수학, 과학, 머신 러닝 라이브러리를 자주 이용하는 환경에서 개발할 때 유용한 편이며, **pandas** 등의 라이브러리를 사용하여 데이터 분석을 하는 데도 사용
- 100줄 이상의 진짜 개발은 파이썬 스크립트 파일을 작성해 하도록

• Jupyter Notebook (우리가 사용할 편집기)

- ipython : python 으로 build된 대화식 shell
- JupyterLab
- **Google Colaboratory**

파이썬 편집기 비교



◆ 사용 용도에 따라 다양한 편집기를 선택적으로 사용

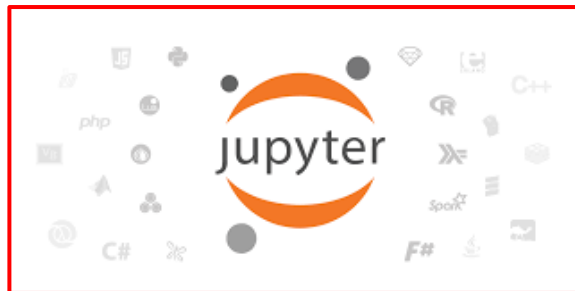
	프로그래밍 능력			개발 목적				운영체제		
	초급	중급	고급	웹개발	데이터 과학	스크립팅	QA	윈도우즈	Linux/Unix	Mac OS
IDLE 또는 온라인 편집기	•							•		
비주얼 스튜디오 코드		•	•	•				•		
파이참 프로페셔널		•	•	•	•			•	•	•
파이참 커뮤니티	•	•	•			•	•	•	•	•
스파이더		•	•		•			•	•	
주피터 노트북		•	•		•		•	•	•	

	프로그래밍 능력			개발 목적				운영체제		
	초급	중급	고급	웹개발	데이터 과학	스크립팅	QA	윈도우즈	Linux/Unix	Mac OS
구글 코랩		•	•		•			•	•	•
Sublime		•				•	•	•	•	•
Atom		•				•	•		•	•
Vim, Vi, nano			•			•			•	
Emacs			•			•			•	
이클립스, 울트라에디터		•	•			•		•	•	•

3. 파이썬 설치 및 프로그램 만들기

◆ 파이썬 설치 시 결정 사항

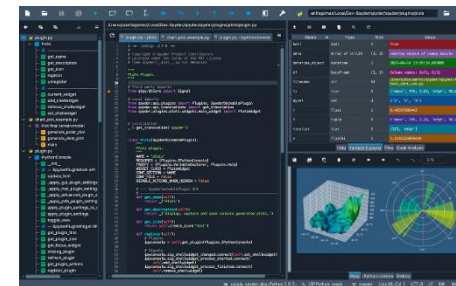
- 1) 사용할 운영체제
- 2) 파이썬 버전
- 3) 파이썬 편집기



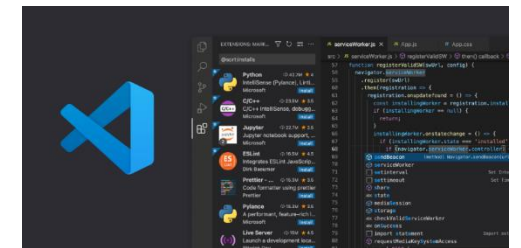
Jupyter Notebook



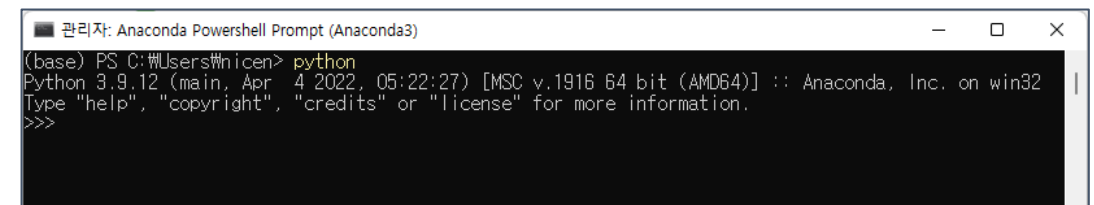
PyCharm



Spider



Visual Studio Code



Anaconda PowerShell

Jupyter Notebook



연세대학교
YONSEI MIRAE
CAMPUS

Jupyter Notebook 소개

- ◆ 데이터 분석을 절차대로 실행하면서 확인이 가능하므로, 특정 단계에서의 결과를 순차적으로 확인하면서 진행할 수 있음
- ◆ Jupyter Notebook은 Anaconda 로 파이썬을 설치하면 자동으로 설치됨
 - 그러나 실습을 위해서 **Jupyter Notebook을 install 하여 사용할 것임**
- ◆ Jupyter Notebook이 머신러닝에 자주 사용되는 이유는 시각화에 유용하고 다양한 프로그래밍 언어를 지원하기 때문
- ◆ 총 40개가 넘는 개발언어를 지원
- ◆ 기본적으로 에디터 프로그램이 아니라 **웹 브라우저에서 실행**되기 때문에 간결한 사용이 가능하고 원격 서버에도 호스팅이 가능하다는 장점이 있음

주피터 노트북 파일 생성

◆ hello.ipynb 생성

The process is shown in four sequential screenshots with numbered red annotations:

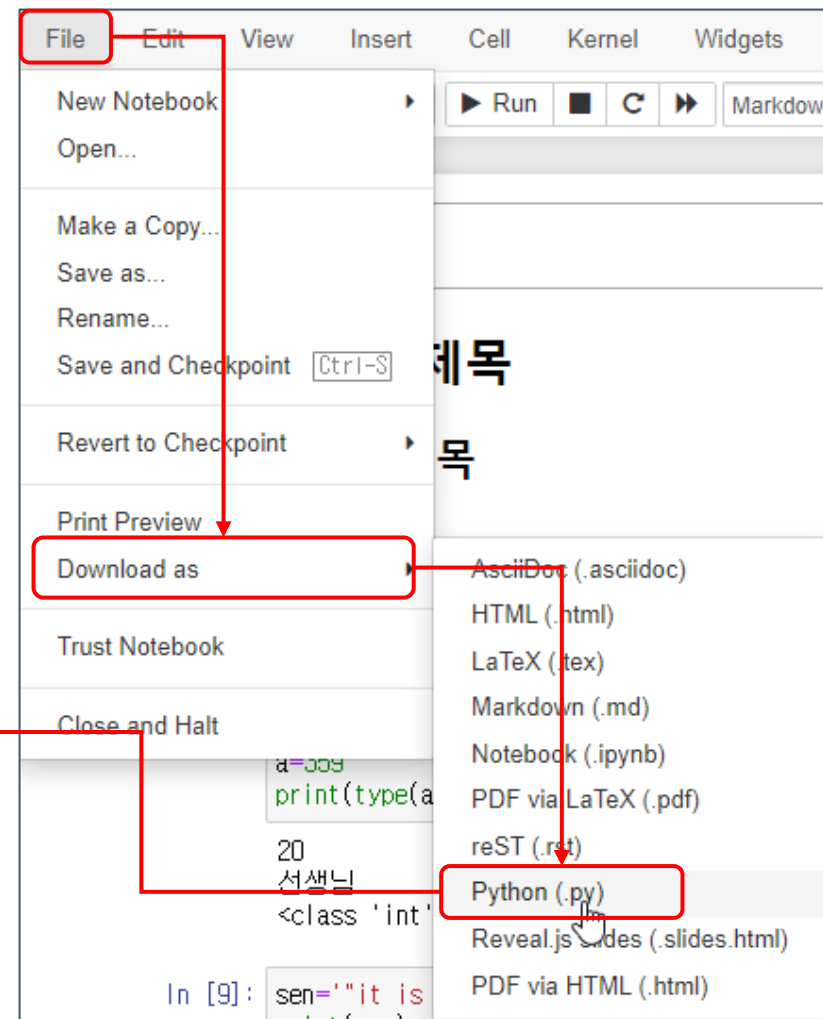
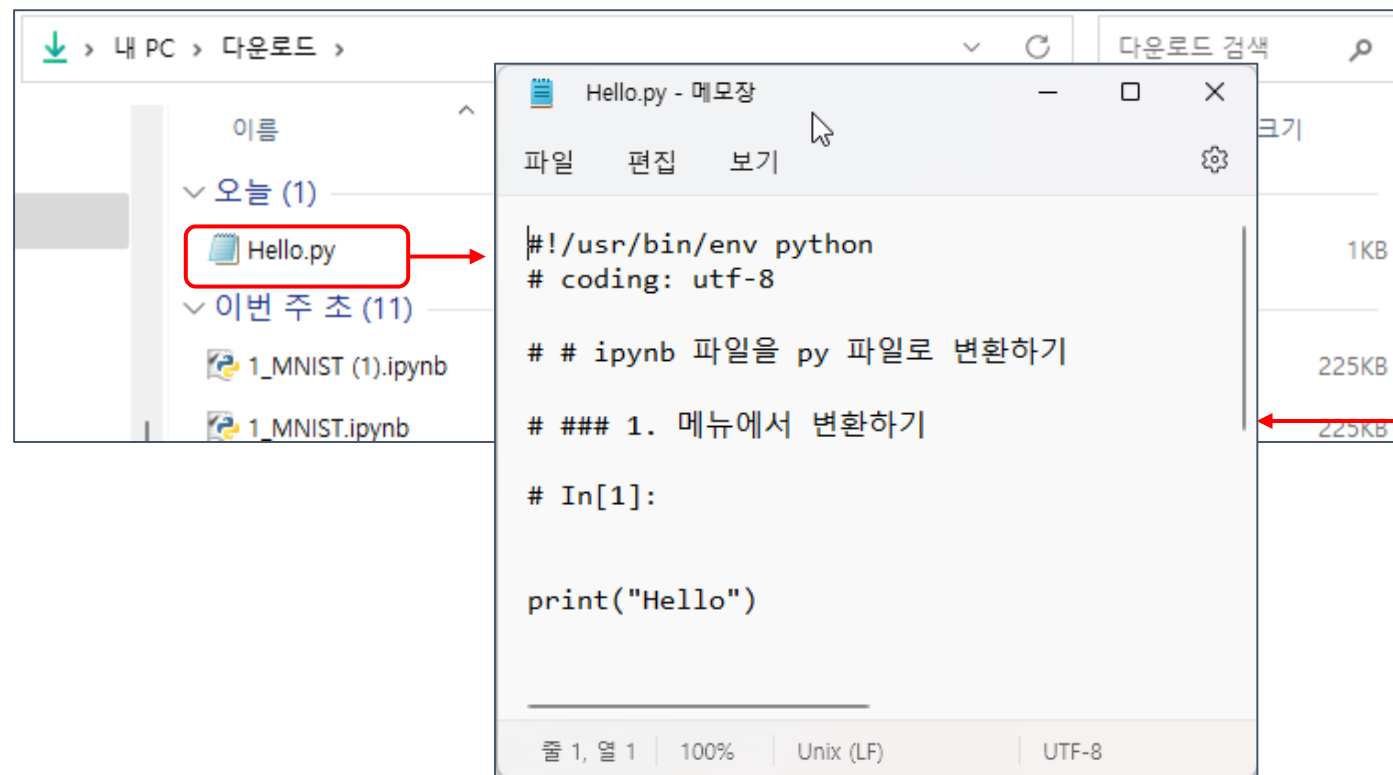
- Step 1:** The Jupyter Home Page is shown. The 'New' button in the top right is clicked, opening a dropdown menu.
- Step 2:** The dropdown menu is open, and 'Python 3 (ipykernel)' is selected. This action opens a new 'Untitled - Jupyter Notebook' window.
- Step 3:** In the new notebook window, the code `print("Hello")` is entered into a cell and executed. A 'Rename Notebook' dialog box appears, prompting for a new name.
- Step 4:** The 'Rename Notebook' dialog box is shown with 'Hello' entered in the text field. The 'Rename' button is clicked. This action saves the notebook as 'Hello.ipynb' in the file browser on the left side of the Jupyter interface.

ipynb 파일을 py 파일로 변환하기(1)

◆ Jupyter Notebook 화면에서 변환하기

◆ [File]-[Download as]-[Python (.py)]

- 변경하기 전, 중간 중간 살펴보기 위해 넣었던 코드를 정리하는 것이 좋다.
- 윈도우에서 지정된 [download] 폴더에 저장된다.



ipynb 파일을 py 파일로 변환하기(1)

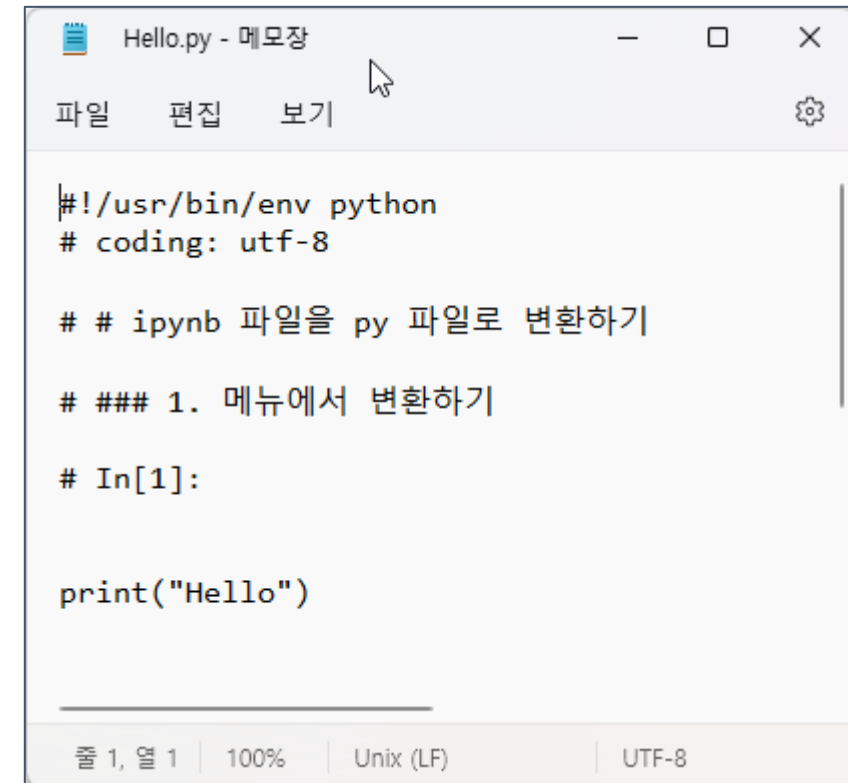
◆ 변환 전 Hello.ipynb

ipynb 파일을 py 파일로 변환하기

1. 메뉴에서 변환하기

```
In [1]: 1 print("Hello")  
Hello
```

◆ 변환 후 Hello.py



```
#!/usr/bin/env python  
# coding: utf-8  
  
# # ipynb 파일을 py 파일로 변환하기  
  
# ### 1. 메뉴에서 변환하기  
  
# In[1]:  
  
print("Hello")
```

줄 1, 열 1 | 100% | Unix (LF) | UTF-8

ipynb 파일을 py 파일로 변환하기(2)

◆ Jupyter Notebook 셀에서 변환하기

%%writefile HelloPython.py

- 그 아래의 모든 내용이 .py 파일로 현재 폴더에 저장

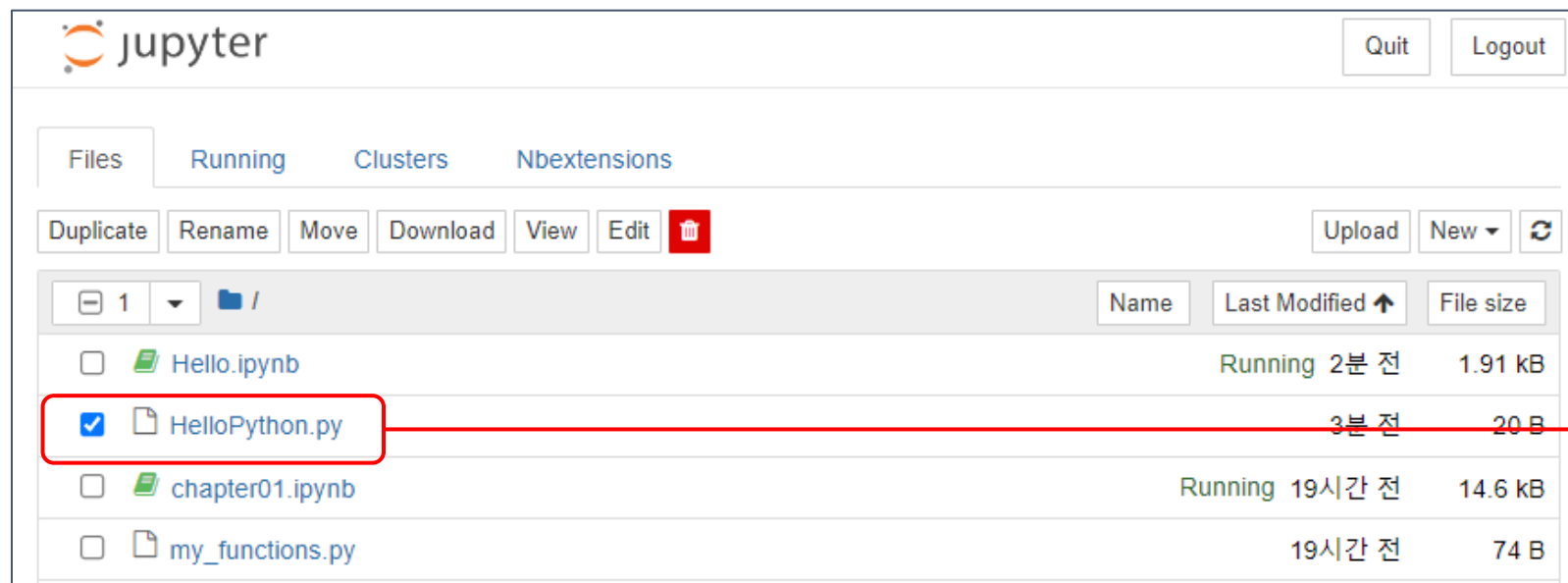
```
In [1]: 1 %%writefile HelloPython.py
        2 a="Hello! Python!"

Writing HelloPython.py
```

```
jupyter HelloPython.py ✓ 2분 전

File Edit View Language

1 a="Hello! Python!"
2
```



The Jupyter Notebook interface shows the file browser. The 'Files' tab is active. A table lists the files in the current directory:

	Name	Last Modified	File size
<input type="checkbox"/>	Hello.ipynb	Running 2분 전	1.91 kB
<input checked="" type="checkbox"/>	HelloPython.py	3분 전	20 B
<input type="checkbox"/>	chapter01.ipynb	Running 19시간 전	14.6 kB
<input type="checkbox"/>	my_functions.py	19시간 전	74 B

A red box highlights the 'HelloPython.py' file, and a red arrow points from it to the code cell above.

ipynb 파일을 py 파일로 변환하기(3)

◆ cmd 창에서 실행하기

- nbconvert 설치하기

- > `pip install nbconvert`

- 주피터 노트북 to 파이썬 변환(ipynb to py)

- > `jupyter nbconvert --to script Hello.ipynb`

- 여러 파일 변환하기

- > `jupyter nbconvert *.ipynb --to script`

파이썬 살펴보기



연세대학교
YONSEI MIRAE
CAMPUS

PEP 8 : 뚜렷한 권장 코드 스타일

- ◆ 파이썬에 맞는 코딩 스타일을 지키기를 권장하는 파이썬 개선 제안(PEP :Python Enhancement Proposals) 문서
- ◆ 블록 처리 규칙
 - 중괄호 대신 들여쓰기, 탭 문자 대신 공백(space) 4문자 강력히 권장
 - 파이썬 인터프리터가 탭문자 하나를 공백 1문자로 처리하기 때문
- ◆ 작명 규칙
 - 일반적으로 **스네이크** 표기를 쓰되, 특정한 종류에는 **파스칼** 표기를 쓴다.
 - 변수는 소문자로 시작.
 - 내부변수는 **맨 앞에 밑줄(underscore _)** 1개로 시작.
 - `_attribute_name = 0`
 - 숨은 변수(hidden)는 **밑줄 2개**로 시작
 - `__hidden_attribute_name = 0`

PEP 8 : 뚜렷한 권장 코드 스타일

◆ 작명 규칙

종류	규칙	예시
패키지(package)	스네이크()	
모듈(module)	스네이크()	import module_name
클래스(class)	파스칼()	class ClassName()
예외(exception)	파스칼	
함수(function)	스네이크()	def function_name()
상수(constant)	대문자+ 밑줄	MODULE_CONSTANT_NAME = 0
변수(variable)	스네이크	variable_name = 0
매개변수(parameter)		
지역변수(local variable)		
인스턴스 변수(instance variable)		
메서드(method)	스네이크()	method_name()

- ◆ Camel case(낙타 표기법) : Java 진영
 - **getMethodScore()** : 소문자로 시작. 2번째 단어부터는 첫글자가 대문자로 시작
- ◆ Pascal case(쌍봉 낙타 표기법) : Java, **Python** 진영
 - 처음 단어도 **대문자**로 시작
 - **CommonUtil { }**
- ◆ Hungarian Notation(헝가리안 표기법) : C 진영
 - 변수명의 앞에 **자료형**을 붙이는 방식
 - `int iMath = 50;`
- ◆ Snake case(스네이크 표기법) : C, C++, **Python**, R, Java 등
 - 뱀 모습을 따서 단어는 모두 소문자로 쓰되, 단어 간의 구분은 **밑줄**로 대체하는 방식
 - `math_score = 50`
- ◆ Kebab case(케밥 표기법) : 꼬치 표기법
 - 언더바 대신 **하이픈(-)**을 사용
 - 주로 프로퍼티(properties) 값 설정, CSS, HTML 에서 일부 사용

PEP 8 : 뚜렷한 권장 코드 스타일

◆ 문법 규칙

- 한 줄은 **79**자로 제한하기
- **import** 는 파일의 맨 위에 적고 → 내장 모듈 → 제 3자 모듈 → 직접 만든 모듈 순서로 불러들이기
- 인스턴스 메서드의 첫 인자는 **self** 로 쓰고, 클래스의 메서드의 첫 인자는 **cls** 로 쓰기
- 할당 연산자(=)의 앞 뒤로 **공백** 넣기

◆ PEP8을 지키는지 확인해주는 패키지들

- **Pylint, Flake8, Black**

파이썬 명명규칙에서 언더바의 역할

1. 인터프리터에서 사용
2. 무시하는 값
3. 루프에서 사용
4. 숫자값의 분리
5. 명명용
 - ① 앞에 하나가 쓰이는 경우
 - ② 뒤에 하나가 쓰이는 경우
 - ③ 앞에 두개가 쓰이는 경우
 - ④ 앞 뒤로 두개씩 쓰이는 경우

1. 인터프리터에서의 사용

- 가장 마지막 표현식의 결과값을 자동적으로 “_” 라는 변수에 저장

In [1]:	1	5+4
Out [1]:	9	
In [2]:	1	_
Out [2]:	9	
In [3]:	1	_ _+6
Out [3]:	15	
In [6]:	1	_
Out [6]:	15	
In [7]:	1	a = _
	2	a
Out [7]:	15	

2. 무시하는 값

- 해당 값을 unpack 하기 싫다면, 그냥 “_” 에다 할당
- packing : 여러 개의 객체를 하나의 객체로 합쳐서 받음
- unpacking : 여러 개의 객체를 포함하고 있는 하나의 객체를 풀어줌

```
1 # 무시하는 값을 버린다
2 a, _, b = (1, 2, 3)    # a=1, b=3, _ 에 2가 할당
3 print(a, b)
```

1 3

```
1 # packing : * 한개를 매개변수 앞에 붙임으로, 위치인자로 보내 모든 객체들을 하나의 객체로 관리
2 def func(*args):
3     print(args)
4     print(type(args))
5
6 func(1, 2, 3, 4, 5, 6, 'a', 'b')
```

(1, 2, 3, 4, 5, 6, 'a', 'b')
<class 'tuple'>

```
1 # unpacking : 여러개 값 버리기
2 # *(변수) 는 unpack할때, 여러개의 값을 하나의 변수에 저장할때 쓰인다.
3 # 이는 확장된 Unpacking이라고 불리며, Python 3.x 버전에서만 가능하다.
4
5 a, *_ , b = (7, 6, 5, 4, 3, 2, 1)
6 print(a, b)
```

7 1

3. 루프에서 사용

```
1 # '_'를 이용해서 루프를 돈다.  
2 for _ in range(5):  
3     print(_)  
4  
5 # 리스트 순회를 _ 를 이용해서 한다.  
6 # _ 를 일반 변수처럼 사용할 수 있다.  
7 languages = ["Python", "JS", "PHP", "Java"]  
8 for _ in languages:  
9     print(_)  
10  
11 _ = 5  
12 while _ < 10:  
13     print(_, end = ' ')    # 'end'의 기본값은 '\n'인데 공백으로 변경해준다.  
14     _ += 1
```

```
0  
1  
2  
3  
4  
Python  
JS  
PHP  
Java  
5 6 7 8 9
```

4. 숫자값의 분리

- 숫자값이 길다면 **자릿수 구분**을 위해 _ 를 중간에 넣어줄 수 있다.

```
1 # 여러 숫자 표현법  
2 # 아래 값들이 정확한지 확인하기 위해 int 함수를 써 볼수도 있다.  
3  
4 million = 1_000_000  
5 binary = 0b_0010  
6 octa = 0o_64  
7 hexa = 0x_23_ab  
8  
9 print(million)  
10 print(binary)  
11 print(octa)  
12 print(hexa)
```

```
1000000  
2  
52  
9131
```

5. 명명용

- 변수, 함수, 클래스 명 등에 언더바가 사용될 수 있음

① 앞에 하나가 쓰이는 경우

- 내부 사용용

```
1 # 5.1 앞의 하나의 언더바 : 내부 사용용
2 class Test:
3
4     def __init__(self):
5         self.name = "datacamp"
6         self._num = 7
7
8 obj = Test()
9 print(obj.name)
10 print(obj._num)
```

datacamp
7

- 변수명 앞에 _ 를 하나 붙였다고 해서, 해당 변수를 접근하지 못하게 되진 않는다. 하지만 모듈을 import 해서 쓰는 경우 예러가 발생한다.
- 언더바 하나로 시작한 이름들은 import 하지 않는다.

my_functions.py

```
1 %%writefile my_functions.py
2 def func():
3     return "datacamp"
4
5 def _private_func():
6     return 7
```

```
1 # 언더바 하나로 시작한 이름들을 import 하지 않는다.
2 from my_functions import *
3 #import my_functions
4 func()
```

'datacamp'

```
1 _private_func()
```

Traceback (most recent call last)
Input In [24], in <cell line: 1>()
----> 1 _private_func()

NameError: name '_private_func' is not defined

```
1 my_functions._private_func()
```

7

5. 명명용

② 뒤에 하나가 쓰이는 경우

- 파이썬 키워드를 변수명, 함수명, 클래스명으로 쓰고 싶을 때.

```
1 # 5.2 뒤의 하나의 언더바 : 기본 키워드와의 충돌을 피한다
2 #def function(class):           # invalid syntax 에러 발생
3 def function(class_):
4     ...
5     pass
6     ...
```

③ 앞에 두개가 쓰이는 경우

- name mangling(짓이기다) 방법. 인터프리터에게 해당 서브클래스의 attribute 이름을 바꾸어서 이름 충돌이 나지 않게 하라고 말하는 것.

```
1 # 5.3 앞의 2개의 언더바 name mangling.
2 class Sample():
3     def __init__(self):
4         self.a = 1
5         self._b = 2
6         self.__c = 3
7
8 obj1 = Sample()
9 dir(obj1)
```

```
['_Sample__c',
'__class__',
'__delattr__',
'__dict__',
'__dir__',
'__doc__',
'__eq__',
'__format__',
'__ge__',
'__getattr__',
'__gt__',
'__hash__',
'__init__',
'__init_subclass__',
'__le__',
'__lt__',
'__module__',
'__ne__',
'__new__',
'__reduce__',
'__reduce_ex__',
'__repr__',
'__setattr__',
'__sizeof__',
'__str__',
'__subclasshook__',
'__weakref__',
'_b',
'a']
```

5. 명명용

④ 앞 뒤로 두개씩 쓰이는 경우

- magic method 혹은 dunder(double underscores) 메소드라 불린다.
 - ✓ magic method : `__init__`, `__add__`, `__len__`, `__repr__` 등
 - ✓ `__init__` : 생성자. 호출 없이 초기화할 때 호출되는 함수. 클래스의 인스턴스가 생성될 때 실행
 - ✓ `__repr__` : 해당 클래스를 출력할 때 내부적으로 호출되는 함수.
- 변수 이름으로 사용할 경우 변수명 충돌이 일어날 수 있기 때문에 가급적 삼가하는 것이 좋다.
- dunder는 보통 연산자 오버로딩을 할 때 많이 사용한다.

```
1 # 5.4 앞뒤로 2개의 언더바 : magic method 혹은 dunder(double underscores)
2 # magic method : __init__, __add__, __len__, __repr__
3
4 class Sample():
5
6     def __init__(self):
7         self.__num__ = 7
8
9 obj = Sample()
10 obj.__num__
```

Out [7]: 7