# 파이썬 실행 환경 만들기

## 아나콘다 설치하기



그림 아나콘다 파이썬 로고

우리가 컴퓨터를 새로 구입하거나, 컴퓨터를 초기화를 하게 되면, 가장 먼저 하게 되는 일은 필수 프로그램을 설치하는 일일 것입니다. 필자의 경우도 카카오톡이나 구글 크롬을 가장 먼저 설치합니다. 아무튼 파이썬도 마찬가지입니다. 파이썬을 설치하면 순수한 파이썬으로는 할 수 있는 것이 크게 많지 않습니다. 예를 들어 데이터분석을 하려면 파이썬에 데이터분석을 할 수 있는 도구를 설치해주어야 합니다. 게임을 만들려면 게임을 만들 수 있는 도구를 파이썬에 설치해주어야 합니다. 그 밖에 필수적으로 설치해야 하는 도구들도 많이 있습니다. 파이썬을 설치하고 이러한 필수 도구들을 파이썬에 설치하는 일은 꽤나 번거로운 작업입니다. 하지만 이러한 번거로운 작업을 쉽게 처리해주는 것이 있는데, 아나콘다라는 패키지 프로그램을 설치하는 것입니다. 아나콘다라는 것은 파이썬에 필수 도구들이 모두 설치가 되어있는 완성형 파이썬을 아나콘다라고 부릅니다. 아나콘다를 설치하면 따로 다양한 필수 도구들을 파이썬에 설치하지 않아도 되기 때문에 바로 다양한 프로젝트 프로그래밍 작업을 할 수 있습니다. 아나콘다 파이썬을 설치하는 방법은 아래와 같습니다.

1. 가장 먼저 아나콘다 공식 사이트(https://www.anaconda.com/products/distribution)에 접속합니다. 아나콘다 파이썬 공식 사이트에 접속하신 후, 아래 그림에 빨갛게 네모로 표시된 곳을 클릭해주세요.

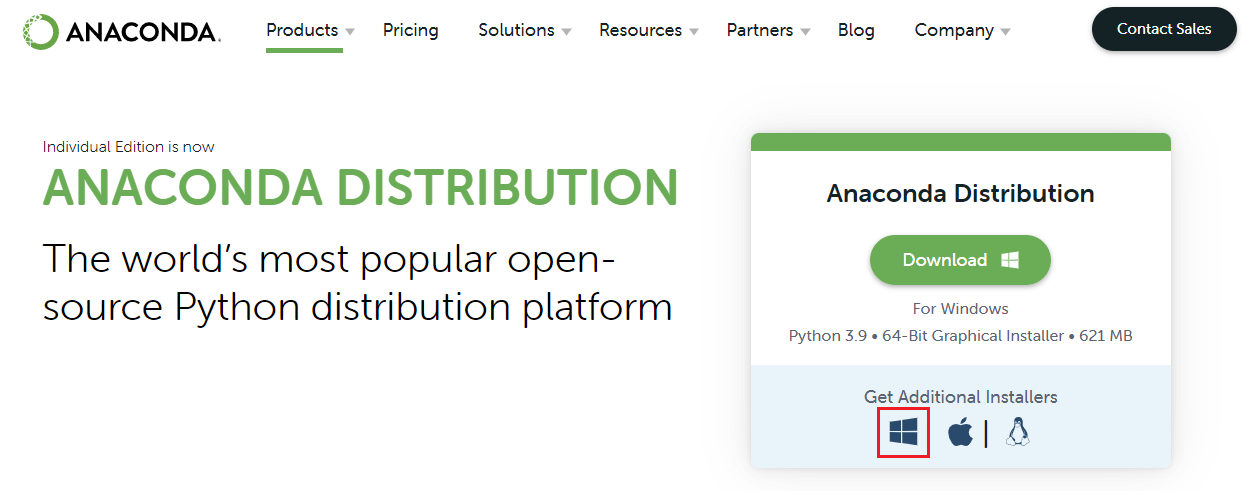


그림 아나콘다 파이썬 공식 사이트

1. 빨갛게 네모로 표시된 곳을 클릭하면 아나콘다 파이썬 페이지 가장 아래에 있는 곳으로 페이지가 이동됩니다. 그 곳에서 운영체제 OS에 맞는 것을 선택해서 아나콘다 파이썬 설치 실행파일을 다운받고, 설치하면 됩니다. (필자는 윈도우 기준으로 작성하였습니다.) 또한 필자와 다운받는 아나콘다 파이썬 버전이 다를 수 있습니다. 하지만 크게 버전이 차이 나지 않으면 상관없습니다. 최신버전으로 다운받아 주시면 됩니다.



그림 OS에 맞게 아나콘다 설치하기

1. 다운받은 파일을 실행하면 아래와 같은 화면이 나올 것입니다. 아래와 같은 화면이 나오면 **“Next”** 버튼을 눌러주세요.

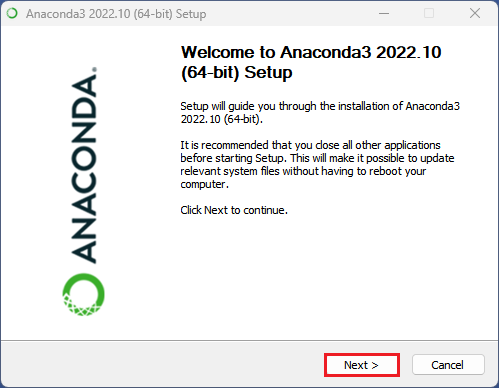


그림 아나콘다 설치 화면1

1. 다음으로는 아나콘다 파이썬 설치 약관 동의화면이 나옵니다. **“I Agree”** 버튼을 눌러 동의를 해주세요.

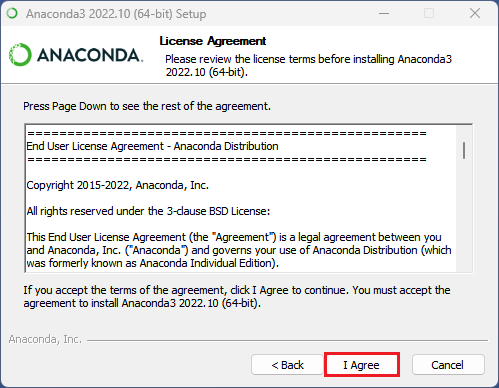


그림 아나콘다 설치 화면2

1. 다음은 설치 타입을 묻는 화면이 나옵니다. **“Just Me”**를 선택하고 **“Next”** 버튼을 눌러주세요. 다만 윈도우 계정명이 한글이라면 제대로 설치되지 않을 수 있습니다. 만약 윈도우 계정명이 한글이라면 아래에 **“All Users”**을 선택하고 **“Next”** 버튼을 눌러주세요.

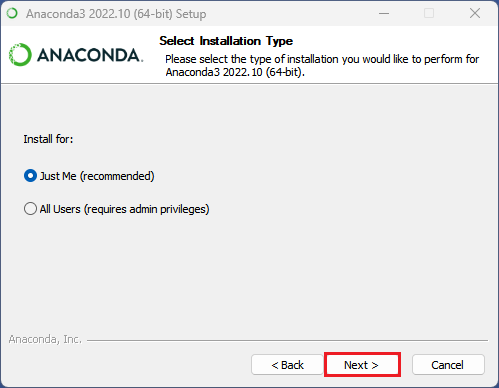


그림 아나콘다 설치 화면3

1. 다음은 아나콘다 파이썬 설치 경로를 묻는 화면이 나옵니다. 설치 경로는 기본으로 두시고, **“Next”** 버튼을 눌러주세요. 하지만 여러 번 아나콘다 파이썬을 중복해서 설치했거나, 윈도우 계정명이 한글인 경우 설치가 원활하게 되지 않으실 수 있습니다. 만약 그런 상황이라면 경로를 원하시는 곳으로 변경해서 설치를 진행해보세요. 다른 프로그래밍 언어도 마찬가지이지만, 윈도우 계정명이 한글이거나, 경로에 한글이 포함되어 있으면 에러가 발생하는 경우가 많습니다. 만약 프로그래밍을 자주 사용해야 하는 사람이라면 윈도우를 초기화하고 윈도우 계정명을 입력하실 때 꼭 영어로 된 이름을 사용해주세요. (꼭 윈도우를 초기화할 필요는 없으나, 일반적인 경우로 계정명을 변경하면 아무런 효과가 없기 때문에 초보자는 초기화하는 편을 더 추천합니다.)

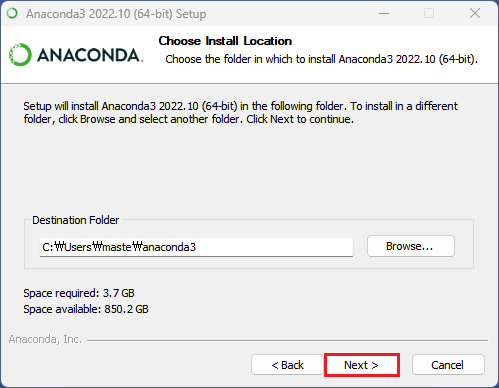


그림 아나콘다 설치 화면4

1. 다음은 아나콘다 설치 옵션을 묻는 화면입니다. 화면에서 2개의 체크박스가 있는데 모두 선택하고 **“Next”** 버튼을 눌러주세요. 위에 있는 체크박스를 클릭하면 추천하지 않는다는 경고가 나옵니다. 위에 있는 체크박스 옵션은 아나콘다 파이썬을 기본 경로로 설정하겠냐는 내용입니다. 파이썬을 전문적으로 다루는 사람은 파이썬을 여러 버전을 가지고 있기 때문에 기본 경로를 특정 파이썬에 두고 사용하지 않습니다. 하지만 아나콘다 파이썬 하나만 있어도 웬만한 것은 모두 개발할 수 있습니다. 따라서 아나콘다 파이썬을 기본 경로로 두고 사용해도 되기 때문에 경고는 무시하시고 체크박스를 모두 선택하시고 **“Install”** 버튼을 눌러주세요.

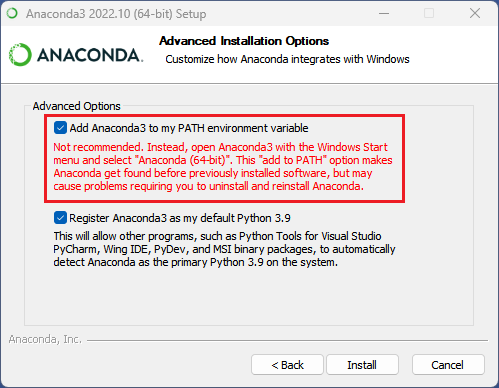


그림 아나콘다 설치 화면5

1. 여기까지 진행했다면 본격적으로 아나콘다 파이썬을 설치하게 됩니다. 설치가 잘 진행되다가 마지막에 멈추는 것처럼 보일 수 있으나 정상이므로 안심하고 기다려 주시면 됩니다. (노트북에서 진행하면 더 느릴 수 있습니다. 보통 10분에서 최대 30분까지 걸릴 수 있습니다.)

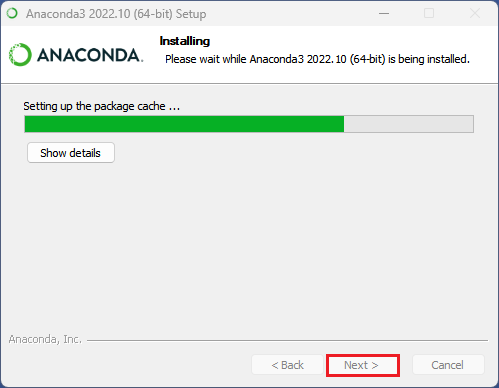


그림 아나콘다 설치 화면6

1. 설치가 끝나면 다음과 같은 화면이 나옵니다. 이 화면이 나오면, **“Next”** 버튼을 눌러서 다음 화면으로 넘어가줍니다.

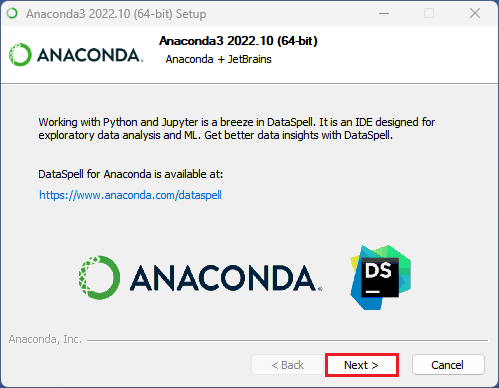


그림 아나콘다 설치 화면7

1. 다음 화면이 나오면 아나콘다 파이썬이 잘 설치되었다는 의미입니다. 2개의 체크박스가 있다면 튜토리얼을 보거나, 아나콘다 파이썬을 바로 실행하겠냐는 옵션인데 모두 필요 없으므로 모두 체크 해제하시고 **“Finish”** 버튼을 눌러주세요.

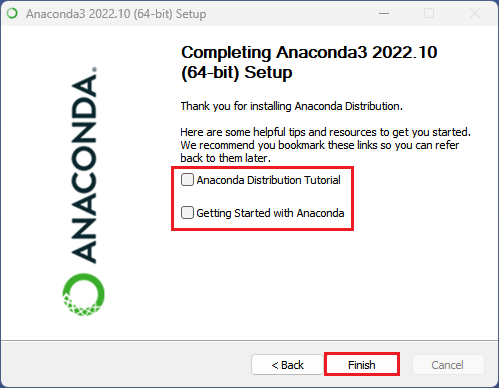


그림 아나콘다 설치 화면8

여기까지 진행했다면 아나콘다 파이썬을 잘 설치하신 것입니다. 이제 아나콘다 파이썬을 사용해서 프로그래밍을 할 수 있는 환경을 만들어봅시다.

## Jupyter Notebook 사용하기

Jupyter Notebook은 파이썬 프로그래밍을 할 수 있는 편집기입니다. 그림을 그리기 위해서는 도화지가 필요하듯 프로그래밍을 하려면 편집기가 필요합니다. 저희는 앞으로 Jupyter Notebook을 사용해서 파이썬 프로그래밍을 진행하려고 합니다. 파이썬 프로그래밍을 할 수 있는 편집기는 아주 많습니다. Jupyter Notebook은 개발 환경이 아주 심플하고 가시적이어서 파이썬 프로그래밍하기 편리합니다. Jupyter Notebook을 실행하는 방법은 아래와 같습니다.

1. 윈도우 검색창에 **“Jupyter Notebook”**을 입력하시면, 노란색 아이콘의 Jupyter Notebook 아이콘이 나옵니다. 아이콘이 나오면 클릭해서 실행해주세요.

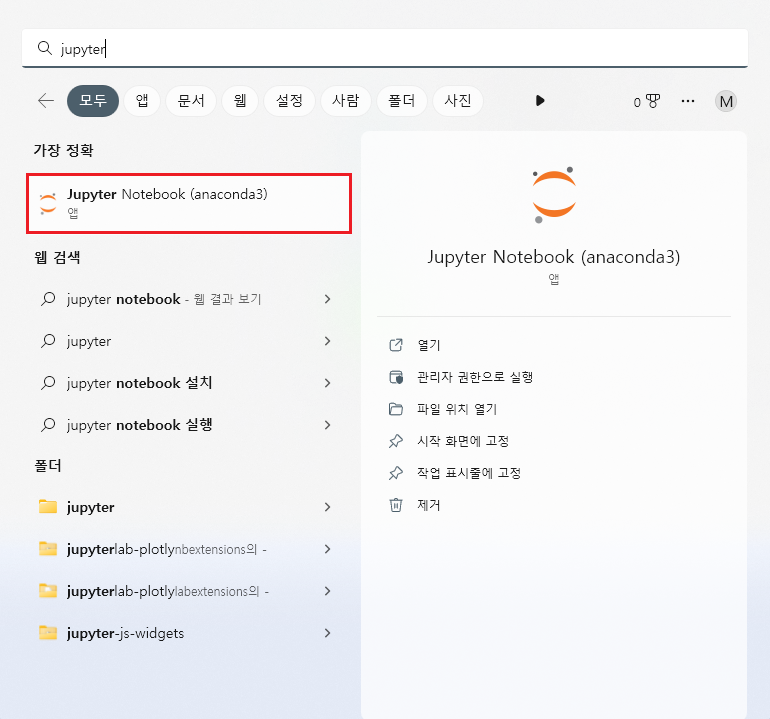


그림 Jupyter Notebook 실행하기1

1. 해당 아이콘을 클릭하면 어떤 검은 창이 하나 생성됩니다. 이 검은 창은 Jupyter Notebook을 실행할 때 사용되는 프로그램 같은 것이라고 생각하면 됩니다. 따라서 Jupyter Notebook을 사용하고 있을 때는 이 검은 창을 절대 끄면 안 됩니다.

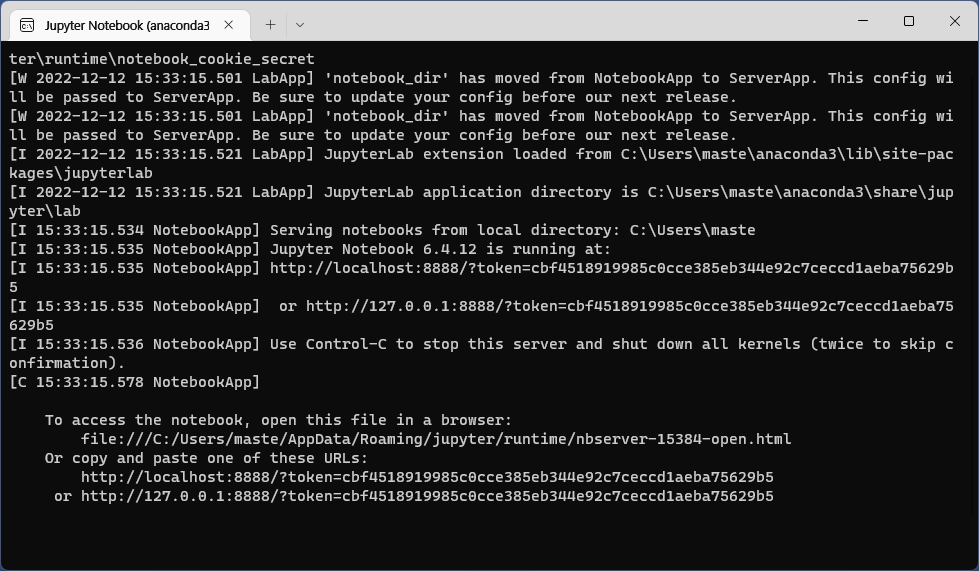


그림 Jupyter Notebook 실행하기2

1. 검은 창이 뜨고 잠시 동안 기다리면, 아래 그림처럼 Jupyter Python 크롬이나 Edge 등 사용하시는 웹에서 창이 하나 뜰 것입니다. 여기서 우리는 파이썬 프로그래밍을 진행할 수 있습니다. 웹에서 실행되었다고 해서 인터넷 상에서 프로그래밍을 진행하는 것은 아닙니다. 그저 창만 빌려왔다고 생각하시면 됩니다. 따라서 온라인이 아닌 오프라인 환경에서도 Jupyter Python을 실행하실 수 있습니다.



그림 Jupyter Notebook 화면

1. 여기까지 진행이 되었다면, 바탕화면에 “python” 폴더를 하나 만들겠습니다. 우리가 앞으로 파이썬 기초, 데이터 분석, 데이터 크롤링, 머신러닝을 진행하면서 원활한 진행을 위해 필요한 파일은 바탕화면에 만든 “python” 폴더에 저장해 놓고 진행하려고 합니다. **폴더는 한글보다는 영어로 지정해 놓는 것이 좋습니다.**



그림 바탕화면에 python 폴더를 만들기

1. 바탕화면에 “python” 폴더를 만들었다면, Jupyter Notebook을 통해 바탕화면에 접근하고 우리가 만든 “python” 폴더에 접근해야 합니다. Jupyter Notebook을 통해 “python” 폴더에 접근하려면 Jupyter Notebook에 보이는 화면에서 **“Desktop”** 또는 **“OneDrive”** 폴더를 클릭해줍니다. (OneDrive에 윈도우가 설치되었다면 OneDrive 폴더 안에 바탕화면 폴더가 있습니다.) 필자는 OneDrive에 윈도우가 설치되었기 때문에 **“OneDrive”** 폴더를 클릭하겠습니다.



그림 Jupyter Notebook에 OneDrive 폴더를 클릭하기

1. OneDrive 폴더에 들어가면, 바탕 화면 폴더가 보일 것입니다. 바탕 화면 폴더에 보인다면, **바탕 화면** 폴더를 클릭해줍니다.

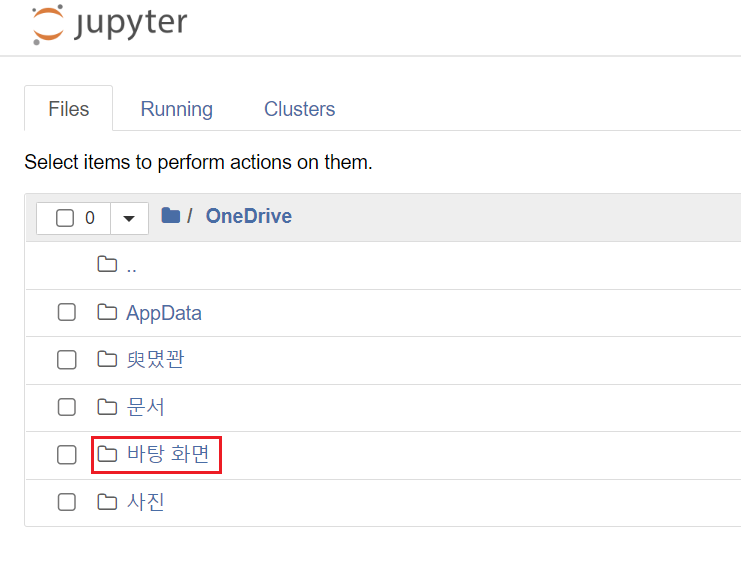


그림 OneDrive 폴더 안에 바탕 화면 폴더 클릭하기

1. 바탕 화면 폴더를 클릭하면, 우리가 바탕화면에 만든 python 폴더가 보일 것입니다. python 폴더가 보인다면, 클릭해줍니다.

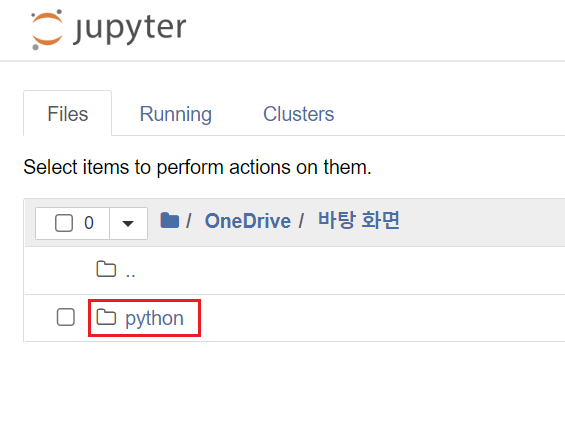


그림 바탕화면에 만든 python 폴더 클릭하기

1. python 폴더에 들어가면 아래와 같은 화면이 보일 것입니다. 우리가 만들고 아직 아무것도 넣지 않았기 때문에 폴더 안은 비어 있을 것입니다. 오른쪽 상단에 **“New”** 버튼을 눌러줍니다.

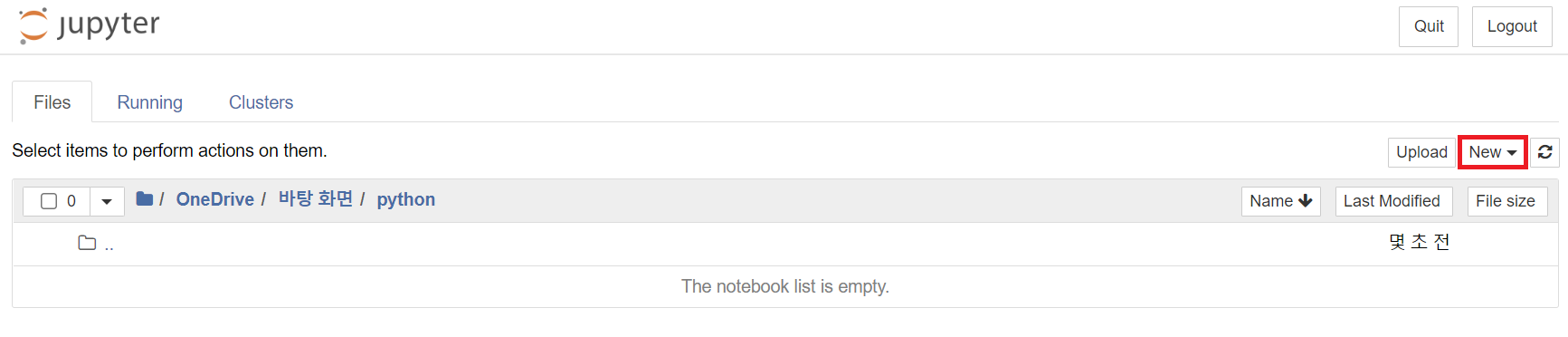


그림 python 폴더에서 New 버튼 누르기

1. New 버튼을 누르면, **Python3 (ipykernel)** 이라는 버튼이 있을 것입니다. 이 버튼은 파이썬을 프로그래밍 할 수 있는 파일을 생성하겠다는 의미이므로, 클릭해줍니다.

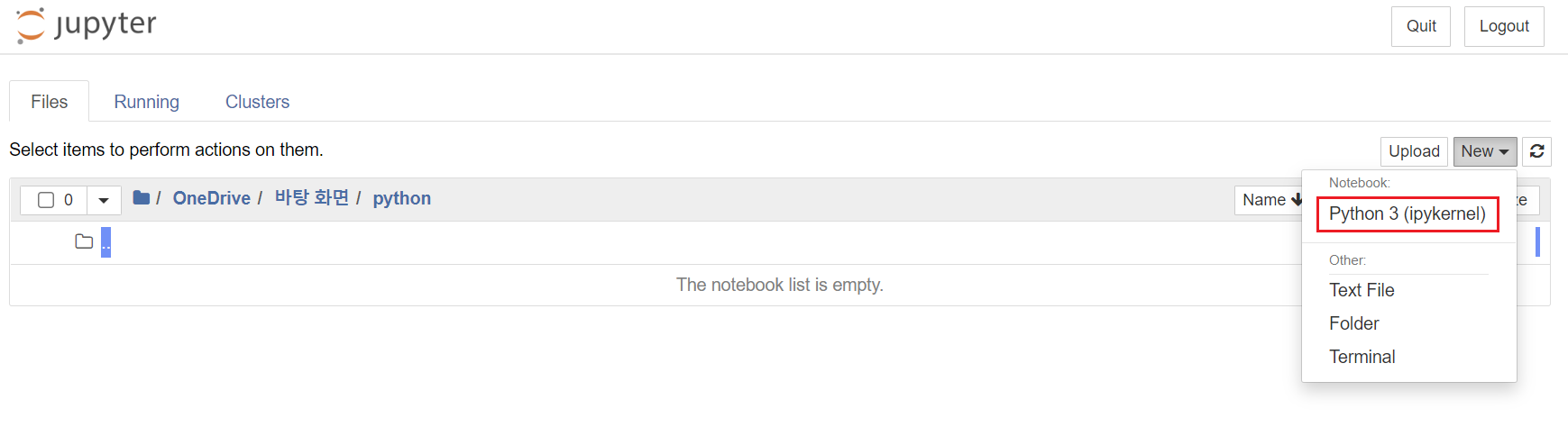


그림 Python3 (ipykernel) 클릭하기

1. 여기까지 진행하셨다면, 아래와 같은 화면이 보일 것입니다. 이 화면은 앞으로 우리가 파이썬 프로그래밍을 진행할 때, 항상 사용하게 될 코드 창입니다. 왼쪽 상단에 “Untitled” 이라고 적힌 것이 제목입니다. 제목 부분을 클릭하면 제목을 수정하실 수 있습니다.

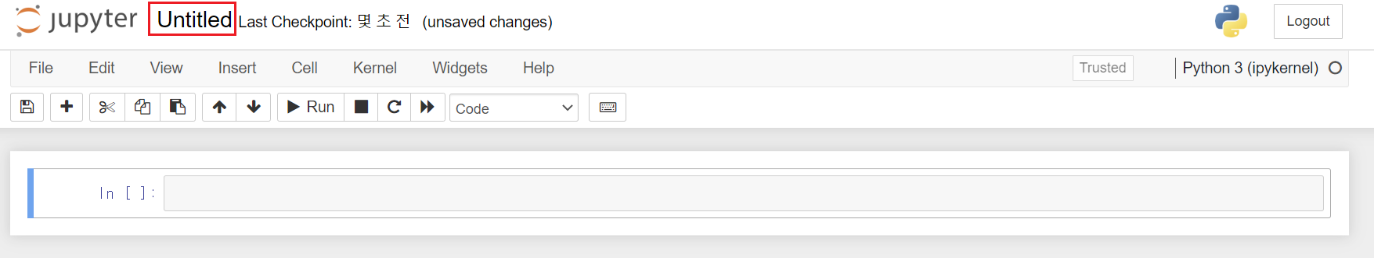


그림 프로그래밍을 할 수 있는 코드 창

1. 나중에 복습하거나 다시 코드를 불러와서 사용해야하기 때문에 제목을 우리가 알아보기 편한 제목으로 수정하는 것이 좋습니다. 제목을 이해하기 쉬운 제목으로 수정하신 뒤, **“Rename”** 버튼을 누르시면 제목이 수정됩니다.

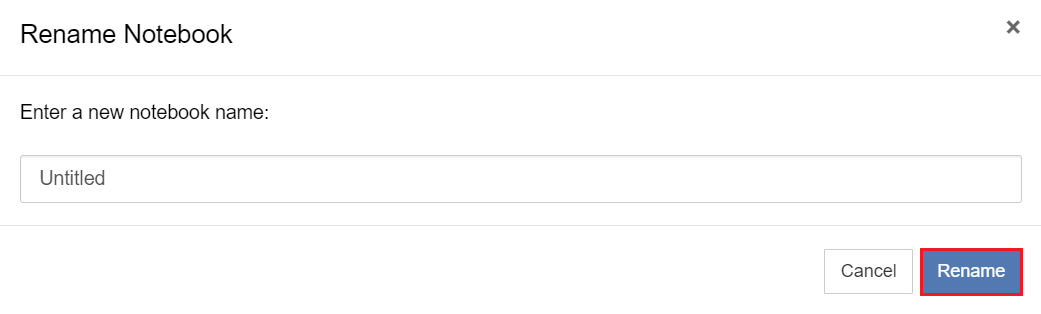


그림 Jupyter Notebook 제목 수정하기

1. 이제 본격적으로 파이썬 프로그래밍을 할 수 있게 되었습니다. 코드 창에 코드를 작성한 후, Shift + Enter 키를 누르면 코드를 실행할 수 있습니다. 간단하게 코드를 작성해서 실행해보겠습니다. 코드 창에 아래 화면처럼 print(“Hello World!”) 를 작성하고 Shift + Enter 키를 눌러보세요. 여기서 print() 함수는 파이썬에서 제공하는 기본 함수입니다. 함수에 대해서는 뒤에서 좀 더 자세히 다루도록 하겠습니다. print() 함수는 입력 값을 화면에 출력해주는 함수입니다. 아마 파이썬에서 가장 많이 사용되는 함수가 아닐까 생각됩니다. print() 함수 안에 “Hello World!” 를 입력해주었기 때문에 화면에 Hello World! 가 출력되는 것입니다. 참고로 어떤 문자를 표현하려면 앞뒤에 작은 따옴표 또는 큰 따옴표를 반드시 입력해주어야 합니다.

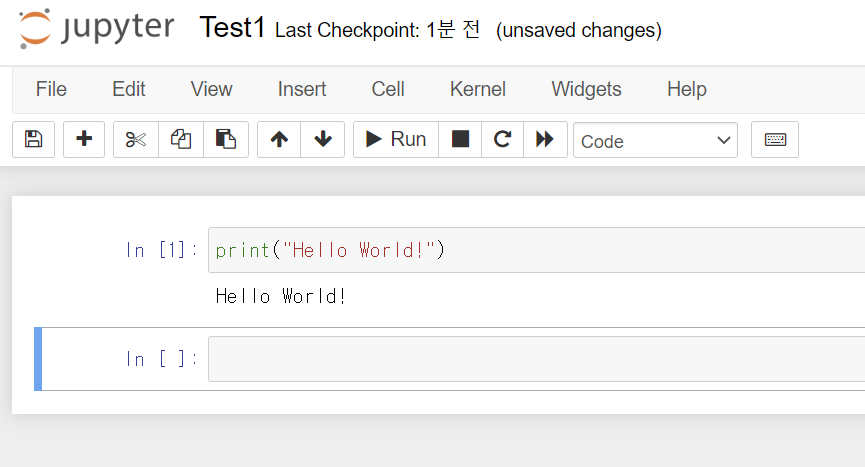


그림 Jupyter Notebook에서 Hello World! 출력하기

## Google Colab 사용하기

Google Colab은 웹 사이트에서 파이썬 프로그래밍을 할 수 있는 플랫폼입니다. 성능도 상당히 강력하고 사용하는 방법도 Jupyter Notebook과 거의 같아서 머신러닝을 다룰 때 자주 사용되는 플랫폼입니다. 구글에서 제공하는 컴퓨터를 원격으로 빌려서 사용하는 것이기 때문에 인터넷이 꼭 되어야 하는 환경에서 사용할 수 있고, 원격이다 보니 성능이 좋지 않은 노트북으로도 높은 성능의 모델을 돌릴 수 있습니다.

Google Colab을 사용하려면 먼저 인터넷 검색 창에 “Google Colab”을 검색합니다. 참고로 Google Colab을 사용하려면 구글 계정이 있어야합니다. 없다면 회원가입을 해주시고 있다면 로그인을 먼저 진행해주세요.

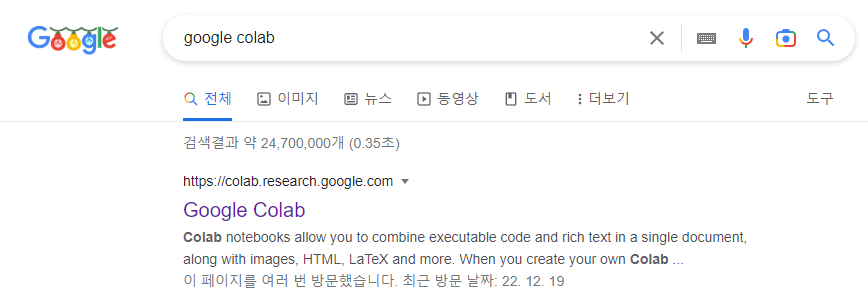


그림 Google Colab 검색

검색 결과의 맨 위에 있는 Google Colab 페이지에 접속하면 가운데에 작은 창이 하나 뜰 것입니다. 창 오른쪽 하단에 있는 “새 노트” 버튼을 누릅니다.

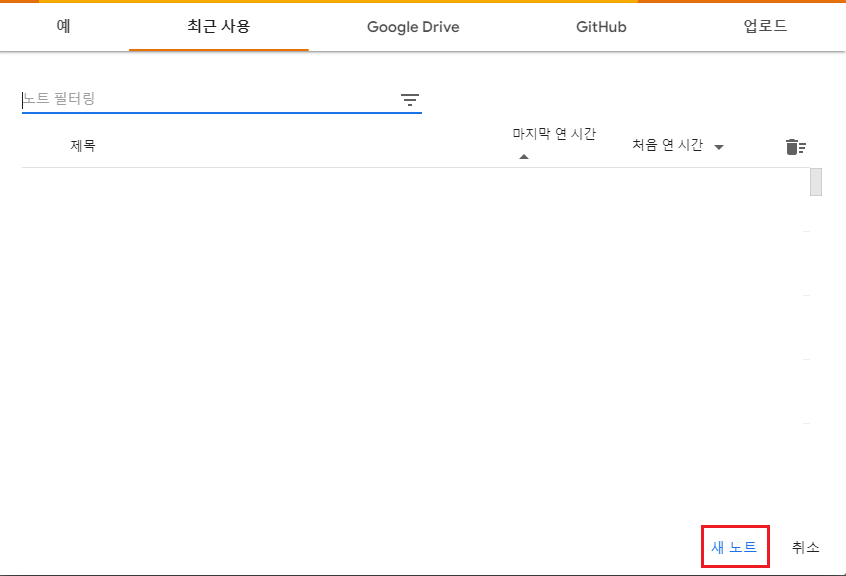


그림 새 노트 버튼 클릭하기

“새 노트” 버튼을 누르면 Google Colab에서 파이썬을 통해 프로그래밍을 진행할 수 있는 코드 창이 나옵니다. 다만 프로그래밍을 진행하기 전에 몇 가지 설정을 해야 합니다. 화면 오른쪽 상단에 있는 톱니바퀴 모양이 보이실 겁니다. 톱니바퀴 모양 버튼을 클릭하겠습니다.

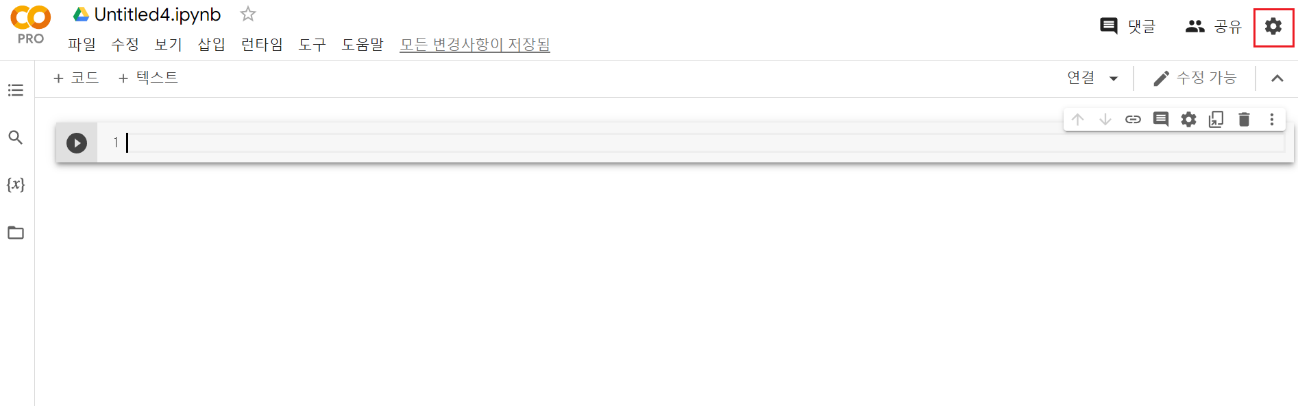


그림 Google Colab 코드 창

톱니바퀴 버튼을 누르면 Google Colab의 환경 창이 나타납니다. 여기서 몇 가지 옵션을 조정하겠습니다.

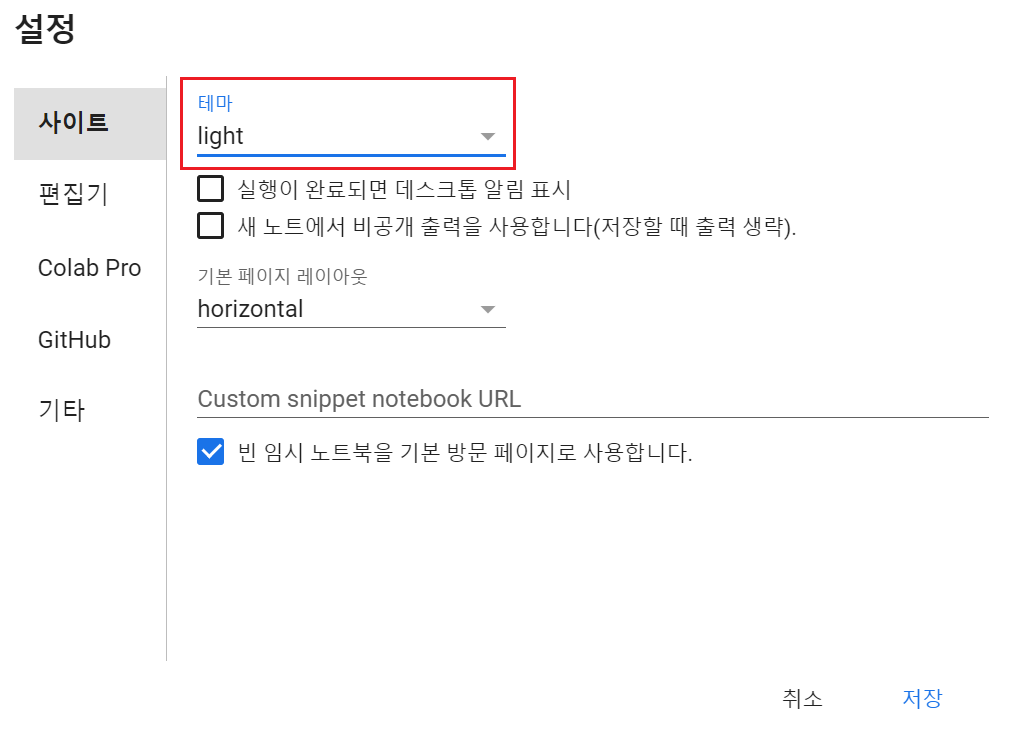


그림 Google Colab 환경 창

왼쪽 탭에 “사이트” 버튼에서는 테마를 바꿀 수 있습니다. 만약 흰색 화면이 프로그래밍을 하는데 불편하다면 “Dark”로 바꿔주세요. 테마를 바꿨다면 왼쪽 탭에 보이는 “편집기” 버튼을 클릭해주세요.

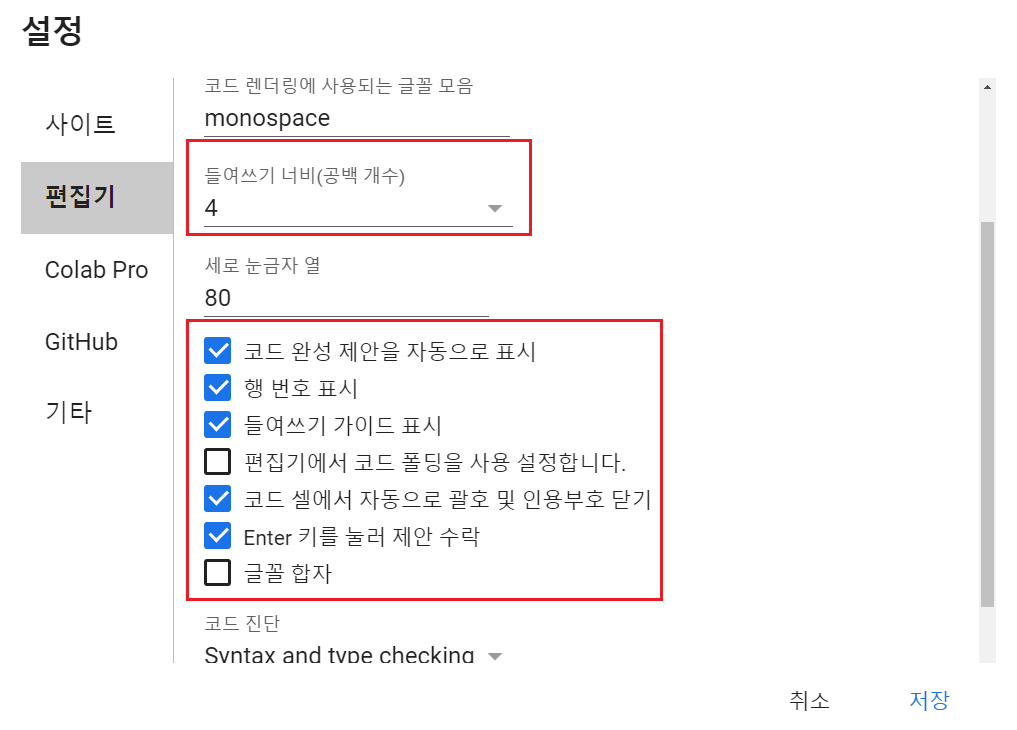


그림 Google Colab 설정창 – 편집기

편집기에서 꼭 들여쓰기 너비를 4로 설정해주세요. 파이썬은 기본적으로 들여쓰기를 4를 기본으로 합니다. 그리고 밑에 있는 체크 박스는 위 그림과 같이 체크해주시고 저장 버튼을 클릭해주세요. 그러면 좀 더 편하게 프로그래밍을 진행하실 수 있습니다.

# 파이썬 기초 프로그래밍

이제 파이썬 프로그래밍을 할 수 있는 환경이 마련되었으니, 본격적으로 파이썬 기초 문법에 대해서 알아보도록 하겠습니다. 시작하기 전에 가장 먼저 Jupyter Notebook에서 우리가 만들었던 python 폴더 안으로 들어가서 새로운 파일을 하나 만들어주세요. 그리고 파일 제목은 “파이썬 기초 문법” 이라고 지정해 놓겠습니다.

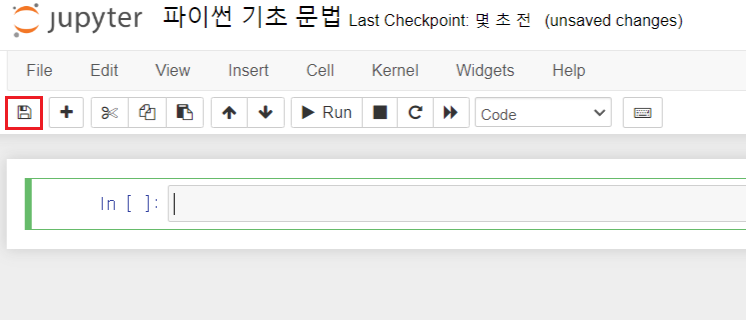


그림 파이썬 기초 문법 제목 설정하기

진행하고 있던 코드를 저장하려면 위의 그림처럼 왼쪽 상단에 디스크 모양 버튼을 누르면 바로 저장할 수 있습니다. 저장된 파일은 우리가 만든 python 폴더 내에 저장이 되어있을 겁니다.

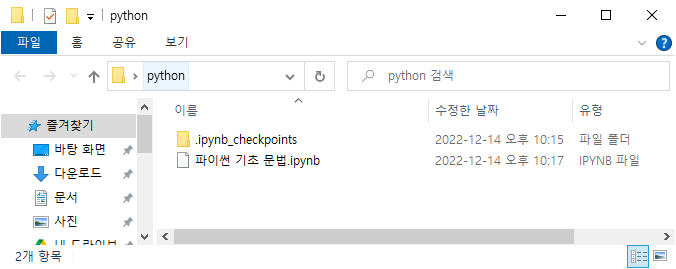


그림 python 폴더에 저장된 파일

## 변수 이해하기

변수(Variable)란 하나의 값을 저장할 수 있는 저장 공간입니다. 변수는 저장되는 값의 형태에 따라서 다른 모습을 지니게 됩니다. 예를 들어 어떤 변수에 숫자가 들어가면 숫자형 변수가 되고, 문자열이 들어가면 문자형 변수가 되는 것이지요. 어떤 값을 어떤 변수에 할당하기 위해서는 등호(=) 모양을 사용해야 합니다. 등호를 기준으로 오른쪽에는 할당하고자 하는 값을 입력해주면 되고, 등호를 기준으로 왼쪽에는 값을 받는 변수를 입력해주면 됩니다.

예시 코드 변수 만들기

|  |
| --- |
| var1 = 5 var2 = "Hello" |

위의 코드를 해석해보면, var1 이라는 변수에는 숫자를 5를 할당한 것이고, var2에는 Hello 라는 문자열을 할당해준 것입니다. 앞서 이야기했지만 문자열을 표현할 때는 항상 문자열 앞, 뒤에 작은 따옴표 또는 큰 따옴표를 붙여주어야 합니다. 다만 변수를 만들 때는 주의해야할 점이 있습니다. 주의해야할 점은 아래와 같습니다.

1. 변수를 만들 때는 영문 문자와 숫자를 사용할 수 있습니다.
2. 대소문자를 구분합니다.
3. 문자부터 시작해야 하며, 숫자부터 시작하면 안 됩니다.
4. \_(밑줄 문자)를 제외한 다른 특수문자는 사용할 수 없습니다.
5. print, len과 같은 파이썬에서 제공하고 있는 함수 이름을 사용하면 안 됩니다.

## 주석 사용하기

주석은 프로그래밍에 있어 내용을 메모하는 목적으로 사용됩니다. 소스 코드를 더 쉽게 이해할 수 있게 만드는 것이 주 목적이며, 협업할 때 특히 유용하게 사용됩니다. 소스 코드가 간단하다면 코드를 이해하는 것이 어렵지 않지만, 코스 코드가 굉장히 길다면 작성자도 시간이 오래 지난 후에 다시 코드를 보면 이해하지 못할 수도 있습니다. 따라서 코드창에 코드가 어떤 역할을 하는지 적어 놓으면 나중에 다시 그 코드를 들여다봤을 때 이해하기가 훨씬 수월합니다. 그런데 코드 창에 메모를 적어 놓으면 에러가 발생합니다. 파이썬이 그 메모조차 코드라고 인식하기 때문입니다. 코드 창에 무언가를 필기할 때는 “#” 모양을 메모할 글 앞에 적어주면 됩니다.

예시 코드 주석 만들기1

|  |
| --- |
| # 여기에 필기하시면 됩니다. |

위의 코드처럼 필기할 내용 앞에 “#” 문자를 사용하면 파이썬이 코드를 실행할 때 “#”가 적힌 줄은 읽지 않기 때문에 에러가 발생하지 않습니다. 또는 문자열로도 주석을 표현하는 방법이 있습니다. 보통 큰 따옴표를 3개 만들어서 주석을 표현합니다.

|  |
| --- |
| """ 여기에 필기하시면 됩니다. """ |

위와 같은 방법으로 주석을 만들면 여러 줄로 필기가 가능합니다.

## 숫자형 자료형

숫자형 자료형은 말 그대로 숫자로 표현된 값을 말합니다. 숫자형 자료형에는 정수형, 실수형 그 밖에 8진수, 16진수 등 다양한 숫자를 표현하는 형태가 있습니다. 가장 많이 사용되는 것은 정수형과 실수형입니다. 사실 프로그래밍 언어는 정수형과 실수형을 엄격하게 구분합니다. 따라서 5 + 3.14를 하면 에러가 나거나, 이상한 값이 나옵니다. 계산이 불가능합니다. 왜냐하면 표현되는 숫자 형태가 다르기 때문에 컴퓨터가 연산을 할 수가 없습니다. 5 + 3.14를 표현하기 위해서는 5.0 + 3.14처럼 두 숫자 모두 실수형으로 변환해주어야 합니다. 하지만 파이썬은 이러한 부분을 신경 쓰지 않아도 됩니다. 표현된 형태가 달라도 연산이 가능합니다. 파이썬이 배우기 쉬운 언어라는 것이 이러한 이유 때문입니다.

예시 코드 숫자형 변수 만들기

|  |
| --- |
| a = 5 b = 3.14 |

위의 코드를 해석해보면, a라는 변수에 정수 5를 할당한 것이며, 변수 b에는 실수 3.14를 할당한 것입니다.

예시 코드 지수 표현

|  |
| --- |
| a = 5.12e10 b = 3.14e-10 |

위의 코드처럼 지수 표현도 가능한데 해석해보면, 5.12 X 10의 10승을 변수 a에 할당한 것이며, 변수 b에는 3.14 X 10의 마이너스 10승을 할당한 것입니다.

### 연산자 알아보기

프로그래밍의 꽃은 당연히 연산이겠죠? 파이썬으로 다양한 연산이 가능합니다. 아마 프로그래밍을 한번도 해보지 않은 독자라도 사칙연산(+, -, \*, /)은 알고 있을 것입니다. 엑셀이나 계산기를 통해서도 같은 특수문자를 활용해서 연산을 하니까요. 간단하게 파이썬을 통해서 연산해봅시다.

예시 코드 사칙 연산

|  |
| --- |
| a = 3 b = 5  print(a + b) print(a - b) print(a \* b) print(a / b) print(a \*\* b) |

[실행 결과]

8

-2

15

0.6

243

실행 결과를 보니 연산이 잘돼서 화면에 출력된 것을 볼 수 있습니다. 여기서 “\*\*” 표현은 제곱을 의미합니다. a \*\* b는 a의 b제곱을 표현하는 것이죠. 몫과 나머지를 표현할 수도 있습니다. 몫은 “//” 문자로 표현하면 되며, 나머지는 “%”로 표현하면 됩니다.

예시 코드 몫과 나머지 표현하기

|  |
| --- |
| print(7 // 3) print(7 % 3) |

[실행 결과]

2

1

위의 코드를 해석해보면, 7 // 3은 7을 3으로 나눈 후의 몫을 의미합니다. 7 % 3은 7을 3으로 나눈 후의 나머지를 의미합니다.

## 문자열 자료형

문자열 자료형은 문자로 표현된 값을 말합니다. 문자열은 숫자처럼 단순히 입력하면 안 되고, 앞에서 설명 드렸듯 앞뒤에 작은 따옴표 또는 큰 따옴표를 사용해야합니다. 작은 따옴표로 문자열을 표현하든 큰 따옴표로 문자열을 표현하든 파이썬은 같은 문자열로 판단합니다.

예시 코드 문자열 표현하기

|  |
| --- |
| print("Hello World!") print('Hello World!') print("""Hello World!""") print('''Hello World!''') |

[실행 결과]

Hello World!

Hello World!

Hello World!

Hello World!

문자열을 표현하는 방법은 크게 4가지가 있습니다.

1. 큰 따옴표로 양쪽 둘러싸기
2. 작은 따옴표로 양쪽 둘러싸기
3. 큰 따옴표 3개를 연속으로 써서 양쪽 둘러싸기
4. 작은 따옴표 3개를 연속으로 써서 양쪽 둘러싸기

파이썬이 문자열을 만드는 방법을 4가지나 만든 이유가 있습니다. 만약에 아래와 같은 문자열이 있다고 가정해보겠습니다.

‘Hong’s House’

파이썬은 위에 있는 문자열이 문자열인지 아닌지 판단하기 위해 가장 앞에 작은 따옴표나 큰 따옴표가 있는지 확인할 것입니다. 근데 문자열 사이에 작은 따옴표가 있으니, 파이썬은 ‘Hong’까지가 문자열이라고 생각하고 뒤에 있는 s House’는 무엇인지 전혀 이해를 하지 못합니다. 그래서 에러를 발생시킵니다. 이러한 문제를 해결하기 위해서는 앞, 뒤에 있는 작은 따옴표를 큰 따옴표로 바꾸어 주어야 합니다. 그래야 파이썬이 어디서부터 어디까지 문자열인지 이해할 수 있기 때문입니다. 큰 따옴표도 마찬가지입니다. 큰 따옴표로 만들어진 문자열 사이에 큰 따옴표가 있다면 에러가 발생할 것입니다. 큰 따옴표를 사용해야하는 문자열이라면 앞, 뒤를 작은 따옴표로 만들어주면 됩니다.

그렇다면 작은 따옴표로 만들어진 문자열 사이에는 작은 따옴표를 쓸 수 없을까요? 그것은 아닙니다. 문자열 사이에 사용된 작은 따옴표가 문자열을 의미하는 작은 따옴표가 아니라 평범한 문자라는 것을 명시해주면 됩니다. 표현하고자 하는 작은 따옴표 앞에 ‘\’ 문자를 함께 사용하면 작은 따옴표로 이루어진 문자열 사이에 작은 따옴표를 사용할 수 있습니다.

예시 코드 '\’ 모양 사용하기

|  |
| --- |
| text = 'Hong\'s House' print(text) |

[실행 결과]

Hong's House

큰 따옴표도 마찬가지입니다. 큰 따옴표로 이루어진 문자열 사이에 큰 따옴표를 사용하려면 큰 따옴표 앞에 ‘\’ 문자를 사용하면 에러 발생 없이 코드를 실행할 수 있습니다. 만약 맥OS를 사용하시는 분들이 있다면 ‘\’ 문자가 없는데 ‘\’ 문자 대신 역 슬래시 문자를 사용해주시면 됩니다.

다른 방법으로는 문자열 앞 뒤에 작은따옴표를 3개 사용하거나, 큰 따옴표를 3개 사용하는 방법이 있습니다. 이러한 방법으로 문자열을 만들 경우 ‘\’ 문자를 사용할 필요 없이 문자열 사이에 작은 따옴표도 사용할 수 있고, 큰 따옴표도 자유롭게 사용할 수 있습니다.

예시 코드 따옴표 3개로 문자열 표현하기

|  |
| --- |
| text1 = '''Hong's House. He says "Hi!".''' print(text1)  text2 = """Hong's House. He says "Hi!".""" print(text2) |

[실행 결과]

Hong's House. He says "Hi!".

Hong's House. He says "Hi!".

문자열을 표현할 때 여러 줄을 표현하기 위해서는 “\n” 문자를 사용해서 표현해야 합니다. “\n” 문자는 줄 바꿈 코드를 의미합니다.

예시 코드 줄바꿈 코드 사용하기

|  |
| --- |
| print("Hello \nWorld!") |

[실행 결과]

Hello

World!

문자열 “Hello” 뒤에 “\n”를 입력하고 “World!” 를 출력해보니, Hello가 화면에 출력되고, 줄 바꿈이 이루어진 후, “World!”가 나온 결과를 볼 수 있습니다.

### 문자열 연산하기

숫자만 연산이 가능한 것이 아니라, 문자열도 연산이 가능합니다. 문자열에서의 덧셈은 서로 붙여줍니다.

예시 코드 문자열 덧셈

|  |
| --- |
| text1 = "python" text2 = " is too fun!" print(text1 + text2) |

[실행 결과]

python is too fun!

문자열의 덧셈이 서로 붙여주는 식으로 연산이 되기 때문에, 문자열끼리 뺄셈은 불가능합니다. 그리고 덧셈이 가능하기 때문에 문자열과 숫자를 곱할 수도 있습니다.

예시 코드 문자열 곱셈

|  |
| --- |
| text1 = "python" print(text1 \* 3) |

[실행 결과]

pythonpythonpython

### 문자열 길이 구하기

문자열의 길이를 알고 싶을 때는 len() 함수를 사용하면 쉽게 문자열이 몇 글자로 이루어져 있는지 알 수 있습니다. len() 함수는 print() 함수와 같이 파이썬에서 기본적으로 제공하고 있는 함수입니다.

예시 코드 문자열 길이 구하기

|  |
| --- |
| text1 = "python" print(len(text1)) |

[실행 결과]

6

### 문자열 인덱싱, 슬라이싱

문자열에 접근하는 것을 문자열 인덱싱이라고 합니다. 예를 들어 “python” 이라는 단어가 있을 때, “python” 이라는 단어의 0번째에 접근하면 문자 “p”를 가져올 수 있습니다. 여기서 중요한 것은 파이썬은 숫자를 셀 때, 0번째부터 센다는 것입니다. 인덱싱을 하려면 대괄호를 사용해서 인덱싱을 해야만 합니다.

예시 코드 문자열 인덱싱

|  |
| --- |
| text1 = "python is too fun!"  print(text1[0]) print(text1[2]) print(text1[-1]) |

[실행 결과]

p

t

!

text1에서 0번째에 접근했더니 “p”가 출력되고, 2번째에 접근하면 “t”가 출력되는 것을 볼 수 있습니다. 참고로 인덱싱할 때, 원하는 위치 값에 음수를 넣어주면 뒤에서부터 위치를 구한다는 의미입니다. 예를 들어 -1번째에 접근하면 뒤에서부터 1번째라는 의미입니다.

인덱싱은 하나씩 접근하는 것이라면, 슬라이싱은 여러 개의 값에 접근하는 것을 말합니다. 슬라이싱을 할 때도 똑같이 대괄호를 사용해서 원하는 위치 값을 넣어주면 됩니다. 여러 개의 접근할 때는 대괄호 안에 콜론(:)을 입력해서 [시작위치:끝위치]를 입력해주면 해당하는 범위의 값에 접근할 수 있습니다.

예시 코드 문자열 슬라이싱

|  |
| --- |
| text1 = "python is too fun!"  print(text1[2:5]) print(text1[:7]) print(text1[3:]) |

[실행 결과]

tho

python

hon is too fun!

위 결과를 해석해보면, text1[2:5]는 2번째부터 5번째전까지라는 의미입니다. 여기서 주의할 것은 2번째부터 5번째까지가 아니라는 것입니다. 마지막 끝위치는 포함하지 않습니다. 즉 2번째부터 4번째까지입니다. 그리고 콜론 앞에 숫자를 넣지 않으면 처음부터 라는 의미이며, 콜론 뒤에 끝 위치를 넣지 않으면 시작 위치부터 끝까지 라는 의미입니다.

### 문자열 전용 함수

우리가 앞서 배웠던 print() 함수 또는 len() 함수는 파이썬에서 기본적으로 제공하고 있는 기본 함수 또는 내장 함수라고 부릅니다. 이러한 내장 함수들은 문자열 자료형 뿐만 아니라 우리가 앞으로 배울 다른 자료형에도 폭넓게 사용할 수 있습니다. 예를 들어 print() 함수는 문자열만 화면에 출력해줄 수 있는 것이 아니고 숫자도 화면에 출력할 수 있었던 것처럼 말이지요. 그런데 문자열에서 사용할 수 있는 문자열 전용 함수가 있습니다. 문자열 전용 함수를 사용하려면 문자열 뒤에 점(.)을 찍고 점 뒤에 함수 이름을 적어주면 됩니다. 문자열 전용 함수에 대해서 하나씩 알아보도록 하겠습니다. 하지만 문자열 전용 함수는 정말 많이 있기 때문에 모두 소개하기에는 한계가 있습니다. 이 책에서는 자주 사용되는 문자열 전용 함수 위주로 설명하겠습니다. 가장 먼저 소개할 문자열 전용 함수는 format() 함수입니다. format() 함수는 문자열 원하는 위치에 내가 입력한 값을 넣어주는 함수입니다.

예시 코드 format() 함수1

|  |
| --- |
| text = "I eat {} apples" print(text.format(3)) |

[실행 결과]

I eat 3 apples

문자열 내에 입력한 값을 넣고 싶은 원하는 위치에 중괄호를 넣어주면 됩니다. 위의 결과를 해석해보면, “I eat {} apples” 에는 “eat”과 “apples” 사이에 중괄호가 있습니다. 그리고 format() 함수 내에 숫자 3을 입력했습니다. 그러면 그 중괄호 안에 숫자 3을 넣겠다는 의미입니다. 중괄호를 여러 개 넣을 수도 있습니다.

예시 코드 format() 함수2

|  |
| --- |
| text = "I eat {} apples and {} oranges" print(text.format(3, 5)) |

[실행 결과]

I eat 3 apples and 5 oranges

첫번째 중괄호에는 format() 함수 안에 첫번째로 입력한 3을 넣겠다는 의미이며, 두번째 중괄호에는 format() 함수 안에 두번째로 입력한 값 5를 넣겠다는 의미입니다. 그래서 “I eat 3 apples and 5 oranges”가 출력되는 것이지요.

format() 함수에 실수형 값을 넣었을 때, 원하는 소수점 자리 까지만 출력하는 방법도 있습니다.

예시 코드 format() 함수3

|  |
| --- |
| a = 3.141592 print("{:.2f}".format(a)) |

[실행 결과]

3.14

소수점이 포함된 실수형 값에서 원하는 소수점 자리까지 출력하려면 중괄호 안에 콜론을 입력하고 뒤에 점(.)을 입력하고 표현하고자 하는 소수점자리를 입력해준 후에 “f”를 입력해주면 됩니다.

문자열에서 특정 문자가 몇 개 들어있는지 확인할 때는 count() 함수를 사용하면 됩니다.

예시 코드 count() 함수

|  |
| --- |
| text = "python is too fun!" print(text.count("o")) |

[실행 결과]

3

위의 결과를 해석해보면 text.count() 함수 안에 “o”를 입력했다는 것은 변수 text 안에 있는 문자열에서 “o”가 몇 개 인지 세겠다는 의미입니다.

문자열에서 특정 값이 어디 있는지 알고 싶다면 find() 함수를 사용하면 됩니다.

예시 코드 find() 함수

|  |
| --- |
| text = "python is too fun!" print(text.find("o")) |

[실행 결과]

4

위의 결과를 해석해보면, text.find() 함수 안에 “o”를 입력하면 변수 text 안에 “o”가 몇 번째에 있는지 알려줍니다. 다만 모든 “o”의 위치를 찾아주는 것이 아니라 가장 처음에 발견한 “o”의 위치만 알려줍니다. 만약 find() 함수 안에 찾고자 하는 문자가 없다면 -1을 반환해줍니다.

문자열 사이에 원하는 값을 넣으려면 join() 함수를 사용하면 됩니다.

예시 코드 join() 함수1

|  |
| --- |
| text = "abcde" print("@".join(text)) |

[실행 결과]

a@b@c@d@e

내가 입력한 “@” 문자를 “abcde” 사이사이에 넣겠다는 의미입니다. 아래와 같은 방법으로도 join() 함수를 쓸 수 있습니다.

예시 코드 join() 함수2

|  |
| --- |
| data = ["one", "two", "three"] print("@".join(data)) |

[실행 결과]

one@two@three

대괄호를 사용해서 안에 문자열 데이터를 3개 넣어주고 data라는 변수에 할당했습니다. 그리고 입력한 “@”를 join() 함수 안에 data를 넣어주면, data 변수 안에 있는 데이터 사이사이에 “@”를 입력해서 하나의 문자열로 만든다는 의미입니다. 대괄호로 만들어진 데이터는 리스트 자료형이라는 것인데, 뒤에서 조금 더 자세히 다루도록 하겠습니다.

입력한 문자열을 모두 소문자로 바꿀 수 있고, 또는 대문자로 바꿀 수도 있습니다. 입력한 문자열을 소문자로 바꾸려면 lower() 함수를 사용하면 되고, 모두 대문자로 바꾸려면 upper() 함수를 사용하면 됩니다.

예시 코드 lower(), upper() 함수

|  |
| --- |
| text1 = "hello world!" print(text1.upper())  text2 = "HELLO WORLD!" print(text2.lower()) |

[실행 결과]

HELLO WORLD!

hello world!

문자열에 만약에 공백이 끼어 있다고 가정해봅시다. 이럴 때는 공백을 없애서 깨끗하게 문자열을 처리해주는 것이 보기에 좋습니다. 왼쪽에 있는 공백, 오른쪽에 있는 공백, 양쪽에 있는 공백을 모두 없애려면 lstrip() 함수, rstrip() 함수, strip() 함수를 사용하면 됩니다.

예시 코드 lstrip(), rstrip(), strip() 함수1

|  |
| --- |
| text = " hello "  print(text.lstrip()) print(text.rstrip()) print(text.strip()) |

[실행 결과]

hello

hello

hello

양쪽에 공백이 있는 변수 text에 lstrip() 함수를 적용하니 왼쪽에 있는 공백이 모두 없어진 것을 확인할 수 있습니다. rstrip() 함수는 오른쪽 공백을 모두 없애 주며, strip() 함수는 양쪽 공백을 모두 없애 주는 함수입니다. 그리고 위의 함수들은 공백만 없앨 수 있는 것이 아닙니다. 기본적으로는 공백을 기준으로 없애 주지만, 함수 안에 없애고자 하는 값을 넣어주면 그 값을 기준으로 앞, 뒤에 있는 값을 없애기도 합니다.

예시 코드 lstrip(), rstrip(), strip() 함수2

|  |
| --- |
| text = "@@@@@hello@@@@@"  print(text.lstrip("@")) print(text.rstrip("@")) print(text.strip("@")) |

[실행 결과]

hello@@@@@

@@@@@hello

hello

특정 단어를 다른 단어로 바꾸는 replace() 함수도 있습니다. replace() 함수에는 2개의 입력 값을 주어야 합니다. 함수에 들어가는 입력 값을 매개변수라고 하는데, 1번째 매개변수에는 기존에 있는 값, 2번째 매개변수에는 바꾸고자 하는 값을 입력해주면 됩니다.

예시 코드 replace() 함수

|  |
| --- |
| text = "python is too fun!" print(text.replace("python", "java")) |

[실행 결과]

java is too fun!

문자열을 공백을 기준으로 자르는 split() 함수도 자주 사용됩니다. split() 함수 역시 기본적으로는 공백을 기준으로 잘라주지만, 특정 문자를 매개변수로 넣어주면 그 값을 기준으로 잘라 주기도 합니다.

예시 코드 split() 함수

|  |
| --- |
| text1 = "python is too fun!" print(text1.split())  text2 = "python:is:too:fun!" print(text2.split(":")) |

[실행 결과]

['python', 'is', 'too', 'fun!']

['python', 'is', 'too', 'fun!']

text1을 보면, 기본적으로는 공백을 기준으로 잘라서 대괄호로 이루어진 어떤 네모 박스에 자른 단어들을 저장해서 반환해줍니다. 이러한 대괄호로 이루어진 네모 박스는 리스트 자료형이라는 것인데, 데이터를 보관하는 자료형입니다. 자세한 내용은 뒤에서 자세히 다루도록 하겠습니다. 하지만 text2 변수처럼 공백이 없는 문장이 있을 수도 있습니다. 이러한 경우에는 특정 문자를 기준으로 자를 수도 있습니다. 특정 문자를 split() 함수 안에 매개변수로 넣어주면 그 문자를 기준으로 잘라서 리스트 자료형으로 만들 수 있습니다.

## 리스트 자료형

리스트 자료형은 여러 개의 값을 저장하고 활용할 수 있는 자료형입니다. 필자가 생각하기에는 파이썬의 자료형 중 가장 중요한 자료형이라고 생각합니다. 일단 많은 데이터를 보관할 수 있어야 데이터 분석도 할 수 있고, 머신러닝도 할 수 있기 때문입니다. 리스트 자료형은 앞서 split() 함수를 다룰 때 보았던 것처럼 대괄호로 이루어져 있습니다. 대괄호 안에 여러 개의 데이터를 콤마(,)를 기준으로 차곡차곡 저장할 수 있습니다.

예시 코드 리스트 기본 구조

|  |
| --- |
| box = [1, 3, 5, 4, 2] |

### 리스트 연산

리스트 자료형도 연산이 가능합니다. 리스트 자료형끼리의 덧셈은 문자열 자료형처럼 서로 붙여줍니다. 이러한 방식으로 연산을 하기 때문에 리스트 자료형 역시 뺄셈은 불가하며, 리스트와 숫자의 곱셈은 가능합니다.

예시 코드 리스트 연산

|  |
| --- |
| box1 = [1, 2, 3] box2 = [4, 5, 6]  print(box1 + box2) |

[실행 결과]

[1, 2, 3, 4, 5, 6]

### 리스트 인덱싱, 슬라이싱

리스트 자료형 역시 문자열 자료형처럼 인덱싱과 슬라이싱이 가능합니다. 사용하는 방법도 같습니다.

예시 코드 리스트 인덱싱, 슬라이싱

|  |
| --- |
| box = [1, 3, 5, 4, 2]  print(box[0]) print(box[-1])  print(box[2:4]) print(box[2:]) print(box[:3]) |

[실행 결과]

1

2

[5, 4]

[5, 4, 2]

[1, 3, 5]

유의해야할 점은 문자열 슬라이싱은 문자열로 결과를 반환하며, 리스트 슬라이싱은 결과를 리스트로 반환한다는 것입니다.

### 리스트 값 수정하기

리스트 안에 있는 값을 수정하려면 수정하고자 하는 값에 인덱싱을 한 후에 인덱싱 한 위치에 수정하고자 하는 값을 할당하면 됩니다. [1,2,3]이라는 리스트를 만들고, 이 리스트에서 숫자 2를 숫자 10으로 바꿔보겠습니다. 숫자 2를 10으로 바꾸려면 2가 있는 곳에 인덱싱을 한 후에 10을 할당하면 됩니다.

예시 코드 리스트 값 수정하기

|  |
| --- |
| box = [1,2,3] box[1] = 10 print(box) |

[실행 결과]

[1, 10, 3]

숫자 2는 변수 box에 1번째에 있기 때문에 box[1]에 인덱싱을 하고, 그 곳에 수정하고자 하는 값을 등호(=)를 사용해서 넣어주면 됩니다.

### 리스트 값 삭제하기

리스트 안에 있는 값을 삭제하는 방법은 2가지가 있습니다. 첫번째는 del 키워드를 이용하는 방법이고, 두번째 방법은 리스트 전용 함수를 이용하는 방법입니다. 두번째 방법은 뒤에서 리스트 전용 함수를 소개할 때 다뤄보도록 하겠습니다. del 키워드를 사용해서 리스트 안에 있는 값을 삭제하려면 del 키워드 옆에 삭제하고자 하는 리스트 값에 인덱싱하면 됩니다. 이번에도 리스트 [1,2,3]에서 숫자 2를 삭제해보겠습니다.

예시 코드 del 키워드로 리스트 값 삭제하기1

|  |
| --- |
| box = [1,2,3] del box[1] print(box) |

[실행 결과]

[1, 3]

삭제하고자 하는 숫자 2가 있는 box[1]에 del 키워드 옆에 인덱싱을 하니 숫자 2가 삭제된 것을 확인할 수 있습니다. del 키워드를 사용해서 인덱싱을 해서 값을 하나만 삭제할 수도 있지만, 슬라이싱을 이용해서 값을 여러 개를 삭제할 수도 있습니다.

예시 코드 del 키워드로 리스트 값 삭제하기2

|  |
| --- |
| box = [1,2,3,1,2,3] del box[2:] print(box) |

[실행 결과]

[1, 2]

del box[2:]를 통해 변수 box의 2번째부터 끝까지 모두 삭제했습니다. 그 결과 0번째 값과, 1번째 값만 남은 것을 확인할 수 있습니다.

### 리스트 길이 구하기

리스트 역시 리스트 안에 있는 값이 총 몇개인지 len() 함수를 사용해서 리스트 안에 있는 값의 개수를 셀 수 있습니다.

예시 코드 리스트 안에 있는 값 개수 세기

|  |
| --- |
| box = [1,2,3] print(len(box)) |

[실행 결과]

3

### 리스트 전용 함수

리스트 전용 함수도 문자열 전용 함수처럼 리스트에만 사용할 수 있는 함수들이 있습니다. 많이 사용되는 전용 함수 위주로 설명해보겠습니다. 리스트 안에 있는 값을 삭제할 때, del 키워드 말고 전용 함수 remove() 함수를 통해 값을 삭제할 수도 있습니다. del 키워드는 값을 삭제하기 위해서 삭제하고자 하는 위치 값에 인덱싱을 했지만, remove() 함수는 매개변수로 삭제하고자 하는 값을 넣어주면 됩니다.

예시 코드 remove() 함수

|  |
| --- |
| box = [1,2,3,1,2,3] box.remove(3) print(box) |

[실행 결과]

[1, 2, 1, 2, 3]

유의해야할 점은 remove() 함수를 통해서 값을 삭제할 때, 요청한 모든 값이 삭제되는 것이 아니라 처음 발견한 값을 삭제해준다는 것입니다. 위의 실행 결과를 보면, 분명 3을 삭제하라고 했는데, 변수 box 안에 앞에 있는 3은 삭제가 되었지만 뒤에 여전히 3이 하나 들어있는 것을 알 수 있습니다. remove() 함수는 처음 발견한 값 하나만 삭제한다는 점을 유의해주세요.

리스트에 새로운 값을 추가할 때는 append() 함수를 사용하면 됩니다. append() 함수를 통해 다양한 값을 추가할 수 있는데, 숫자뿐만 아니라 문자열, 리스트 등 다양한 자료형을 리스트에 추가할 수 있습니다.

예시 코드 append() 함수

|  |
| --- |
| box = [1,2,3] box.append(4) print(box) |

[실행 결과]

[1, 2, 3, 4]

append() 함수를 통해 값을 추가할 때는 리스트의 가장 뒤에 새로운 값이 추가됩니다. 만약 리스트 내에 원하는 위치에 새로운 값을 추가할 때는 insert() 함수를 사용해야 합니다.

예시 코드 insert() 함수

|  |
| --- |
| box = [1,2,3] box.insert(0, 4) print(box) |

[실행 결과]

[4, 1, 2, 3]

insert() 함수 안에는 2개의 매개변수를 입력해야 하는데, 첫번째 매개변수에는 리스트 안에 새로운 값을 추가할 위치, 두번째 매개변수에는 추가할 값을 입력해주면 됩니다.

리스트 안에 있는 값 중에서 특정 값이 어디에 있는지 찾을 때는 index() 함수를 사용해주면 됩니다.

예시 코드 index() 함수

|  |
| --- |
| box = [1,2,3,1,2,3] print(box.index(3)) |

[실행 결과]

2

index() 함수를 사용할 때 유의해야할 점은 index() 함수를 통해 값을 찾을 때, 리스트 안에 특정 값이 여러 개가 있어도, 처음 발견한 값의 위치만 반환하며 만약 없는 값을 찾으려고 한다면 에러가 발생합니다.

리스트 안에 있는 값들을 정렬하는 sort() 함수도 있습니다. sort() 함수는 기본적으로는 오름차순으로 정렬해줍니다.

예시 코드 리스트 정렬하기1

|  |
| --- |
| box = [1,3,5,4,2] box.sort() print(box) |

[실행 결과]

[1, 2, 3, 4, 5]

sort() 함수는 굉장히 강력합니다. 숫자가 들어있는 리스트만 정렬이 가능한 것뿐만 아니라 문자열이 들어있는 리스트도 정렬이 가능합니다.

예시 코드 리스트 정렬하기2

|  |
| --- |
| box = ['banana', 'car', 'apple'] box.sort() print(box) |

[실행 결과]

['apple', 'banana', 'car']

문자열로 이루어진 리스트를 정렬할 때는 사전 순(a, b, c순)으로 정렬을 합니다.

만약 내림차순으로 정렬을 하고 싶다면 sort() 함수 안에 매개변수에 **reverse=True**를 입력해주면 내림차순으로 정렬할 수 있습니다.

예시 코드 리스트 정렬하기3

|  |
| --- |
| box1 = [1,3,5,4,2] box1.sort(reverse=True) print(box1)  box2 = ['banana', 'car', 'apple'] box2.sort(reverse=True) print(box2) |

[실행 결과]

[5, 4, 3, 2, 1]

['car', 'banana', 'apple']

기본적으로 리스트 내에 있는 값들의 가장 뒤에 있는 값을 꺼내는 pop() 함수도 있습니다. remove() 함수와 비슷하다고 볼 수 있지만, 삭제하는 것과 꺼내는 부분에서 약간 다릅니다. remove() 함수는 요청한 값을 삭제하기위해 가장 앞에서부터 순차적으로 값을 찾지만, pop() 함수는 가장 뒤에 있는 값을 바로 꺼내서 없앤다는 점에서 맨 뒤에 있는 값을 삭제해야 하는 경우에는 훨씬 효율적이고 빠른 속도로 원하는 값을 삭제할 수 있습니다.

예시 코드 pop() 함수

|  |
| --- |
| box = [1,3,5,4,2] a = box.pop() print(a) print(box) |

[실행 결과]

2

[1, 3, 5, 4]

## 튜플 자료형

튜플 자료형은 리스트 자료형과 매우 흡사합니다. 튜플 자료형 역시 리스트 자료형처럼 데이터들을 보관하는 역할을 합니다. 하지만 리스트와 똑같으면 존재 이유가 없겠죠? 튜플 자료형은 리스트 자료형과는 다르게 한번 만들면, 절대 수정하거나 삭제할 수가 없습니다. 리스트 자료형은 대괄호를 사용해서 만들었다면, 튜플 자료형은 소괄호를 사용해서 만든다는 차이점도 있습니다.

예시 코드 튜플 자료형 기본 구조

|  |
| --- |
| box = (1,3,5,4,2) |

### 튜플 연산

튜플의 연산은 리스트 연산과 똑같습니다. 2개의 리스트끼리 덧셈을 했을 때 합쳐주었다면, 튜플의 연산 역시 2개의 튜플끼리 덧셈을 한다면 합쳐줍니다. 튜플 역시 뺄셈은 불가능하며, 튜플과 숫자끼리 곱셈은 가능합니다.

예시 코드 튜플 연산하기

|  |
| --- |
| box1 = (1,2,3) box2 = (4,5,6)  print(box1+box2) # 튜플 덧셈 print(box1 \* 3) # 튜플 곱셈 |

[실행 결과]

(1, 2, 3, 4, 5, 6)

(1, 2, 3, 1, 2, 3, 1, 2, 3)

### 튜플 인덱싱, 슬라이싱

튜플 자료형 역시 인덱싱과 슬라이싱이 가능합니다. 그리고 튜플 자료형에서 슬라이싱을 할 경우 결과값을 튜플 자료형을 반환합니다.

예시 코드 튜플 인덱싱, 슬라이싱

|  |
| --- |
| box = (1,3,5,4,2)  print(box[0]) # 튜플 인덱싱 print(box[2:]) # 튜플 슬라이싱 |

[실행 결과]

1

(5, 4, 2)

튜플 자료형이 수정과 삭제가 불가능하다고 해서, 전혀 불가능한 것은 아닙니다. 튜플을 리스트로 변환한다면 수정이 가능합니다. 파이썬에서 기본적으로 제공하는 함수는 list() 함수는 어떤 자료형을 리스트 자료형으로 변환해주는 함수이며, tuple() 함수는 어떤 값을 튜플 자료형으로 변환해줍니다. 만약 튜플 자료형을 수정하거나 삭제해야 하는 상황이라면 잠시 튜플을 리스트로 변환시킨 후에, 값을 수정하거나 삭제하고 다시 튜플로 바꿔주면 됩니다. 만약 튜플 (1,3,5,4,2)가 있다고 했을 때, 이 튜플 안에 있는 숫자 3을 삭제해보도록 하겠습니다.

예시 코드 튜플 안에 있는 값 삭제하기

|  |
| --- |
| box = (1,3,5,4,2) box = list(box) # 튜플 리스트로 변환 del box[1] box = tuple(box) # 리스트 튜플로 변환 print(box) |

[실행 결과]

(1, 5, 4, 2)

위의 결과를 보면 튜플은 한번 정하면 절대 수정하거나 삭제가 불가능한데, (1,3,5,4,2)로 만든 튜플 안에 있는 숫자 3을 삭제한 것을 확인해볼 수 있습니다. 튜플을 list() 함수를 통해서 리스트로 변환한 뒤 다시 box에 할당하고 변수 box의 1번째 값을 삭제한 후에 다시 tuple() 함수를 사용해서 리스트를 튜플로 변환해주었습니다.

### 튜플의 특이한 성질

그리고 튜플에는 한 가지 또 유의해야할 점이 있습니다. 하나의 변수는 하나의 값만 가집니다. 만약 하나의 변수에 여러 개의 값을 넣으려고 한다면 에러가 나겠죠? (여러 개의 값이 들어있는 리스트는 여러 개의 값이 아니라 리스트 하나라고 보시면 됩니다.) 하지만 파이썬에서는 에러가 나지 않습니다. 만약 하나의 변수에 여러 개의 값을 할당한다면, 파이썬은 자동으로 그 여러 개의 값을 튜플로 묶어서 변수에 할당합니다.

예시 코드 여러 개의 값을 하나의 변수에 할당하기

|  |
| --- |
| a = 1,3,5,4,2 print(a) |

[실행 결과]

(1, 3, 5, 4, 2)

리스트는 하나의 값을 가질 수 있지만, 튜플은 기본적으로 하나의 값은 취급하지 않습니다. 하나의 값을 가지는 튜플을 만들려면 값 옆에 콤마(,)를 함께 입력해주어야 합니다.

예시 코드 하나의 값을 가진 튜플

|  |
| --- |
| box = (3) print(box)  box = (3,) print(box) |

[실행 결과]

3

(3,)

### 튜플 길이 구하기

튜플 역시 리스트처럼 튜플 내에 있는 데이터의 개수를 len() 함수를 통해 알 수 있습니다.

예시 코드 튜플 길이 구하기

|  |
| --- |
| box = (1,3,5,4,2) print(len(box)) |

[실행 결과]

5

## 딕셔너리 자료형

딕셔너리 자료형도 리스트, 튜플 자료형처럼 데이터들을 보관하는 역할을 합니다. 하지만 딕셔너리 자료형은 약간 형태가 다릅니다. 딕셔너리는 사전이라는 이름 답게 데이터들을 마치 사전처럼 저장해 놓습니다. 마치 영한사전에 “school” 이라는 단어는 “학교”와 1:1 대응으로 이루어져 있는 것처럼 데이터들을 저장해 놓습니다. 딕셔너리 자료형은 중괄호를 사용해서 만들 수 있습니다.

예시 코드 딕셔너리 자료형 기본 구조

|  |
| --- |
| dic = {"key1" : 'value1', "key2" : 'value2'} |

딕셔너리 자료형은 위와 같이 Key값과 Value으로 데이터들을 1:1 대응시켜 저장합니다. 딕셔너리에 간단하게 몇 가지 정보를 넣고 만들어보도록 하겠습니다.

예시 코드 딕셔너리 만들기

|  |
| --- |
| dic = {'이름' : "홍길동", "나이" : 30, "성별" : "남자"} |

딕셔너리를 만들 때, 유의해야할 점은 딕셔너리 내에 중복된 Key값은 존재할 수 없다는 것입니다. 만약 2개 이상의 같은 값을 가지는 Key값을 가지는 딕셔너리를 만들었다면 가장 마지막에 입력한 Key값을 가지고 그 전에 입력했던 Key값은 사라집니다. 2개의 “이름” 이라는 Key값을 가진 딕셔너리를 만들어보겠습니다.

예시 코드 딕셔너리에 중복된 Key값은 존재할 수 없다

|  |
| --- |
| dic = {'이름' : "홍길동", "나이" : 30, "성별" : "남자", "이름" : "강동원"} print(dic) |

[실행 결과]

{'이름': '강동원', '나이': 30, '성별': '남자'}

마지막에 입력한 “강동원”이 대응된 “이름” Key값만 있다는 것을 알 수 있습니다.

### 딕셔너리 인덱싱

딕셔너리는 순서가 없습니다. 따라서 인덱싱을 할 때도 순서로 인덱싱하는 것은 불가능합니다. 또한 인덱싱을 할 때, 항상 Key값에만 인덱싱을 할 수 있습니다.

예시 코드 딕셔너리 인덱싱

|  |
| --- |
| dic = {'이름' : "홍길동", "나이" : 30, "성별" : "남자"} print(dic['이름']) |

[실행 결과]

홍길동

### 딕셔너리의 성질

항상 Key값에만 인덱싱을 할 수 있는 것처럼 딕셔너리는 무조건 Key값 기준으로만 판단을 합니다. 위에서 만든 딕셔너리에는 “이름”도 들어있고, “홍길동”도 들어있습니다. 한번 둘 다 있는지 확인해볼까요?

예시 코드 딕셔너리 내에 값이 있는지 확인하기

|  |
| --- |
| dic = {'이름' : "홍길동", "나이" : 30, "성별" : "남자"}  print("이름" in dic) print("홍길동" in dic) |

[실행 결과]

True

False

“이름”은 딕셔너리 변수 dic에 있기 때문에 True라고 대답해줍니다. 하지만 “홍길동”이라는 값도 변수 dic 안에 있는데도 불구하고 False라고 대답해줍니다. 이렇게 딕셔너리는 무조건 Key값으로만 판단을 합니다. 여기서 나오는 True, False는 불 자료형이라고 하는 것인데 참과 거짓을 의미하는 자료형입니다. 자세한 내용은 뒤에서 다루도록 하겠습니다.

### 딕셔너리 값 수정하기

딕셔너리 안에 있는 값을 수정하는 방법은 리스트 자료형과 비슷합니다. 다만 딕셔너리는 순서로 인덱싱을 할 수 없기 때문에 바꾸고자 하는 Key값에 인덱싱을 해서 수정하고자 하는 값을 할당해주어야 합니다.

예시 코드 딕셔너리 값 수정하기

|  |
| --- |
| dic = {'이름' : "홍길동", "나이" : 30, "성별" : "남자"} dic['이름'] = "강동원" print(dic) |

[실행 결과]

{'이름': '강동원', '나이': 30, '성별': '남자'}

### 딕셔러리에 값 추가하기

딕셔너리에 어떤 값을 추가할 때는 인덱싱과 비슷하게 해야 합니다. 추가하고 싶은 Key값에 인덱싱을 한 후, 그 곳에 대응하고자 하는 값을 입력해주면 됩니다.

예시 코드 딕셔너리에 값 추가하기

|  |
| --- |
| dic = {'이름' : "홍길동", "나이" : 30, "성별" : "남자"} dic['사는곳'] = "수원" print(dic) |

[실행 결과]

{'이름': '홍길동', '나이': 30, '성별': '남자', '사는곳': '수원'}

“사는곳” 이라는 Key값에 “수원”을 대응시켰더니 해당 값이 딕셔너리 dic 변수에 추가된 것을 확인할 수 있습니다.

### 딕셔너리 전용 함수

딕셔너리도 딕셔너리에만 사용할 수 있는 전용 함수들이 있습니다. 하나씩 소개해보도록 하겠습니다. 가장 먼저 딕셔너리의 Key값들만 확인하는 keys() 함수가 있습니다.

예시 코드 keys() 함수

|  |
| --- |
| dic = {'이름' : "홍길동", "나이" : 30, "성별" : "남자"} print(dic.keys()) |

[실행 결과]

dict\_keys(['이름', '나이', '성별'])

반대로 딕셔너리의 Value값들만 확인하는 values() 함수가 있습니다.

예시 코드 values() 함수

|  |
| --- |
| dic = {'이름' : "홍길동", "나이" : 30, "성별" : "남자"} print(dic.values()) |

[실행 결과]

dict\_values(['홍길동', 30, '남자'])

마지막으로 딕셔너리의 Key값과 Value값을 각각 튜플로 묶어서 리스트로 반환해주는 items() 함수도 있습니다.

예시 코드 items() 함수

|  |
| --- |
| dic = {'이름' : "홍길동", "나이" : 30, "성별" : "남자"} print(dic.items()) |

[실행 결과]

dict\_items([('이름', '홍길동'), ('나이', 30), ('성별', '남자')])

## 집합 자료형

집합 자료형은 우리가 중, 고등학교를 다녔을 때 배웠던 그 집합을 말합니다. 아래 그림처럼 두 개의 집합이 있다면 공통된 집합을 교집합이라 부르며, 두 개의 집합 안에 있는 모든 요소를 합집합, 순수한 A 안에 있는 값들은 A와 B의 차집합이라고 부릅니다.

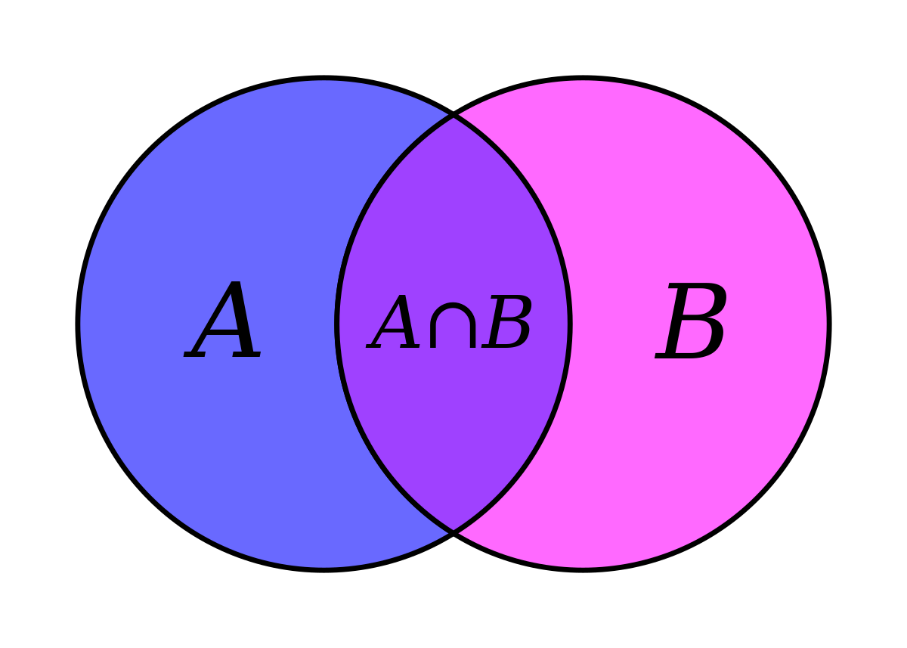


그림 집합(출처: Wikipedia)

집합에는 중요한 성질이 2가지가 있습니다.

1. 중복을 허용하지 않는다.
2. 순서가 없다.

집합은 순서가 없기 때문에 집합 역시 인덱싱은 불가능하고, 딕셔너리처럼 Key값도 없기 때문에 인덱싱 자체가 불가능합니다. 만약 집합 자료형에서 인덱싱이 필요한 경우라면 우리가 앞서 배웠던 list() 함수나 tuple() 함수를 이용해서 집합 자료형을 리스트, 튜플 자료형으로 변환한 후에 인덱싱을 해주어야 합니다.

예시 코드 집합 자료형 리스트 자료형으로 변환하기

|  |
| --- |
| set1 = set([1,2,3,1,1,1]) set1 = list(set1)  print(set1) print(set1[0]) |

[실행 결과]

[1, 2, 3]

1

이러한 성질은 프로그래밍을 할 때 유용하게 사용할 수 있습니다. 만약 무수한 단어들이 있는 데이터가 있을 때, 중복된 단어를 모두 제외하고 고유한 단어들만 보고 싶다면 집합을 사용하면 쉽게 중복된 데이터를 지울 수 있죠. 집합을 만들려면 파이썬에서 제공하고 있는 내장 함수인 set() 함수를 사용해야 합니다.

예시 코드 set() 함수로 집합 만들기

|  |
| --- |
| set1 = set([1,2,3,1,1,1]) print(set1)  set2 = set('Hello World!') print(set2) |

[실행 결과]

{1, 2, 3}

{'r', ' ', 'd', 'e', 'H', '!', 'W', 'l', 'o'}

위의 결과를 보시면, 중복된 데이터가 모두 사라지고, 고유한 데이터들만 집합에 남은 것을 볼 수 있습니다. 집합 자료형 역시 중괄호로 표현이 되기 때문에 처음에는 딕셔너리와 집합 자료형이 헷갈릴 수 있습니다. 중괄호 안에 데이터가 Key값과 Value값이 대응되어 있는 상태로 저장되어 있다면 딕셔너리이고, 아니라면 집합 자료형이라고 이해하면 됩니다.

### 교집합, 합집합, 차집합

두 개의 집합이 있다고 했을 때, 교집합을 알아보려면 두 개의 집합 사이에 “&” 문자를 사용해주면 됩니다.

예시 코드 교집합

|  |
| --- |
| set1 = set([1,2,3,4,5,6]) set2 = set([4,5,6,7,8,9])  print(set1 & set2) |

[실행 결과]

{4, 5, 6}

또한 합집합을 알아보려면 두 개의 집합 사이에 “|” 문자를 사용해주면 됩니다. “|” 문자는 Shift+”\”을 눌러주면 만들 수 있습니다.

예시 코드 합집합

|  |
| --- |
| set1 = set([1,2,3,4,5,6]) set2 = set([4,5,6,7,8,9])  print(set1 | set2) |

[실행 결과]

{1, 2, 3, 4, 5, 6, 7, 8, 9}

두 개의 집합의 차집합을 알아보려면 단순하게 “-“기호를 사용해서 빼 주면 됩니다. 만약 집합A와 B가 있다고 했을 때, A-B를 하면 순수한 A에 대한 값들을 볼 수 있고, B-A를 하면 순수한 B에 대한 값들을 확인할 수 있습니다.

예시 코드 차집합

|  |
| --- |
| set1 = set([1,2,3,4,5,6]) set2 = set([4,5,6,7,8,9])  print(set1 - set2) print(set2 - set1) |

[실행 결과]

{1, 2, 3}

{8, 9, 7}

### 집합 전용 함수

집합에 어떤 값을 추가하고자 한다면 add() 함수와 update() 함수를 사용할 수 있습니다. add() 함수는 값을 하나만 추가할 수 있으며, update() 함수는 값을 여러 개 한꺼번에 추가할 때 사용합니다.

예시 코드 add() 함수, update() 함수

|  |
| --- |
| set1 = set([1,2,3,4,5,6]) set1.add(7) print(set1)  set1.update([8,9,10]) print(set1) |

[실행 결과]

{1, 2, 3, 4, 5, 6, 7}

{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

집합 안에 있는 특정 값을 삭제하려면 remove() 함수를 사용하면 됩니다.

예시 코드 remove() 함수

|  |
| --- |
| set1 = set([1,2,3,4,5,6]) set1.remove(6) print(set1) |

[실행 결과]

{1, 2, 3, 4, 5}

## 불 자료형

불 자료형은 참과 거짓을 의미하는 자료형입니다. 파이썬은 모든 값을 참 아니면 거짓으로 분류할 수 있습니다. 불 자료형은 우리가 뒤에서 조건문을 다룰 때 특히 많이 사용하게 될 겁니다. 왜냐하면 조건이 참일 때와 거짓일 때 다르게 프로그램을 수행하는 코드를 작성하는 경우가 많은데 그 때 꼭 확인하는 것이 불 자료형이기 때문입니다.

### 불 연산

파이썬은 어떤 연산에 있어서 연산 결과를 항상 불 자료형으로 반환해줍니다.

예시 코드 연산 결과 확인하기

|  |
| --- |
| print(5 > 3) print(3 > 5) |

[실행 결과]

True

False

“5가 3보다 크다” 라고 말하니 파이썬은 참(True)이라고 대답해주었고, 반대로 “3이 5보다 크다” 라고 말하니 파이썬은 거짓(False)이라고 대답해주는 것을 확인할 수 있습니다. 당연한 결과죠? 앞서 불 자료형은 모든 값을 참 또는 거짓으로 분류할 수 있다고 했습니다.

### 불 자료형의 성질

그렇다면 단순히 숫자 5는 참일까요 거짓일까요? 애매하죠? 어떤 값이 참인지 거짓인지 알기 위해서는 파이썬에서 기본적으로 제공하는 bool() 함수를 통해 알 수 있습니다. 숫자 5가 참인지 거짓인지 bool() 함수를 통해 알아봅시다.

예시 코드 bool() 함수

|  |
| --- |
| print(bool(5)) |

[실행 결과]

True

숫자 5를 bool() 함수를 통해 참인지 거짓인지 알아보니 참입니다. 이렇게 연산하지 않아도 파이썬은 모든 값을 참 또는 거짓으로 분류할 수 있습니다. 그런데 왜 참(True)이라고 분류했을까요? 파이썬에서는 비어 있는 값은 거짓, 비어 있지 않으면 참이라고 분류합니다. 숫자 5는 비어 있지 않다고 판단했기 때문에 참이라고 분류한 것입니다. 파이썬에서는 숫자형 자료형에서는 숫자 0을 제외하면 모두 참이라고 분류합니다.

예시 코드 bool() 함수 성질 알아보기1

|  |
| --- |
| print(bool(0)) |

[실행 결과]

False

리스트나 튜플, 딕셔너리 등 다른 자료형도 마찬가지입니다. 비어 있으면 거짓, 비어 있지 않으면 참이라고 판단합니다.

예시 코드 bool() 함수 성질 알아보기2

|  |
| --- |
| print(bool([1,3,5,4,2])) print(bool([]))  print(bool("Hello")) print(bool(""))  print(bool({"이름" : "홍길동"})) print(bool({})) |

[실행 결과]

True

False

True

False

True

False

리스트 안에 값이 하나 이상이라도 들어 있다면 참, 아니면 거짓인 것을 알 수 있고, 문자열, 딕셔너리도 마찬가지입니다. 따라서 자료형의 참과 거짓을 구분하는 기준은 아래와 같습니다.

표 참, 거짓 구분 기준표

|  |  |  |
| --- | --- | --- |
| **자료형** | **값** | **참/거짓** |
| 문자열 | “Hello” | 참 |
| “” | 거짓 |
| 리스트 | [1,2,3] | 참 |
| [] | 거짓 |
| 튜플 | (1,2,3) | 참 |
| () | 거짓 |
| 딕셔너리 | {“이름” : “홍길동”} | 참 |
| {} | 거짓 |
| 숫자 | 0이 아닌 숫자 | 참 |
| 0 | 거짓 |

값이 비어 있는지, 비어 있지 않는지 확인한다면 쉽게 참인지 거짓인지 분류할 수 있습니다.

## 데이터 복사

어떤 변수를 다른 변수에 복사할 때 어떻게 해야 할까요? 단순히 변수에 할당하면 된다고 생각하는 분들이 많을 겁니다. 단순히 특정 변수를 다른 변수에 할당하고 출력해보겠습니다.

예시 코드 데이터 복사하기1

|  |
| --- |
| a = 5 b = a  print(a) print(b) |

[실행 결과]

5

5

결과를 확인해보면, 값이 잘 복사되었다고 생각할 수 있습니다. 하지만 복사된 것이 아닙니다. 위에 있는 코드를 잘 해석해보면, 변수 a에 5를 넣은 것은 변수 a가 5가 있는 곳을 바라보게 한 것입니다. 그리고 변수 b에 a를 할당한 것도 a가 5를 바라보고 있으니, 변수 b도 5를 바라보게 한 것이라고 이해해야 합니다. 이해를 돕기 위해 이번에는 리스트 형태로 데이터를 복사해보겠습니다.

예시 코드 데이터 복사하기2

|  |
| --- |
| a = [1,3,5,4,2] b = a  b[1] = 10  print(b) print(a) |

[실행 결과]

[1, 10, 5, 4, 2]

[1, 10, 5, 4, 2]

위에 코드를 보시고, 실행 결과를 보시면 조금 이상하다고 느낄 것입니다. 변수 a를 b에 복사했고, b의 값만 바꿨을 뿐인데, a의 값도 같이 바뀐 것을 알 수 있습니다. 즉, 이것은 복사한 것이 아니라 a가 리스트 [1,3,5,4,2]를 바라보게 했고, 변수 b에 a를 할당한 것도 변수 b가 a가 보고 있는 곳을 보게 했을 뿐입니다. 따라서 a와 b가 같은 곳을 바라보고 있었고, b가 바라보고 있는 곳의 값을 수정했으니 a가 바라보고 있는 값도 같이 바뀌게 된 것이죠.

### copy 모듈로 복사하기

만약 정말로 데이터를 복사해야 한다면 copy라는 라이브러리 안에 있는 copy 함수를 사용해야 합니다. 라이브러리라는 것은 다른 누군가가 만들어 놓은 프로그램을 가져와서 쓰는 것이라고 이해해주시면 됩니다. 아래에서는 간단히 copy 라이브러리를 활용해서 데이터를 복사해보고, 라이브러리에 대해서는 뒤에서 자세히 다루도록 하겠습니다.

예시 코드 copy 라이브러리를 활용해서 데이터 복사하기

|  |
| --- |
| from copy import copy  a = [1,3,5,4,2] b = copy(a)  b[1] = 10  print(a) print(b) |

[실행 결과]

[1, 3, 5, 4, 2]

[1, 10, 5, 4, 2]

copy 함수를 활용해서 a를 복사해서 b에 할당했기 때문에 각자 다른 [1,3,5,4,2]를 바라보게 됩니다. 그래서 b의 값을 바꿔도 a의 값이 바뀌지 않는 것입니다. 이러한 복사에 대한 개념이 잘 안되어 있다면 프로그래밍을 할 때 이상한 결과를 많이 만나 보실 수 있으니 유의해 주셔야 합니다.

## 변수 교환하기

프로그래밍을 할 때, 두 개의 변수가 가지고 있는 값을 서로 교환하는 방법도 일반적인 방법으로는 힘듭니다. 단순히 아래와 같이 코딩해서 두 개의 변수를 교환해보겠습니다.

예시 코드 변수 교환하기1

|  |
| --- |
| a = 3 b = 5  a = b b = a  print(a) print(b) |

[실행 결과]

5

5

변수 a에 3을 넣고, b에 5를 넣고나서 a에 b를 넣고, b에 a를 넣어서 두 개의 변수를 교환했습니다. 하지만 결과를 보면 둘 다 5가 출력된 것을 볼 수 있습니다. 왜냐하면 a에 b를 넣은 순간 3이었던 a는 5가 되어버렸고, 변수 b에 a를 넣으면 5가 b에 들어가기 때문에 a에도 5가 들어있고, b에도 5가 들어있게 되어 둘 다 5가 출력된 것입니다. 만약 2개의 변수를 교환하려면 아래와 같이 입력해주면 됩니다.

예시 코드 변수 교환하기2

|  |
| --- |
| a = 3 b = 5  a,b = b,a  print(a) print(b) |

[‘실행 결과]

5

3

파이썬에서는 두 개의 변수를 교환할 때, 단순히 a,b = b,a 처럼 변수의 위치를 각각 바꿔서 적어주면 됩니다. 신기하죠? 이러한 간단한 방식은 오로지 파이썬에서만 가능하며, 다른 언어에서는 사용할 수 없습니다.

## 조건문

### if문

드디어 조건문으로 넘어갑니다. 이제부터 본격적으로 컴퓨터에게 일을 시키는 방법을 배운다고 보시면 됩니다. 프로그래밍을 할 때, 상황에 따라서 다르게 프로그램이 실행되게끔 해야 하는 경우가 많습니다. 예를 들어 학생이 50점 이상이면 “합격” 50점 미만이면 “불합격”을 출력해주어야 하는 상황에서도 학생의 점수를 보고 50점보다 높은 지 높지 않은 지 판단해야 합니다. 이러한 기능은 모두 조건문 IF문을 사용해서 만들 수 있습니다. IF문의 기본 구조는 아래와 같습니다.

예시 코드 IF문의 기본 구조

|  |
| --- |
| if 조건:  수행할 코드  ... |

if문을 사용하려면 if를 적고나서 조건을 적어주고 마침표인 콜론(:)까지 꼭 입력해주어야 합니다. 그러면 if문 밑에 4칸이 자동으로 들여쓰기가 됩니다. 4칸이 들여쓰기가 된 곳에 수행할 코드를 작성합니다. 들여쓰기는 파이썬에서 어디에 속하는지를 의미합니다. 따라서 수행할 코드는 if문 안에 속해 있다는 의미입니다. 간단하게 if문을 사용하는 예시를 만들어보겠습니다.

예시 코드 if문 활용하기

|  |
| --- |
| money = True if money == True:  print("Taxi") |

[실행 결과]

Taxi

위 코드를 해석해보면 맨 윗줄에 money 라는 변수에 참(True)을 할당해 놓습니다. 그리고나서 if문을 통해 money가 True가 같은 지 판단합니다.

### 비교 연산자

참고로 등호가 “==” 이렇게 2개 연속으로 적히면 “같다”라는 의미입니다. 이렇게 크다(>), 작다(<), 같다(==) 등 비교하는 연산자를 비교 연산자라고 하는데, 비교 연산자의 종류는 아래와 같습니다.

표 비교 연산자

|  |  |
| --- | --- |
| **비교 연산자** | **기능** |
| X < Y | X가 Y보다 작다 |
| X > Y | X가 Y보다 크다 |
| X == Y | X와 Y가 같다 |
| X != Y | X와 Y는 다르다 |
| X >= Y | X가 Y보다 같거나 크다 |
| X <= Y | X가 Y보다 같거나 작다 |

### else문 활용하기

참일 때 “Taxi”가 화면에 출력되고, 만약 참이 아니면 “Walk”가 출력되게끔 하려면 어떻게 해야 할까요? 그렇게 하려면 if문과 else문을 함께 사용해주면 됩니다. else문은 if문이 참이 아닐 때 실행되는 조건문과 같이 쓰이는 키워드입니다. else문도 if문도과 마찬가지로 콜론을 사용해서 마침표를 찍어줘야 하며, 4칸을 들여쓰기를 하고 수행할 코드를 작성해주어야 합니다. 앞서 사용했던 if문을 이어서 작성해보겠습니다.

예시 코드 else문 활용하기

|  |
| --- |
| money = False  if money == True:  print("Taxi") else:  print("Walk") |

[실행 결과]

Walk

이번에는 변수 money에 거짓(False)을 할당했습니다. 그리고 if문을 활용해서 money가 참과 같은 지 확인했습니다. 하지만 money는 거짓이기 때문에 참이 아니죠? 따라서 if문의 조건이 참이 아니기 때문에 if문이 실행되지 않습니다. if문의 조건이 참이 아닐 때는 밑에 있는 else문이 실행됩니다.

### 논리 연산자

조건을 물어볼 때, 여러 개의 조건을 물어볼 수도 있습니다. 여러 개의 조건을 물어볼 때 사용하는 키워드는 or과 and가 있습니다. or는 “또는” 이라는 뜻으로 조건 중 하나만 참이어도 참입니다. and는 “그리고” 라는 뜻으로 모든 조건이 참이어야 참입니다. 이러한 연산자는 논리 연산자라고 부릅니다.

예시 코드 논리 연산자 or 활용하기

|  |
| --- |
| money = 2000 card = True  if money >= 3000 or card == True:  print("Taxi") else:  print("Walk") |

[실행 결과]

Taxi

위의 결과를 보면, money가 2000이어서 3000보다 같거나 크지 않으니 조건이 참이 아닙니다. 하지만 or 키워드 옆에 card가 참인지 물어보는 조건이 하나 더 있습니다. 그리고 or이기 때문에 둘 중 하나만 참이어도 참이기 때문에 if문은 참이 되어 “Taxi”를 화면에 출력하게 됩니다. and 연산자도 한번 사용해보겠습니다.

예시 코드 논리 연산자 and 활용하기

|  |
| --- |
| money = 2000 card = True  if money >= 3000 and card == True:  print("Taxi") else:  print("Walk") |

[실행 결과]

Walk

위의 코드는 money가 3000보다 커야 하고 동시에 card가 참이어야 하기 때문에 두 조건이 모두 참이 아니어서 “Walk”를 화면에 출력하게 됩니다. 논리 연산자의 종류는 아래와 같습니다.

표 논리 연산자

|  |  |
| --- | --- |
| **논리 연산자** | **기능** |
| X or Y | X와 Y 둘 중에 하나만 참이어도 참 |
| X and Y | X와 Y 모두 참이어야 참 |
| not Y | X가 거짓이면 참 |

### 멤버 연산자

if문과 함께 사용되는 연산자 중에 멤버 연산자라는 것도 있습니다. 멤버 연산자는 해당 값이 어떤 값 안에 속해 있는지 판단하는 연산자입니다. 멤버 연산자는 in이라는 키워드를 사용합니다.

예시 코드 멤버 연산자 in 키워드 활용하기

|  |
| --- |
| pocket = ['paper', 'iphone', 'money']  if "money" in pocket:  print("Taxi") else:  print("walk") |

[실행 결과]

Taxi

위에 코드에서 if문을 보면 “money” 라는 문자열이 pocket 변수 안에 있는지 in 키워드를 통해서 물어보고 있습니다. 마치 영어 문법과 거의 흡사하죠? 반대로 not in 키워드를 통해서 특정 값이 어떤 값 안에 없는지도 확인할 수 있습니다. 멤버 연산자는 주로 아래 표에 나와있는 상황에서 많이 사용됩니다.

표 멤버 연산자

|  |  |
| --- | --- |
| **in** | **not in** |
| x in 리스트 | x not in 리스트 |
| x in 튜플 | x not in 튜플 |
| x in 문자열 | x not in 문자열 |
| x in 딕셔너리 | x not in 딕셔너리 |

### elif문 활용하기

if문과 else문은 참일 때는 if문, 거짓일 때는 else문을 실행합니다. 하지만 이렇게 두 가지 상황만 있는 경우가 아니라면 어떻게 해결할 수 있을까요? 이러한 문제는 elif문으로 해결이 가능합니다. elif문은 if문 아래에 사용할 수 있는데, if문에 물어보는 조건 외에 조건이 더 있을 때 사용합니다. 만약 학생들의 점수가 90점 이상이면 “A”, 80점 이상이면 “B”, 70점 이상이면 “C”, 그 외에는 “D”의 학점을 줘야하는 프로그램을 짠다고 해보겠습니다. 조건이 많죠? 이런 경우에는 elif문을 사용하면 쉽게 해결할 수 있습니다. 해당 내용을 elif문을 활용하여 코드로 표현해보겠습니다.

예시 코드 elif문 활용하기

|  |
| --- |
| student = 86  if student >= 90:  print("A") elif student >= 80:  print("B") elif student >= 70:  print("C") else:  print("D") |

[실행 결과]

B

위에 코드처럼 elif문은 if문 아래에 연속해서 사용할 수 있습니다. 가장 먼저 if문의 조건을 확인하고, if문의 조건이 거짓이면 아래에 있는 elif문을 실행하고, 또 거짓이면 아래에 있는 elif문을 실행하면서 순차적으로 아래로 조건이 참이 될 때까지 조건을 물어봅니다. 그리고 조건이 참이 발견되면 더 이상 내려가지 않고 조건문을 끝냅니다.

## 반복문

컴퓨터가 사람보다 확실히 잘할 수 있는 것이 바로 단순한 반복 노동일 것입니다. 사람은 잠도 자야 하고 실수도 하지만, 컴퓨터는 절대 잠도 자지 않고 실수도 하지 않으니까요. 아주 단순한 반복적인 일은 컴퓨터에게 시킬 수 있다면 아주 효과적으로 높은 생산성을 기대할 수 있습니다. (필자도 반복문을 통해서 단순한 재택 근무를 해서 돈을 벌어다 주는 프로그램을 만들어 소소하게 용돈을 벌고 있습니다.)

### While문

반복문은 크게 2가지가 있습니다. while문과 for문입니다. while문은 조건이 참인 동안 반복하는 반복문이며, for문은 요청한 만큼 순차적으로 반복하는 반복문입니다. while문은 주로 몇 번 반복해야 할지 모를 때 사용합니다. while문의 기본 구조는 아래와 같습니다.

예시 코드 while문의 기본 구조

|  |
| --- |
| while 조건:  수행할 코드  ... |

반복문 while은 조건이 참인 동안 계속해서 반복합니다. 즉, 조건이 참인 동안 while문 내에 속한 수행할 코드를 계속해서 실행하는 것이죠. 반복문 while문의 진행 프로세스는 아래와 같습니다.

1. while문의 조건이 참인지 확인한다. 거짓이라면 반복문을 실행하지 않는다.
2. 참이라면 while문 안에 있는 코드를 순차적으로 실행한다.
3. while문 안에 있는 모든 코드가 끝나면 (1)로 이동한다.

while문을 사용할 때는 주의해야할 점이 있습니다. 만약 조건이 항상 참이라면 무한 반복한다는점입니다. while문을 사용할 때는 반복문이 무한 반복하지 않게 만드는 것이 중요합니다. while문을 활용해서 간단한 반복문 코드를 만들어보겠습니다.

예시 코드 무한 반복하는 while문

|  |
| --- |
| hit = 0 while hit < 10:  print("Hello") |

[실행 결과]

Hello

Hello

...무한반복...

Hello

Hello

위의 코드를 해석해보면 hit에 0을 할당하고, while문에서 변수 ht이 10보다 작은지를 묻고 있습니다. 10보다 작기 때문에 while문 안에 있는 코드가 실행됩니다. 화면에 “Hello”가 출력되고, 코드가 모두 끝났으므로 다시 while문에게 hit이 10보다 작은지 묻습니다. hit은 여전히 10보다 작기 때문에 while문이 또 다시 실행됩니다. 문제는 이 과정이 무한 반복한다는 것입니다. 만약 프로그램이 무한 반복하는 코드가 되었다면, jupyter Notebook 상단에 있는 정지 버튼을 눌러서 코드를 강제로 멈추어야 합니다.

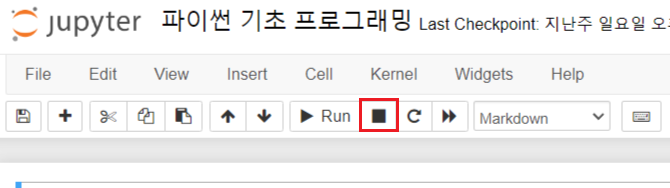


그림 Jupyter Notebook에서 코드 강제로 멈추기

Jupyter Notebook에서 정지 버튼을 누르면 코드에 강제로 오류를 발생시켜 멈추는 작업을 합니다. while문을 무한 반복하지 않게 하는 방법은 여러가지가 있지만 위와 같은 코드에서는 hit이 언젠가는 10보다 크게 되게끔 해주면 됩니다. while문이 한번 실행될 때마다 변수 hit에 1씩 더해보겠습니다.

예시 코드 while문 10번 실행하기

|  |
| --- |
| hit = 0 while hit < 10:  hit = hit + 1  print("Hello") |

[실행 결과]

Hello

Hello

Hello

Hello

Hello

Hello

Hello

Hello

Hello

Hello

위 코드를 해석해보면, 변수 hit은 처음엔 0이었지만 while문이 한번 실행될 때마다 변수 hit에 1을 더한 것을 다시 hit에 할당함으로써 1씩 증가시키고 있습니다. 언젠가는 hit이 10보다 커지면서 while문의 조건이 거짓이 되고 while문은 총 10번만 실행된다는 것을 알 수 있습니다.

다만 while문을 사용할 때는 조건을 얼마나 줘야 하는지 모르는 경우가 많습니다. 예를 들어 우리나라 네이버 블로그에서 맛집 블로그만 선택해서 정보를 크롤링해야 하는 경우를 생각해보겠습니다. 여기서 크롤링이란 인터넷에서 데이터를 수집하는 기술을 말합니다. 네이버 블로그에 맛집 블로그에 접근해서 데이터를 가져올 때 조건을 어떻게 줘야 할까요? 네이버 블로그의 맛집 블로그 개수가 몇 개인지 알 수가 없기 때문에 조건을 주기가 매우 어렵습니다. 따라서 while문을 사용할 때는 주로 조건을 무조건 참으로 두는 편입니다.

예시 코드 while문 조건이 항상 참일 때1

|  |
| --- |
| coffee = 10 while True:  coffee = coffee - 1  print("커피를 한잔 팔고, {}잔 남았습니다.".format(coffee)) |

[실행 결과]

...생략...

커피를 한잔 팔고, -488015잔 남았습니다.

커피를 한잔 팔고, -488016잔 남았습니다.

커피를 한잔 팔고, -488017잔 남았습니다.

커피를 한잔 팔고, -488018잔 남았습니다.

커피를 한잔 팔고, -488019잔 남았습니다.

위의 코드를 보면 while문의 조건이 참(True)인 것을 알 수 있습니다. 반복문의 조건을 이렇게 설정하면 무한 반복하는 반복문이 되어버립니다. 처음에 coffee 변수에 10을 할당하고, while문이 한번 실행될 때마다 1씩 빼고 있습니다. 그리고 0잔이 되었을 때 코드를 멈추고 싶지만 조건이 항상 참이기 때문에 coffee의 값이 음수로 계속해서 내려가는 것을 볼 수 있습니다.

#### break

이 문제를 해결하려면 어떻게 해야 할까요? break 키워드를 사용하면 쉽게 이 문제를 해결할 수 있습니다. 반복문 실행 중에 break 키워드를 만나게 되면 반복문을 강제로 나가버립니다. break 키워드를 사용해서 coffee의 값이 0이 되었을 때, while문을 강제로 멈추어 보겠습니다.

예시 코드 while문 조건이 항상 참일 때2

|  |
| --- |
| coffee = 10 while True:  coffee = coffee - 1  print("커피를 한잔 팔고, {}잔 남았습니다.".format(coffee))  if coffee == 0:  break |

[실행 결과]

커피를 한잔 팔고, 9잔 남았습니다.

커피를 한잔 팔고, 8잔 남았습니다.

커피를 한잔 팔고, 7잔 남았습니다.

커피를 한잔 팔고, 6잔 남았습니다.

커피를 한잔 팔고, 5잔 남았습니다.

커피를 한잔 팔고, 4잔 남았습니다.

커피를 한잔 팔고, 3잔 남았습니다.

커피를 한잔 팔고, 2잔 남았습니다.

커피를 한잔 팔고, 1잔 남았습니다.

커피를 한잔 팔고, 0잔 남았습니다.

조건이 항상 참인 while문에서 coffee의 값이 1씩 계속 빠지고, 조건문 if문을 통해 coffee의 값이 0이 되었을 때 break키워드를 만나게끔 만들었습니다. 그 결과 정말 coffee의 값이 0이었을 때 까지만 while문이 실행되고 반복문을 강제로 빠져나간 것을 알 수 있습니다.

#### continue

break 키워드랑 비슷하지만 살짝 다른 continue라는 키워드도 있습니다. break 키워드는 반복문 실행 중 break를 만나면 반복문을 강제로 나가게 하는 거라면, continue 키워드는 반복문을 나가는 것은 아니고 continue 키워드를 만나는 그 즉시 다시 반복문의 조건을 물어보는 곳으로 이동시킵니다. 간단하게 활용 예시를 보도록 하겠습니다.

예시 코드 continue 키워드를 활용해서 while문 사용하기

|  |
| --- |
| a = 0 while a < 10:  a = a + 1  if a % 2 == 0:  continue  print(a) |

[실행 결과]

1

3

5

7

9

위의 코드를 해석해보면, 변수 a에 0이 할당되어 있고, while문의 조건에서 a가 10보다 작으면 조건이 참이기 때문에 while문이 실행이 됩니다. 그리고 무한 반복하지 않기 위해서 while문이 한번 실행될 때마다 변수 a에 1씩 더하고 있습니다. 그리고 만약 a를 2로 나눈 나머지가 0과 같을 경우(a가 짝수라는 의미입니다.) continue 키워드를 만나게 됩니다. 즉 변수 a가 짝수일 때마다 continue 키워드를 만나게 되는 것이죠. continue 키워드를 만나게 되면, while문의 조건을 물어보는 부분부터 다시 시작합니다. 따라서 print(a)를 만나보지도 못하고 다시 반복문의 시작점으로 이동하는 것이죠. 그리고 a가 홀수일 경우에는 if문이 실행이 되지 않기 때문에 print(a)를 만나서 화면에 a를 출력하게 됩니다. 따라서 이 코드는 홀수만 출력하는 프로그램입니다.

### for문

반복문 for문은 요청한 만큼 순차적으로 반복하는 반복문입니다. 주로 몇 번 반복해야 할지 알 때 사용합니다. for문을 통해서 문자열이나 리스트 내에 있는 것들을 순차적으로 꺼낼 수 있습니다. for문의 기본 구조는 아래와 같습니다.

예시 코드 for문의 기본 구조

|  |
| --- |
| for 변수이름 in 반복하고 싶은 것(문자열, 리스트, 튜플, 딕셔너리 등):  수행할 코드  ... |

for 키워드 뒤에 변수 이름을 임의로 설정하면 됩니다. 알아보기 쉬운 변수 이름으로 설정해주시면 됩니다. 그리고 in 키워드 뒤에 순차적으로 반복하고 싶은 것을 적으면 됩니다. 리스트 자료형 안에 있는 값을 for문을 사용해서 리스트 내에 있는 값을 하나씩 화면에 출력해보겠습니다.

예시 코드 for문을 활용하여 리스트 반복하기

|  |
| --- |
| box = ['one', 'two', 'three']  for i in box:  print(i) |

[실행 결과]

one

two

three

위의 결과를 보면, 리스트 내에 있는 값을 순서대로 변수 i에 할당한다는 뜻입니다. 가장 먼저 “one”이 변수 i에 할당되고, 그 다음 “two”, 마지막으로 “three”가 변수 i에 할당되고, 순차적으로 print() 함수로 인해 순서대로 출력됩니다. 리스트 뿐만 아니라, 문자열, 튜플, 딕셔너리 등 for문을 사용해서 각 자료형 내에 있는 값을 순서대로 반복할 수 있습니다. 이번에는 문자열에 있는 값을 순서대로 화면에 출력해보겠습니다.

예시 코드 for문을 활용하여 문자열 반복하기

|  |
| --- |
| for i in "Hello World!":  print(i) |

[실행 결과]

H

e

l

l

o

W

o

r

l

d

!

문자열을 for문을 활용해서 반복할 경우 문자열에 가장 앞에 있는 문자부터 순서대로 변수 i에 할당됩니다. (참고로 변수 i가 아닌 다른 변수 이름을 적어도 괜찮습니다. 필자 습관입니다.)

딕셔너리를 for문을 활용해서 딕셔너리 내에 있는 값을 반복하면 key값을 기준으로 반복합니다.

예시 코드 for문을 활용하여 딕셔너리 반복하기

|  |
| --- |
| dic = {"이름" : "홍길동", "나이" : 30, "성별" : "남자"}  for i in dic:  print(i) |

[실행 결과]

이름

나이

성별

2차원 리스트를 for문을 통해 반복하면 어떻게 될까요? 2차원 리스트란 리스트 안에 리스트 값들이 있는 형태를 말합니다. 이런 경우에도 똑같이 앞서 반복했던 것처럼 리스트 내에 있는 값이 순서대로 for문 변수에 들어갈 것입니다.

예시 코드 for문을 활용하여 2차원 리스트 반복하기

|  |
| --- |
| box = [[1,2], [3,4], [5,6]]  for i in box:  print(i) |

[실행 결과]

[1, 2]

[3, 4]

[5, 6]

이렇게 2차원 리스트고 리스트 내에 있는 값이 리스트로 2개씩 값이 들어 있다면, for문을 반복할 때 변수 이름을 2개 적을 수 있습니다.

예시 코드 for문에 변수 이름 2개 사용하기

|  |
| --- |
| box = [[1,2], [3,4], [5,6]]  for i,j in box:  print(i, j) |

[실행 결과]

1 2

3 4

5 6

만약 2차원 리스트 내에 있는 값이 리스트나 튜플로 3개 이상이었다면, 그 개수만큼 변수 이름을 늘릴 수 있습니다.

#### 범위 연산(range)

만약 for문을 이용해서 1부터 100까지 더해야 한다면 리스트 안에 숫자를 1부터 100까지 넣어야 할까요? 어떻게 넣을 수는 있겠지만 힘든 작업이 될 것입니다. 심지어 1부터 10,000까지 더해야 한다면 리스트에 이러한 값들을 넣는 것은 거의 불가능에 가까울 것입니다. 이런 경우에는 for문과 range() 함수를 함께 사용하면 쉽게 이 문제를 해결할 수 있습니다. range() 함수는 범위를 나타내는 파이썬에서 기본적으로 제공하는 내장함수입니다. range() 함수에는 1개에서 3개까지 매개변수를 넣을 수 있습니다. 기능은 아래와 같습니다.

1. range(n): 0부터 n-1까지
2. range(n1, n2): n1부터 n2-1까지
3. range(n1, n2, n3): n1부터 n2-1까지 n3씩 증가

간단히 range() 함수를 사용해서 0부터 9까지 출력해보겠습니다.

예시 코드 for문을 활용해서 0부터 9까지 출력하기

|  |
| --- |
| for i in range(10):  print(i) |

[실행 결과]

0

1

2

3

4

5

6

7

8

9

range() 함수에 숫자가 10 하나만 들어갔기 때문에 자동적으로 0부터 10전까지 변수 i에 순서대로 할당한다는 의미입니다.

이번에는 for문과 range() 함수를 활용해서 1에서 100까지 더한 값을 구해보겠습니다.

예시 코드 for문으로 1부터 100까지 합 구하기

|  |
| --- |
| total = 0 for i in range(1, 101):  total = total + i  print(total) |

[실행 결과]

5050

위 코드를 해석해보면, range() 함수가 1부터 101-1까지이기 때문에 1부터 100까지 순서대로 for문을 통해 반복한다는 의미입니다. 처음에 1이 변수 i에 들어가고 그 다음에는 2, 3, ... 100까지 순서대로 변수 i에 할당됩니다. 그리고 변수 total에 i를 더한 값을 다시 total에 누적해서 더하는 식으로 100번 반복합니다. 따라서 1부터 100까지 총 합 5050을 구할 수 있게 됩니다.

이번에는 1부터 10까지 3씩 증가시키면서 화면에 출력해보겠습니다.

예시 코드 for문을 활용해서 1부터 10까지 3씩 증가시켜 출력하기

|  |
| --- |
| for i in range(1, 11, 3):  print(i) |

[실행 결과]

1

4

7

10

위 코드를 해석해보면 1부터 11전까지 3씩 증가시키면서 변수 i에 숫자를 할당한다는 의미가 됩니다. 이렇게 for문과 range() 함수를 활용하면 쉽게 특정 범위를 순차적으로 실행시킬 수 있습니다.

## 함수

함수라는 것은 마치 공식과 같습니다. 만약 직각 삼각형이 있다고 가정해봅시다. 직각 삼각형의 빗변의 길이를 구하려면 어떻게 해야 할까요? 아마 대부분 직각 삼각형의 밑변의 길이를 제곱하고, 높이의 길이를 제곱해서 더한 후에 루트를 씌울 것입니다. 피타고라스 정리죠? 우리는 빗변의 길이를 구하기 위해서 피타고라스 공식을 바로 사용하지 피타고라스 정리를 증명한 후에 사용하지 않습니다. 왜냐하면 이미 중고등학생때 증명을 했으니까요. 그 후에는 빗변의 길이가 궁금할 때마다 피타고라스 공식에 입력 값인 밑변의 길이와 높이의 길이를 넣어준 후에 결과를 확인할 뿐입니다. 함수도 마찬가지입니다. 미리 만들어 놓고 필요할 때마다 가져다 쓰기 위해서 만드는 것이라고 이해해주시면 됩니다. 마치 아래 그림과 같이 어떤 함수에 입력 값을 넣어주면 원하는 결과 값이 나오는 기능인 것이죠.

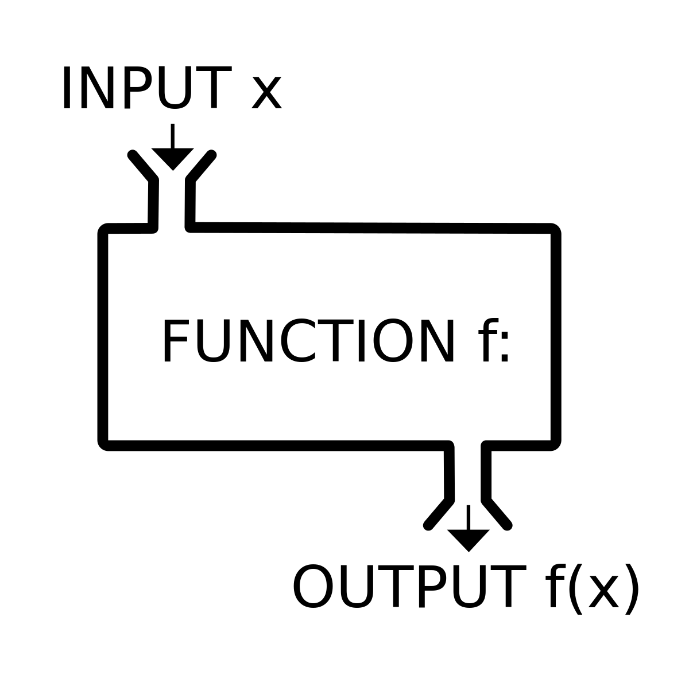


그림 함수

이미 우리는 다양한 함수들을 사용해봤습니다. 화면에 입력한 값을 출력해주는 print() 함수, 문자열이나 리스트 값의 개수를 세어주는 len() 함수 등 편리한 함수들을 사용해봤습니다. 하지만 우리가 원하는 특정 기능을 가진 함수를 파이썬에서 제공해주지 않는다면 어쩔 수 없이 우리가 직접 만들어야 합니다. 파이썬에서 함수를 만들 때는 def 키워드를 사용해주면 됩니다. 함수를 만드는 기본 구조는 아래와 같습니다.

예시 코드 함수의 기본 구조

|  |
| --- |
| def 함수이름(매개변수):  수행할 코드  ...  return 반환값 |

def 키워드 옆에 만들고자 하는 함수의 이름을 적어주고 매개변수가 자리에는 함수에 들어갈 입력 값을 입력해주면 됩니다. 그 함수 내에서 계산을 하고 그 결과값을 return 키워드를 통해 반환해주면 됩니다. def 키워드를 사용해서 아주 간단한 함수를 하나 만들어보겠습니다. 두 개의 입력 값이 들어가면 그 두 개의 값을 더해서 더해진 값을 반환해주는 add() 함수를 만들어보겠습니다.

예시 코드 두 개의 입력 값을 더해서 반환해주는 add() 함수 만들기

|  |
| --- |
| def add(num1, num2):  return num1 + num2  print(add(3, 4)) |

[실행 결과]

7

### 입력 값이 없는 함수

def 키워드를 통해 add() 함수를 만들고 add() 함수에는 두 개의 매개변수 num1, num2가 들어가야 합니다. 두 개의 매개변수가 들어가면 두 개의 매개변수를 모두 더한 값을 반환해줍니다. 만든 add() 함수에 숫자 3, 4를 넣으면 add() 함수가 실행되면서 3은 num1에 할당되고, 4는 num2에 할당이 됩니다. 따라서 7이 반환됩니다.

함수를 만들 때 꼭 입력 값이 들어갈 필요는 없습니다.

예시 코드 입력값이 없는 함수

|  |
| --- |
| def say\_hello():  return "Hello!"  print(say\_hello()) |

[실행 결과]

Hello!

위 코드에서 만든 say\_hello() 함수는 아무런 입력 값이 들어가지 않습니다. 아무런 입력 값이 들어가지 않아도 say\_hello() 함수를 실행하면 “Hello” 문자열을 반환합니다. 이렇게 함수를 만들 때 입력 값이 없는 함수를 만들 수도 있습니다.

반대로 반환 값이 없는 함수를 만들 수도 있습니다. 앞서 만들었던 add() 함수를 반환 값이 없는 함수로 바꾸어 만들어보겠습니다.

### 반환 값이 없는 함수

예시 코드 반환 값이 없는 함수

|  |
| --- |
| def add(num1, num2):  print(num1 + num2)  print(add(2, 5)) |

[실행 결과]

7

None

위 코드를 실행해보면 결과가 약간 이상합니다. add() 함수 안에 있는 print() 함수 덕분에 숫자 7이 하나만 화면에 출력되어야 하는데, None도 같이 출력된 것을 알 수 있습니다. 여기서 None이란 아무것도 없는 값을 의미합니다. 이것은 무슨 의미일까요?? add() 함수의 결과 값이 아무것도 없다는 의미입니다. 결과 값처럼 보이는 두 개의 입력 값을 더해져 출력된 숫자 7은 print() 함수 때문에 나온 연산 결과 값이지 add() 함수의 반한 값이 아닙니다.

### 입력 값, 반환 값이 없는 함수

입력 값도 없고 반환 값도 없는 함수를 만들 수도 있습니다. 앞서 만들었던 say\_hello() 함수를 입력 값도 반환 값도 없는 함수로 바꾸어 만들어보도록 하겠습니다.

예시 코드 입력 값, 반환 값없는 함수

|  |
| --- |
| def say\_hello():  print("Hello!")  print(say\_hello()) |

[실행 결과]

Hello!

None

### 여러 개의 입력 값을 받는 함수

우리가 앞서 만들었던 매개변수가 있는 함수들은 매개변수의 개수가 정해져 있었습니다. 만약 함수를 만들 때 매개변수가 2개였다면, 그 함수를 실행하기 위해서는 2개의 입력 값을 넣어야만 하죠. 만약 2개의 매개변수가 들어가야하는 함수에 2개가 아닌 입력 값을 넣으면 에러가 발생합니다. 함수를 사용할 때마다 들어가야하는 매개변수의 개수가 다르다면 이 문제를 어떻게 해결할 수 있을까요? 함수 내 매개변수에 “\*” 모양을 앞에 붙여주면 됩니다.

예시 코드 여러 개의 입력 값을 받는 함수

|  |
| --- |
| def many\_var(\*var):  return var  print(many\_var(1,3,5,4,2)) print(many\_var(1,2,3)) |

[실행 결과]

(1, 3, 5, 4, 2)

(1, 2, 3)

many\_var() 함수에는 var이라는 매개변수가 들어가 있습니다. 마치 하나의 입력 값을 받는 것 가죠? 하지만 many\_var() 함수 안에 여러 개의 입력 값을 입력해봐도 에러가 발생하지 않습니다. 여러 개의 입력 값을 넣으면 튜플로 강제로 묶어주는 것을 볼 수 있습니다. 왜냐하면 many\_var() 함수에 들어가는 매개변수 var 앞에는 “\*” 모양이 붙어있습니다. 이것의 의미는 var변수에는 여러 개의 값이 들어갈 수 있으니 여러 개의 값이 들어오면 하나로 묶어 달라는 의미입니다.

하나의 입력 값과 여러 개의 값을 받는 식으로 함수를 만들 수도 있습니다.

예시 코드 하나의 값과 여러 개의 입력 값을 받는 함수

|  |
| --- |
| def many\_var(num, \*var):  return [num, var]  print(many\_var(1,3,5,4,2)) print(many\_var(1,2,3)) |

[실행 결과]

[1, (3, 5, 4, 2)]

[1, (2, 3)]

이번에는 매개변수에 num과 var이 들어가 있습니다. 여기서 매개변수 var 앞에만 “\*” 모양이 붙어있습니다. 이렇게 만들어진 many\_var() 함수에 여러 개의 입력 값이 들어가면 맨 앞에 있는 값이 num변수에 들어가며, 나머지는 튜플로 묶인 채 var변수에 들어가게 됩니다.

### 기본 값을 가지는 함수

함수를 만들 때, 함수 안에 들어가는 매개변수에 기본 값을 설정할 수 있습니다. 이름, 나이, 성별을 입력하면 자기소개를 해주는 intro() 함수를 만들어보겠습니다.

예시 코드 기본 값을 가지는 함수

|  |
| --- |
| def intro(name, age, sex = 1):  print("나의 이름은 {}이고, 나이는 {}살입니다.".format(name, age))  if sex == 1:  print("나는 남자입니다.")  else:  print("나는 여자입니다.")  intro("홍길동", 30) intro("장원영", 19, 3) |

[실행 결과]

나의 이름은 홍길동이고, 나이는 30살입니다.

나는 남자입니다.

나의 이름은 장원영이고, 나이는 19살입니다.

나는 여자입니다.

위 코드에서 만든 intro() 함수는 3개의 매개변수를 입력 받습니다. 하지만 intro(“홍길동”, 30) 코드를 실행할 때는 매개변수를 2개만 입력하는 것을 볼 수 있습니다. 그래도 에러가 발생하지 않습니다. 왜냐하면 intro() 함수 안에 입력할 name, age, sex 매개변수 중 sex 매개변수에는 1이 할당 되어있는 것을 볼 수 있습니다. 이렇게 미리 함수 안에 매개변수에 어떤 값을 할당해 놓으면 나중에 함수를 실행할 때, 미리 할당된 값을 입력하지 않으면 자동으로 미리 할당된 값을 가진다는 의미입니다. 만약에 미리 할당된 값을 바꾸려면 함수를 실행할 때 기본 값으로 미리 할당된 값에도 입력 값을 넣어주면 바꿀 수 있습니다.

### 전역변수와 지역변수

함수를 사용할 때 주의해야할 점이 있습니다. 바로 변수인데요. 함수 안에서 사용되는 변수는 함수 안에서만 영향을 미친다는 사실에 유의해야합니다. 예시를 하나 보도록 하겠습니다.

예시 코드 지역변수와 전역변수

|  |
| --- |
| def my\_func():  a = 5   a = 0 my\_func() print(a) |

[실행 결과]

0

위 코드를 보면, 처음에 변수 a에 0을 할당하고 있습니다. 그리고 my\_func() 함수를 실행합니다. my\_func() 함수는 변수 a에 5를 넣는 함수이기 때문에 변수 a에는 5가 들어갔을 겁니다. 그리고 변수 a를 출력했는데 5가 나오는 것이 아닌 0이 나옵니다. 왜 이럴까요? 우리 눈에는 my\_func() 함수 안에 있는 변수 a와 함수 밖에 있는 변수 a가 같은 변수처럼 보입니다. 하지만 아닙니다. 이해를 돕기 위해 쉽게 설명하자면 변수 a와 my\_func() 함수 안에 있는 a로 구분되어 있다고 생각해주시면 됩니다. 앞서 이야기했지만 함수 안에서 사용되는 변수는 함수 안에서만 영향을 미칩니다. 그래서 함수 안에 있는 변수는 지역적으로 영향을 미친다고 해서 “지역변수” 라고 부릅니다. 그리고 함수 밖에 있는 변수는 전역에 영향을 미친다고 해서 “전역변수” 라고 부릅니다. 만약 함수 밖에 있는 변수와 함수 안에 있는 변수를 같은 변수로 취급하게 하려면 어떻게 할 수 있을까요? global 키워드를 사용하면 이 문제를 해결할 수 있습니다.

예시 코드 지역변수 전역변수로 만들기

|  |
| --- |
| def my\_func():  global a  a = 5  a = 0 my\_func() print(a) |

[실행 결과]

5

함수 안에 global 키워드를 적고 옆에 전역변수 취급을 할 변수 이름을 적어주면 됩니다. global 키워드는 함수에게 특정 변수는 지역변수가 아니라 전역변수라고 명시해주는 키워드입니다. global 키워드를 사용하면 함수 내에서 사용되는 특정 변수를 전역변수로 취급해서 사용할 수 있습니다. global 키워드를 사용해서 my\_func() 함수 안에서 사용되는 변수 a를 전역변수로 취급했고, 함수 안에서 변수 a에 5를 할당했으니 전역변수 a에 5를 할당한 것입니다. 그래서 결과로 5가 화면에 잘 출력되는 것을 볼 수 있습니다.

### lambda

함수를 만들 때는 def 키워드를 사용해서 함수를 만듭니다. 그런데 lambda 키워드를 사용해서 함수를 만들 수도 있습니다. 사실 잘 몰라도 되는 내용이긴 하지만 알아 두면 편합니다. 아주 간단한 함수를 즉석해서 만들 때는 lambda 키워드를 사용해서 만들기도 합니다. 우리가 앞서 만들었던 add() 함수를 lambda 키워드를 활용해서 만들어보도록 하겠습니다.

예시 코드 lambda 키워드로 함수 만들기

|  |
| --- |
| add = lambda num1, num2 : num1 + num2 print(add(4,5)) |

[실행 결과]

9

lambda 키워드 뒤에 콜론을 기준으로 왼쪽에는 들어갈 입력 값 오른쪽에는 반환 값을 입력해주고 지정하고자 하는 함수 이름에 할당해주면 쉽게 함수를 만들 수 있습니다. 지금은 “굳이 def 키워드를 사용해서 함수를 만들면 되는데 왜 lambda 키워드를 사용해서 함수를 만들지?” 라고 생각하시는 분들이 많을 수 있습니다. 맞는 말 이긴 하지만 프로그래밍에 익숙해지다 보면 어느새 lambda의 편리성의 매력을 느끼실 때가 올 것입니다.

## 클래스

클래스는 파이썬 기초 문법 중 가장 어려운 개념일 것이라고 생각합니다. 클래스는 객체를 생성하는 기능을 합니다. 클래스를 쉽게 설명하자면 함수와 변수들을 모아 놓은 주머니 같은 것이라고 생각하면 됩니다. 즉, 객체를 생성하는 클래스가 객체를 생성하면 그 객체는 클래스 안에 있는 함수들과 변수들을 자유롭게 사용할 수 있게 되는 것이지요. 클래스를 만들기 위해서는 class 키워드를 사용해서 만들면 됩니다. 은행에 있는 ATM기기는 다양한 기능을 하지만 중요한 것은 입금과 출금의 기능일 겁니다. 입금해주는 기능을 가진 ATM기기 프로그램을 클래스를 통해서 구현해보도록 하겠습니다.

예시 코드 입금 기능이 있는 Bank 클래스

|  |
| --- |
| class Bank:  def deposit(self, var):  self.money = var |

Bank 라는 이름의 클래스를 만들고, Bank 클래스 안에 입금 기능을 하는 deposit() 함수를 하나 만들었습니다. 여기서 중요한 점은 클래스 안에서 함수를 만들 때 첫번째 매개변수에는 항상 self가 들어가야 한다는 점입니다. 가장 먼저 self를 적고 두번째 매개변수부터 넣고자 하는 매개변수 이름을 입력해주면 됩니다. 그리고 deposit() 함수 안에는 self.money라는 변수에 매개변수로 입력한 var값을 할당하고 있습니다. self에 대한 조금 더 자세한 내용은 뒤에서 다루도록 하겠습니다. 자 이제 클래스를 만들었으니 클래스를 통해서 객체를 하나 만들어보도록 하겠습니다.

예시 코드 Bank 클래스로 person1 객체 생성하기

|  |
| --- |
| class Bank:  def deposit(self, var):  self.money = var   person1 = Bank() person1.deposit(50000) print(person1.money) |

[실행 결과]

50000

위 코드를 해석해보겠습니다. person1에 Bank 클래스를 할당했습니다. 이 의미는 Bank 클래스로 person1이라는 객체를 생성한 것이며, person1 객체는 Bank 클래스 안에 있는 deposit() 함수를 자유롭게 쓸 수 있는 권한을 받은 것입니다. person1 객체에서 deposit() 함수를 사용하려면 객체 이름 뒤에 점(.)을 찍고 클래스 내에 있는 함수 이름을 적어주면 됩니다. person1.deposit(50000)을 실행하게 되면, 입력한 50000은 person1 객체 안에 있는 deposit() 함수 안에 var변수로 입력이 됩니다. var 변수에 50000이 할당되었고, 변수 var의 값은 다시 self.money에 할당됩니다. 그래서 person1.money를 화면에 출력하면 50000이 출력되는 것이죠. 그런데 조금 이상하지 않나요? self.money에 50000을 할당했는데 self.money를 출력한 것이 아닌 person1.money를 출력했는데 50000이 화면에 출력됩니다. 아마 눈치채신 분들이 있을 것입니다. 여기서 self란 객체의 이름이 들어갈 빈칸 같은 것이라고 보시면 됩니다. 처음에 person1변수에 Bank 클래스를 할당했을 때, person1 변수에게는 모든 self가 person1으로 취급을 하게 됩니다. 예시를 하나 더 보겠습니다.

예시 코드 Bank 클래스로 객체 여러 개 생성하기

|  |
| --- |
| class Bank:  def deposit(self, var):  self.money = var  person1 = Bank() person2 = Bank()  person1.deposit(50000) person2.deposit(30000)  print(person1.money) print(person2.money) |

[실행 결과]

50000

30000

이번에는 2개의 객체(person1, person2)를 만들었습니다. person1 객체는 모든 self가 person1으로 보일 것입니다. 마찬가지로 person2는 모든 self가 person2라고 보입니다. 각각 독립된 객체가 생성된 것입니다. person1이 어떤 행동을 한다고 해서 person2가 영향을 받지 않습니다. person1 객체가 deposit() 함수를 사용해서 50000을 넣고, person2가 deposit() 함수를 사용해서 30000을 넣었습니다. 그러면 person1.money에는 50000이 할당되고, person2.money에는 30000이 할당되는 것이죠. 그래서 출력 결과가 위와 같은 것입니다.

만약 이렇게 코드를 작성해서 프로그램을 실행하면 어떨까요?

예시 코드 클래스 오류 발생 예시

|  |
| --- |
| class Bank:  def deposit(self, var):  self.money = var  person1 = Bank() print(person1.money) |

위 코드를 실행하면 에러가 발생합니다. 왜냐하면 person1 객체를 생성했지만 돈을 입금(deposit)하지 않았기 때문에 person1.money를 실행하면 에러가 발생하는 것이지요. 그런데 우리가 사용하는 은행 ATM기기가 우리가 계좌 개설을 하고 입금한 적은 없지만 잔액 조회 버튼을 눌렀다고 에러가 발생하나요? 발생하지 않습니다. 0원이라고 조회가 되겠죠? 하지만 우리가 만든 Bank 클래스는 입금을 하지 않으면 잔액 조회를 하면 에러가 발생하고 있습니다. 이 부분을 해결하기 위해서 Bank 클래스에 함수를 하나 더 추가해보겠습니다.

### 초기화 함수(\_\_init\_\_)

Bank 클래스에서 함수를 만들 때 우리가 원하는 이름으로 함수 이름을 정의해도 되지만 어느 함수는 이름을 꼭 정해준 이름대로 만들어야합니다. Bank 클래스에 \_\_init\_\_() 함수를 만들어보도록 하겠습니다. \_\_init\_\_() 함수는 초기화 함수라고 하는데 객체를 생성할 때, 우리가 이 함수를 실행하지 않아도 자동으로 1번 자동으로 실행되는 함수입니다.

예시 코드 \_\_init\_\_() 함수

|  |
| --- |
| class Bank:   def \_\_init\_\_(self):  print("계좌 개설을 축하합니다!")  self.money = 0   def deposit(self, var):  self.money = var  person1 = Bank() print(person1.money) |

[실행 결과]

계좌 개설을 축하합니다!

0

person1 객체를 생성하는 순간 우리는 \_\_init\_\_() 함수를 실행하지도 않았는데 “계좌 개설을 축하합니다!” 라는 문장이 화면에 출력되는 것을 보니 \_\_init\_\_() 함수가 자동으로 한번 실행된 것을 알 수 있습니다. 그리고 self.money에 0을 할당했기 때문에 우리가 따로 deposit() 함수를 실행하지 않아도 바로 person1.money를 출력하면 에러가 발생하지 않고 0이 잘 출력되는 것이지요.

사실 파이썬은 거의 대부분 자료형이 클래스로 이루어져 있습니다. 예를 들어 a = [1,3,5,4,2]라는 코드가 있습니다. 변수 a에는 리스트 자료형 [1,3,5,4,2]가 들어가 있죠. 그러면 a는 리스트 자료형에서 제공하고 있는 append(), sort(), remove() 함수 등 다양한 함수를 사용할 수 있는 권한을 가집니다. 리스트도 클래스였던 것입니다.

### 상속

우리가 만든 Bank 클래스가 너무 인기가 많아서 Bank2 클래스를 하나 더 만든다고 가정해보겠습니다. Bank2 클래스 역시 은행 ATM의 역할을 하는 프로그램이기 때문에 입금하는 기능을 넣어주어야 합니다. 그런데 Bank 클래스에서 이미 만들었는데, 또 만들 필요가 있을까요? 단순히 Bank 클래스에서 이미 만든 deposit() 함수를 가져오는 방법은 없을까요? 있습니다. 상속의 기능을 사용하면 됩니다. Bank 클래스에 있는 기능을 Bank2에게 상속해주는 것이죠. 클래스를 상속할 때는 클래스를 생성할 때 소괄호를 통해서 상속해올 클래스 명을 입력해주면 됩니다.

예시 코드 Bank 클래스를 상속받은 Bank2 클래스 만들기

|  |
| --- |
| class Bank:   def \_\_init\_\_(self):  print("계좌 개설을 축하합니다!")  self.money = 0   def deposit(self, var):  self.money = var  class Bank2(Bank):  pass  person = Bank2() person.deposit(30000) print(person.money) |

[실행 결과]

계좌 개설을 축하합니다!

30000

실행 결과를 보면 분명 person 객체는 Bank2에 의해서 만들어졌습니다. 그런데 Bank2 클래스 안에는 아무것도 없습니다. pass라는 키워드만 있죠. (여기서 pass 키워드는 적을 것이 없지만 무엇이라도 적어야 할 때 사용하는 키워드입니다.) 그런데도 불구하고 person 객체에서 Bank 클래스 안에 있는 \_\_init\_\_() 함수도 실행이 되었고, person.money도 잘 출력이 됩니다. 왜냐하면 Bank2 클래스는 Bank 클래스에 있는 모든 기능을 상속받아왔기 때문에 이것이 가능했던 것입니다.

클래스에 대한 개념은 상당히 깊어서 클래스 개념 하나만 다룬다고 하더라도 거의 책 한권이 나올 정도로 내용이 깊습니다. 하지만 우리는 클래스에 대해서 그렇게 깊게 공부할 필요는 없습니다. 나중에 프로그래밍에 대해서 충분히 익숙해졌을 때 클래스에 대해서 깊게 공부해도 늦지 않습니다. “클래스는 함수와 변수들을 모아 놓은 주머니 같은 개념이구나” 하는 정도와 “클래스로 객체를 만들 수가 있는데 만들어진 객체는 해당 클래스 안에 있는 함수들을 자유롭게 쓸 수 있구나” 정도로만 이해하셔도 충분합니다!

## 예외처리

코드가 실행이 되는 도중 에러가 발생하면 즉시 코드를 멈추고 에러를 발생시킵니다. 예외 처리라는 것은 에러가 발생했을 때, 에러를 무시하는 기술입니다. 사실 예외 처리를 남용해서 사용하면 좋지 않습니다. 에러를 발생했을 때 가장 좋은 방법은 에러가 안 나게 코드를 고치는 것입니다. 그런데 예외 처리가 필요한 이유는 아무런 잘못된 것이 없는데 에러가 발생하는 경우입니다. 이런 경우에는 에러를 무시하는 것도 좋은 방법입니다. 예를 들어 키움 증권 API를 가져와서 사용하는데 API에 문제가 있어서 에러가 발생하는 경우 또는 어떤 블로그에 있는 내용을 크롤링하려고 했는데, 블로그에 문제가 있어서 에러가 발생하는 경우입니다. 사실 다른 곳에 문제가 있지 코드에는 문제가 없는 경우에는 예외 처리를 사용하면 좋습니다.

### try – except

예외 처리는 try와 except라는 키워드를 사용해서 만듭니다. 간단하게 예시로 알아보겠습니다.

예시 코드 예외 처리 기본 구조

|  |
| --- |
| try:  수행할 코드  ... except:  수행할 코드  ... |

try 키워드 안에서 에러가 발생하면 에러를 발생시키지 않고 except 키워드 안에 있는 코드를 수행합니다. 간단하게 에러를 발생시켜보겠습니다.

예시 코드 예외 처리 예시

|  |
| --- |
| try:  5 / 0 except:  print("분모가 0입니다!") |

[실행 결과]

분모가 0입니다!

연산에서 분모가 0이면 부정이라고 해서 있을 수 없는 연산입니다. 따라서 분모가 0일 경우 프로그래밍에서도 마찬가지로 에러가 발생합니다. 하지만 try 키워드 안에서 에러가 발생했으므로 에러가 발생하지 않고 자연스럽게 except 키워드 내에 있는 코드를 수행하고 끝이납니다.

### 에러 발생시키기

반대로 에러를 일부러 발생시키는 raise 키워드도 있습니다. raise 키워드 옆에 에러 내용을 적어주면 일부러 해당 에러를 발생시킬 수 있습니다.

예시 코드 에러 발생시키기(raise)

|  |
| --- |
| raise 에러 이름 |

## 내장함수

우리는 앞서 print() 함수, len() 함수 등 몇 가지 파이썬에서 기본적으로 제공하는 내장 함수를 사용했습니다. 내장 함수도 종류가 많기 때문에 모든 내장 함수를 소개하기에는 무리가 있습니다. (자세한 내장 함수를 찾는다면 구글에 “파이썬 내장 함수”를 검색해보세요.) 우리가 앞으로 프로그래밍을 하면서 알아 두면 좋은 몇 가지 내장 함수를 소개해보겠습니다. 또한 앞서 설명 드렸던 내장 함수는 건너뛰겠습니다.

### 절대 값 반환: abs

abs() 함수는 안에 숫자 값을 넣어주면 절대 값으로 반환해주는 함수입니다.

예시 코드 abs() 함수

|  |
| --- |
| print(abs(-5)) print(abs(5)) print(abs(-3.14)) |

[실행 결과]

5

5

3.14

### 모든 값이 참인지 확인: all

all() 함수는 리스트, 튜플, 문자열, 딕셔너리 등과 같은 자료형 안에 있는 값들이 모두 참이면 참을 반환해주는 함수입니다.

예시 코드 all() 함수

|  |
| --- |
| print(all([1,3,5,4,2])) print(all([1,3,5,4,0])) |

[실행 결과]

True

False

### 참이 존재하는지 확인: any

all() 함수와 반대로 리스트, 튜플, 문자열, 딕셔너리 등과 같은 자료형 안에 있는 값들 중 단 한개의 참이라도 있으면 참을 반환해주는 함수입니다.

예시 코드 any() 함수

|  |
| --- |
| print(any([0,0,0,1])) print(any([0,0,0,0])) |

[실행 결과]

True

False

### 값의 위치와 값을 반환: enumerate

enumerate() 함수는 리스트, 튜플, 문자열같은 순서가 있는 자료형을 입력 값으로 넣어주면 순서와 해당 자료형 안에 있는 값들을 반환해줍니다. enumerate() 함수는 주로 for문과 함께 많이 사용됩니다.

예시 코드 enumerate() 함수

|  |
| --- |
| box = ['장원영', '강동원', '홍길동']  for i in enumerate(box):  print(i) |

[실행 결과]

(0, '장원영')

(1, '강동원')

(2, '홍길동')

### 사용자의 입력 값을 받음: input

input() 함수는 사용자의 입력을 실시간으로 받는 함수입니다. 다만 사용자의 입력 값을 항상 문자열로 받는다는 것에 유의해야합니다.

예시 코드 input() 함수

|  |
| --- |
| a = input("값을 입력해주세요 : ") print(a) |

[실행 결과]

값을 입력해주세요 : python

python

### 정수 값으로 변환: int

int() 함수는 integer의 줄임 말로 입력 받은 값을 정수로 변환해주는 함수입니다. 문자로 된 숫자를 정수로 바꿀 수도 있고, 실수형태의 숫자를 정수 형태로 바꾸기도 합니다.

예시 코드 int() 함수

|  |
| --- |
| a = "5" print(int(a))  b = 3.14 print(int(b)) |

[실행 결과]

5

3

### 최대값과 최소값: max, min

max(), min() 함수는 리스트, 튜플, 문자열 등과 같은 자료형에서 최댓값, 최소값을 반환해주는 함수입니다.

예시 코드 min(), max() 함수

|  |
| --- |
| print(max([1,3,5,4,2])) print(min([1,3,5,4,2]))  print(max("Hello")) print(min("Hello")) |

[실행 결과]

5

1

o

H

### 파일 생성: open

open() 함수는 파일을 생성할 때 사용하는 함수입니다. 데이터 분석이나 머신러닝 인공지능 모델을 만들면 그 결과를 따로 파일로 저장해야 하는 경우가 있습니다. 이럴 때 open() 함수를 통해서 프로그래밍 결과물을 파일로 저장할 수 있습니다. 간단하게 메모장 파일을 만들어보겠습니다. (메모장 파일의 확장자는 txt입니다.)

예시 코드 open() 함수로 파일 저장하기

|  |
| --- |
| f = open("my\_text.txt", 'wt') f.write("Hello World!") f.close() |

[실행 결과]

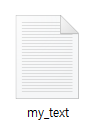


그림 open() 함수로 인해 생성된 파일

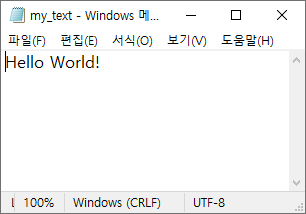


그림 생성된 my\_text.txt 파일 안 내용

open() 함수에 생성하고자 하는 파일 이름과 옵션 값을 적어주면 됩니다. 옵션 값은 크게 “w”와 ”r”이 있습니다. “w”는 write의 약자이며, “r”은 read의 약자입니다. 그리고 위 코드에서 적은 “wt”는 write text의 약자로 text로 쓰겠다는 의미입니다. 즉 해석해보면 my\_text.txt 파일을 하나 쓰기 위해서 열었다는 의미이고, 그 파일은 변수 f에 할당하겠다는 의미입니다. 변수 f가 메모장 그 자체인 셈이지요. 그 변수 f에 write() 함수를 적용해서 그 파일 안에 적을 내용을 입력할 수 있습니다. 그리고 f.close()은 파일을 저장하고 끄겠다는 의미입니다. close() 함수를 꼭 적용해 주어야만 저장이 되니 꼭 파일을 닫는 것까지 해주서 야합니다. 위 코드를 실행하면 my\_text.txt 파일이 코드를 실행하고 있는 파일이 있는 같은 폴더에 아래 그림과 같이 생성됩니다. 그리고 파일 안에는 아래와 같이 우리가 앞서 적었던 “Hello World!”가 적혀 있습니다.

저장했던 파일을 불러오려면 open() 함수를 똑같이 사용하면 됩니다. 다만 옵션 값에는 “rt”라고 적어 주셔야 합니다. “rt”는 read text의 약자입니다.

예시 코드 open() 함수로 파일 불러오기

|  |
| --- |
| f = open("my\_text.txt", 'rt') data = f.read() f.close()  print(data) |

[실행 결과]

Hello World!

my\_text.txt 파일을 “rt”로 텍스트를 읽기 위해서 열었다 라는 식으로 이해해주시면 됩니다. 연 파일은 변수 f에 할당했습니다. 다만 읽은 것은 아니고 읽기 위해서 파일을 연 것이기 때문에 read() 함수를 적용해서 파일 안에 있는 내용을 읽어 주셔야 합니다.

### 반올림 값 반환: round

round() 함수는 입력 받은 값을 반올림해주는 함수입니다. 2번째 매개변수도 입력해주면 소수점 몇 번째자리까지 표현할 것인지 정할 수도 있습니다.

예시 코드 round() 함수

|  |
| --- |
| print(round(4.6)) print(round(4.2)) print(round(3.14, 1)) |

[실행 결과]

5

4

3.1

### 문자열로 변환: str

str() 함수는 입력 받은 값을 문자열로 반환해주는 함수입니다.

예시 코드 str() 함수

|  |
| --- |
| print(str(5)) |

[실행 결과]

5

출력 결과를 보면 숫자 5인 것처럼 보일 수 있지만 숫자 5가 문자로 5로 변환된 것입니다.

### 숫자 값들의 총합: sum

sum() 함수는 리스트나 튜플 안에 있는 값을 모두 더해주는 함수입니다.

예시 코드 sum() 함수

|  |
| --- |
| box = [1,3,5,4,2] print(sum(box)) |

[실행 결과]

15

### 같은 위치에 있는 값 묶기: zip

zip() 함수는 2개 이상의 리스트의 같은 위치에 있는 값들을 묶어주는 함수입니다. zip() 함수 역시 for문을 사용할 때 함께 자주 사용합니다.

예시 코드 zip() 함수

|  |
| --- |
| box1 = [1,2,3] box2 = ['가','나','다']  for i, j in zip(box1, box2):  print(i, j) |

[실행 결과]

1 가

2 나

3 다

실행 결과를 보면 box1, box2의 0번째 값이 먼저 변수 i와 j에 들어가고 그 다음 1번째, 그 다음 2번째 순으로 순차적으로 각 리스트에 있는 위치의 값을 가져올 수 있습니다.

## 라이브러리

라이브러리는 다른 사람들이 만들어 놓은 프로그램(모듈)들이라고 보시면 됩니다. 프로그램 각각을 모듈이라고 합니다. 내가 만약 무언가를 개발해야 하는데, 개발하고자 하는 프로그램을 누군가가 만들어서 무료로 공유해 놓고 있다면 개발 시간이 훨씬 단축되겠죠? 파이썬에서는 다른 사람들이 만들어 놓은 프로그램을 아주 쉽게 가져올 수 있습니다. 다른 사람들이 만들어 놓은 프로그램을 가져오려면 import 키워드를 사용하면 아주 쉽게 다른 사람이 만든 프로그램을 가져올 수 있습니다. 라이브러리도 4만개 이상 존재하기 때문에 모든 라이브러리를 소개하기에는 무리가 있습니다. (필자도 자주 사용하는 라이브러리가 30~40개 정도 밖에 되지 않습니다.) 필요한 라이브러리가 있다면 구글에 필요한 라이브러리가 있는지 먼저 검색해보고, 없다면 그 때 필요한 프로그램을 개발하는 것이 좋습니다. 앞으로 데이터 분석과 머신러닝 등을 진행하면서 필요한 라이브러리는 그 때마다 소개하도록 하겠습니다. 먼저 알아 두면 좋은 라이브러리 모듈 몇 개를 소개하면서 라이브러리를 불러오는 방법을 익혀보도록 하겠습니다.

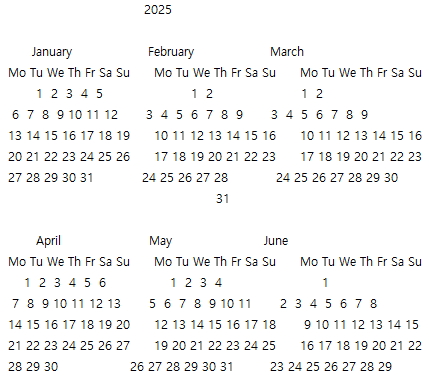
### 모듈 불러오기

가장 먼저 소개할 라이브러리는 calendar 모듈입니다. 만약 달력 프로그램을 개발해야 한다면 꽤 번거로운 작업이 될 것입니다. 시간이 많이 소요되겠죠. 달력에는 어려운 수학적 내용들이 많이 들어가기 때문입니다. 그런데 누군가가 완벽하게 만들어 놓은 달력 프로그램이 있다면 개발 시간이 매우 단축시킬 수 있습니다. calendar 모듈 안에는 달력과 관련된 많은 프로그램(모듈 또는 함수)들이 있습니다. calendar 모듈 안에는 특정 년도의 달력을 보여주는 calendar() 함수가 있습니다. 2025년도 달력을 calendar 라이브러리로부터 구해보도록 하겠습니다.

예시 코드 calendar - calendar() 함수

|  |
| --- |
| import calendar print(calendar.calendar(2025)) |

[실행 결과]



calendar 모듈 안에 calendar() 함수에 특정 년도를 입력하니 해당 년도가 예쁘게 출력되었습니다. 단 2줄로 달력을 그릴 수 있게 된 것입니다.

calendar 모듈 안에 weekday() 함수를 통해 특정 일이 무슨 요일인지도 알 수 있습니다.

예시 코드 calendar - weekday() 함수

|  |
| --- |
| import calendar print(calendar.weekday(2023, 5, 24)) |

[실행 결과]

2

2023년 5월 24일을 weekday() 함수에 입력하니 숫자 2가 출력되었습니다. 참고로 0이 월요일입니다. 따라서 2023년 5월 24일은 수요일인 것을 알 수 있습니다.

이번에는 time 모듈을 소개해보도록 하겠습니다. time 모듈은 시간에 대한 다양한 함수들을 제공합니다. time 모듈로부터 현재 시간을 문자열로 출력해주는 ctime() 함수를 사용해보겠습니다.

예시 코드 time - ctime() 함수

|  |
| --- |
| import time print(time.ctime()) |

[실행 결과]

Sun Dec 18 20:18:39 2022

time 모듈에서 가장 많이 사용되는 함수는 sleep() 함수 일 것입니다. sleep() 함수는 강제로 프로그램을 입력한 초만큼 멈추게 하는 함수입니다. sleep() 함수를 사용해서 1부터 10까지 1초씩 쉬면서 출력해보도록 하겠습니다.

예시 코드 time - sleep() 함수

|  |
| --- |
| import time  for i in range(1, 11):  print(i)  time.sleep(1) |

[실행 결과]

1

2

3

4

5

6

7

8

9

10

위 코드를 실행해보면 1부터 10까지 1초씩 쉬면서 화면에 숫자를 출력하는 것을 볼 수 있습니다. sleep() 함수는 특히 데이터를 크롤링할 때 많이 사용됩니다. 왜냐하면 데이터를 너무 빠르게 가져오게 되면 다른 사람의 서버에 큰 무리를 줄 수 있기 때문입니다.

### 모듈 일부만 가져오기(from)

그런데 만약 time 모듈로부터 sleep() 함수 하나만 사용한다고 가정해보겠습니다. import 키워드를 통해 time 모듈을 가져오게 되면 sleep() 함수 뿐만 아니라 time 라이브러리 안에 있는 모든 프로그램들을 다 가져오게 됩니다. 그러면 용량 낭비일 수 있습니다. 만약 time 모듈 안에 있는 수많은 모듈 중 sleep() 함수 하나만 가져오려면 어떻게 해야 할까요? 모듈을 불러올 때 from 키워드를 함께 사용해서 가져오면 됩니다.

예시 코드 from 키워드를 사용해서 특정 모듈만 가져오기

|  |
| --- |
| from time import sleep  for i in range(1, 11):  print(i)  sleep(1) |

[실행 결과]

1

2

3

4

5

6

7

8

9

10

from 키워드 옆에 가져오고자 하는 모듈명을 적어주고, import 키워드 옆에 사용하고자 하는 모듈명 또는 함수 이름을 적어 주시면 됩니다. 단 특정 모듈 하나만 가져왔기 때문에 사용할 때는 특정 모듈 이름을 적어서 사용해 주셔야 합니다.

### 모듈 이름 바꾸기(as)

만약 불러온 모듈명이 너무 길어서 쓰기 불편하다면 어떨까요? 그럴 때는 as 키워드를 사용해서 쉬운 이름으로 바꾸어 주면 됩니다. 앞서 가져온 sleep() 함수 이름을 sp라는 이름으로 가져오거나 가져온 time 모듈을 tm이라는 이름으로 사용해보겠습니다.

예시 코드 as 키워드를 사용해서 라이브러리명 바꾸기

|  |
| --- |
| from time import sleep as sp import time as tm |

라이브러리는 우리가 앞으로 자주 사용하게 될 것입니다. 다양한 모듈 소개는 데이터 분석과 데이터 크롤링, 머신러닝을 배울 때 관련 라이브러리를 소개하면서 진행하도록 하겠습니다. 라이브러리 사용 방법을 정리해보면 아래와 같습니다.

1. import 모듈명 [as 이름]
2. from 모듈명(전체) import 서브 모듈명 [as 이름]

# 데이터 분석

책이나 뉴스, 인터넷 등 다양한 매체에서 데이터 분석이라는 단어를 많이 들어 보았을 것입니다. 특히 인공지능이 이 시대의 흐름이 된 요즘, 데이터 분석은 거의 매일 빠지지 않고 등장하는 뉴스 거리입니다. 그럼 데이터 분석이라는 것은 무엇일까요? 데이터 분석은 사실 한마디로 정의하기가 매우 어렵습니다. 사전적 의미로만 보면 데이터는 자료, 정보를 의미하고 분석은 복잡한 대상을 정확하게 이해하기 위해 단순한 요소로 나누어서 설명하는 것을 말합니다. 그런데 두 단어를 합쳐 놓으면 정의하기가 어렵습니다. 분야나 상황에 따라 데이터 분석의 역할 수행과 범위가 매우 다르기 때문입니다. 위키피디아에서는 데이터 분석을 “유용한 정보를 발견하고 결론을 유추하거나 의사 결정을 돕기 위해 데이터를 조사, 정제, 변환, 모델링하는 과정” 으로 정의하고 있습니다. 현대 사회에서 데이터 분석은 비즈니스 결정을 과학적으로 내리기 위한 도구로 사용되는 경향이 있습니다. 데이터 분석과 함께 자주 언급되는 용어는 “데이터 과학” 입니다. 대부분 데이터 분석과 데이터 과학을 동일하게 취급하지만 두 용어를 분리해서 보면 데이터 과학은 통계학, 데이터 분석, 머신 러닝, 데이터 마이닝 등을 아우르는 큰 개념으로 볼 수 있습니다.

데이터 과학은 데이터 세계와 비즈니스 세계를 잇는 다리입니다. 데이터 과학을 활용해서 소프트웨어나 제품을 개발할 수 있지만 이것이 전부는 아닙니다. 또 데이터 과학이 통계학과 관련이 많다지만 통계학 자체는 아니며, 학술적인 분야 또한 아닙니다. 멋진 그래프를 그리기도 하지만 이것이 데이터 과학의 전부는 아닙니다. 오히려 데이터 과학은 이 모든 것을 포함합니다. 데이터 과학을 하려면 프로그래밍, 통계학, 시각화와 더불어 비즈니스 감각을 갖추어야 합니다. 데이터 분석과 데이터 과학이 가장 크게 다른 점은 두 분야가 만들어 내는 결과물입니다. 데이터 분석은 올바른 의사 결정을 돕기 위한 통찰을 제공하는 데 초점을 맞추며, 데이터 과학은 한 걸음 더 나아가 문제 해결을 위한 최선의 솔루션을 만드는 데 초점을 맞추고 있습니다.

|  |  |  |
| --- | --- | --- |
| **특징** | **데이터 분석** | **데이터 과학** |
| 범주 | 비교적 소규모 | 대규모 |
| 목표 | 의사 결정을 돕기 위한 통찰을 제공 | 문제 해결을 위해 최선을 솔루션을 만드는 일 |
| 주요 기술 | 컴퓨터 과학, 통계학, 시각화 등 | 컴퓨터 과학, 통계학, 머신 러닝, 인공지능 등 |
| 빅 데이터 | 사용 | 사용 |

이 책의 범위도 데이터를 가지고 통계를 통하여 데이터를 이해하고, 시각화를 통하여 데이터의 이해를 돕고 데이터를 머신 러닝에 사용하기 위해 가공하는 방법을 배우고 결과적으로 머신 러닝을 활용해서 의미 있는 결과를 도출하는데 초점을 맞춥니다.

지금까지 파이썬 자료구조에는 리스트, 튜플, 딕셔너리 등이 있었습니다. 파이썬은 자료구조를 기본적으로 제공하기 때문에 다른 프로그래밍 언어보다 더욱 쉽게 프로그램을 작성할 수 있습니다. 하지만 지금까지 배웠던 리스트, 튜플, 딕셔너리만 사용해서는 2차원 형태의 데이터를 다루는데 불편한 점이 있습니다. 2차원 데이터란 행과 열이 있는 데이터 형태를 말합니다. 흔히 우리가 엑셀에서 다루는 표 형태의 데이터를 2차원 데이터라고 이해하면 됩니다.

테이블이(가) 표시된 사진

자동 생성된 설명

그림 2차원 데이터의 예

판다스 모듈은 1차원 데이터를 다루는 시리즈(Series) 타입과 2차원 데이터를 다루는 데이터프레임(DataFrame) 타입을 제공합니다. 시리즈와 데이터프레임을 사용하면 데이터를 엑셀에 저장하거나 그래프로 시각화 하는 일을 쉽게 처리할 수 있습니다.

## 시리즈

판다스의 시리즈를 다루려면 가장 먼저 판다스 모듈을 불러와야 합니다. 판다스 모듈은 주로 pd라는 이름으로 많이 사용합니다.

예시 코드 판다스 모듈 불러오기

|  |
| --- |
| import pandas as pd |

판다스에서 다루는 시리즈는 1차원 데이터를 저장하기에 효과적인 자료구조입니다. 여기서 1차원 데이터라는 것은 엑셀 시트에 있는 하나의 행 또는 하나의 열에 있는 데이터를 의미합니다.

테이블이(가) 표시된 사진

자동 생성된 설명

그림 시리즈와 데이터프레임

### 시리즈 만들기

판다스 모듈의 Series() 함수를 활용하면 쉽게 시리즈를 만들 수 있습니다.

예시 코드 시리즈 만들기

|  |
| --- |
| s = pd.Series([100, 200, 300]) print(s) |

[실행 결과]

0 100

1 200

2 300

dtype: int64

위 코드를 실행하면 0부터 시작하는 정수 인덱스와 함께 리스트의 데이터가 출력됩니다. 0부터 시작하는 정수 인덱스에 값이 매칭되는 것을 보면 파이썬의 리스트와 비슷해 보입니다. 시리즈를 생성할 때, index 매개변수를 통해 행 이름도 다른 값으로 수정할 수 있습니다.

예시 코드 시리즈에 행 이름 설정하기

|  |
| --- |
| s = pd.Series([100, 200, 300], index = ['가', '나', '다']) print(s) |

[실행 결과]

가 100

나 200

다 300

dtype: int64

### 시리즈의 행 이름 가져오기

시리즈에 저장된 값은 인덱스 이름을 가져오려면 index를 사용해서 얻어올 수 있습니다.

예시 코드 시리즈 행, 열 이름 가져오기

|  |
| --- |
| import pandas as pd  s = pd.Series([100, 200, 300], index=['가','나','다'])print(s.index) |

[실행 결과]

Index(['가', '나', '다'], dtype='object')

### 시리즈 연산

시리즈는 파이썬의 리스트 자료형과 매우 흡사하지만 연산하는 데에 있어서 성격이 매우 다릅니다. 리스트 자료형은 단순히 데이터들을 저장하는데 의미가 있다면 시리즈는 병렬 연산이 가능합니다. 병렬 연산이 가능하다는 장점은 뒤에서 머신러닝을 다룰 때 큰 효과를 나타냅니다. 2개의 시리즈를 서로 더하거나 곱할 수 있습니다.

예시 코드 시리즈 연산하기

|  |
| --- |
| import pandas as pd  s = pd.Series([100, 200, 300], index=['가','나','다'])  print(s + s) print(s \* s) print(s \* 10) |

[실행 결과]

가 200

나 400

다 600

dtype: int64

가 10000

나 40000

다 90000

dtype: int64

가 1000

나 2000

다 3000

dtype: int64

유의해야할 점은 시리즈끼리 연산할 때, 실행 결과를 보면 같이 위치에 있는 값끼리 연산이 된 것처럼 보이지만 사실 같은 위치에 있는 행끼리 연산을 해주는 것이 아닙니다. 행 이름이 같은 값끼리 연산이 됩니다.

만약 개수가 다른 서로 다른 시리즈끼리 연산을 하면 어떻게 될까요?? 이 부분도 역시 같은 행 이름을 가진 값끼리 연산을 해줍니다.

예시 코드 개수가 다른 시리즈끼리 연산하기

|  |
| --- |
| import pandas as pd  s1 = pd.Series([100, 200, 300], index=['가','나','다']) s2 = pd.Series([100, 200], index=['가','나'])  print(s1 + s2) |

[실행 결과]

가 200.0

나 400.0

다 NaN

dtype: float64

위의 결과를 보면 s1에는 “다” 행이 있지만, s2에는 “다” 행이 없습니다. 그렇다면 “다” 행은 연산이 불가능하기 때문에 NaN(없는 값)으로 결과가 반환됩니다.

### 시리즈 전용 함수

시리즈에 있는 값을 활용해서 다양한 통계 함수와 데이터를 처리하는 다양한 함수들을 사용할 수 있습니다. 하나씩 소개해보겠습니다.

평균이란 모든 값의 합을 합한 데이터 개수만큼 나눈 값을 말합니다. 즉 특정 데이터 값들의 대표되는 값이라 볼 수 있습니다. 시리즈의 평균 값을 확인하려면 시리즈 내의 mean() 함수를 사용해주면 됩니다.

예시 코드 시리즈 평균값 확인하기

|  |
| --- |
| import pandas as pd  s = pd.Series([1,3,5,4,2]) print(s.mean()) |

[실행 결과]

3.0

중앙 값은 평균과 비슷합니다. 중앙 값은 값들을 크기 순으로 정렬했을 때, 가운데에 있는 값을 말합니다. 중앙 값을 표현하려면 median() 함수를 사용하면 됩니다.

예시 코드 시리즈 중앙값 확인하기

|  |
| --- |
| import pandas as pd  s = pd.Series([1,3,5,4,2]) print(s.median()) |

[실행 결과]

3

시리즈 내에 있는 값의 최소값, 최대값을 확인하려면 시리즈의 min(), max() 함수를 사용하면 됩니다.

예시 코드 시리즈 최소값, 최대값 확인하기

|  |
| --- |
| import pandas as pd  s = pd.Series([1,3,5,4,2])  print(s.min()) print(s.max()) |

[실행 결과]

1

5

시리즈 내에 있는 값을 정렬해서 보기 위해서는 시리즈의 sort\_values() 함수를 사용해주면 됩니다. 기본적으로는 오름차순으로 시리즈 값을 정렬해줍니다. 만약 내림차순으로 표현하려면 함수 내 매개변수로 ascending=False 값을 넣어주면 됩니다.

예시 코드 시리즈 정렬하기

|  |
| --- |
| import pandas as pd  s = pd.Series([1,3,5,4,2])  print(s.sort\_values()) print(s.sort\_values(ascending=False)) |

[실행 결과]

0 1

4 2

1 3

3 4

2 5

dtype: int64

2 5

3 4

1 3

4 2

0 1

dtype: int64

만약 시리즈 값들 중에서 중복된 값이 있고, 중복된 값을 없애고 싶을 때는 어떻게 할 수 있을까요? drop\_duplicates() 함수를 사용하면 됩니다.

예시 코드 시리즈 중복 값 삭제하기

|  |
| --- |
| import pandas as pd  s = pd.Series([1,3,5,4,2,1,3,5,4,2]) print(s.drop\_duplicates()) |

[실행 결과]

0 1

1 3

2 5

3 4

4 2

dtype: int64

시리즈 안에 있는 값들을 정렬을 하면 안 되는 상황이면서 정렬된 값을 확인해야하는 경우가 필요할 때가 있습니다. 그런 경우에는 argsort() 함수를 사용하면 시리즈 안에 있는 값들을 정렬하는 것이 아닌 값의 크기 순으로 인덱스만 가져오는 방법이 있습니다.

예시 코드 시리즈의 정렬된 인덱스 값 가져오기

|  |
| --- |
| import pandas as pd  s = pd.Series([1,3,5,4,2]) print(s.argsort()) |

[실행 결과]

0 0

1 4

2 1

3 3

4 2

dtype: int64

위의 결과를 보면 0, 4, 1, 3, 2 순으로 값이 나온 것을 알 수 있습니다. 이 결과의 의미는 시리즈의 0번째 행이 값이 제일 작았고, 그 다음에는 4번째 값이 제일 작았다는 의미입니다. 즉 값을 정렬해주는 것이 아닌 크기 순으로 해당 값들의 위치 값만 가져온다고 이해해주시면 됩니다.

시리즈 내에 만약 누락 값이 있는 경우 누락 값을 모두 없애기 위해서는 dropna() 함수를 사용해주면 쉽게 누락 값을 모두 없앨 수 있습니다. 누락 값이 있는 시리즈를 만들어 보겠습니다. 파이썬에서는 numpy 모듈을 사용해서 누락 값을 일부러 만들 수도 있습니다. 누락 값을 만들기 위해서는 numpy 모듈 안에 nan을 가져오면 됩니다.

예시 코드 시리즈 누락 값 처리하기

|  |
| --- |
| import pandas as pd import numpy as np  s = pd.Series([1,3,np.nan, 5,4,2, np.nan]) print(s.dropna()) |

[실행 결과]

0 1.0

1 3.0

3 5.0

4 4.0

5 2.0

dtype: float64

몇 가지 시리즈에서 많이 다루는 함수들을 알아보았습니다. 그 밖에도 다양한 시리즈 함수들이 존재합니다. 자세한 내용은 판다스 공식 사이트(<https://pandas.pydata.org/>)에 들어가 보시면 자세하게 설명하고 있으니 궁금하신 분들은 참고해주세요.

## 데이터프레임

시리즈들이 여러 개 모이면 하나의 데이터프레임이 됩니다. 이번에는 데이터 프레임에 대해서 살펴보도록 하겠습니다.

### 데이터프레임 만들기

가장 먼저 데이터 프레임을 만드는 방법은 판다스 모듈의 DataFrame() 함수를 사용하면 데이터 프레임을 만들 수 있습니다. DataFrame() 함수 안에 딕셔너리 형태로 데이터프레임을 만들 수도 있고, 리스트를 사용해서 데이터프레임을 만들 수도 있고, 시리즈를 활용해서 데이터프레임을 만들 수도 있습니다. 가장 먼저 딕셔너리를 이용해서 간단하게 데이터프레임을 만들어 보도록 하겠습니다.

예시 코드 딕셔너리로 데이터프레임 만들기

|  |
| --- |
| import pandas as pd  df = pd.DataFrame({  "이름" : ['홍길동', '장원영', '이순신'],  "성별" : ['남자', '여자', '남자'],  "나이" : [40, 20, 50] })  print(df) |

[실행 결과]

텍스트이(가) 표시된 사진

자동 생성된 설명

Jupyter Notebook에서는 print(df)를 입력할 시 위의 결과처럼 나오지 않습니다. 실행 결과처럼 데이터프레임을 확인하고 싶으신 분들은 print(df) 대신 print() 함수를 빼고 df만 적어주세요. 열 이름이 되는 곳에 key값을 할당하고, 값으로 표현하고자 하는 곳에는 리스트로 표현하고자 하는 값들을 묶어서 데이터프레임을 만들어 주시면 됩니다.

### 데이터프레임 행 이름 설정하기

데이터프레임이 만들어지면 자동으로 행 이름은 0부터 시작해서 1씩 증가하는 형태로 만들어집니다. 데이터프레임을 만들 때 행 이름을 다른 이름으로 설정할 수 있습니다. 데이터프레임을 생성할 때, DataFrame() 함수 안 매개변수에 index=[표시할 행 이름들]을 입력해주면 됩니다.

예시 코드 데이터프레임 행 이름 설정하기

|  |
| --- |
| import pandas as pd  df = pd.DataFrame({  "이름" : ['홍길동', '장원영', '이순신'],  "성별" : ['남자', '여자', '남자'],  "나이" : [40, 20, 50] }, index=['가', '나', '다'])  print(df) |

[실행 결과]

텍스트이(가) 표시된 사진

자동 생성된 설명

### 데이터프레임 열 순서 설정하기

데이터 프레임을 만들 때, 열 이름을 이름, 성별, 나이 순으로 설정했습니다. 하지만 표현할 때는 순서를 다르게 해서 표현할 수도 있습니다. 데이터프레임을 생성할 때, 표현할 열 순서는 DataFrame() 함수 내에 columns = [열 순서]를 설정해주시면 됩니다.

예시 코드 데이터프레임 열 순서 설정하기

|  |
| --- |
| import pandas as pd  df = pd.DataFrame({  "이름" : ['홍길동', '장원영', '이순신'],  "성별" : ['남자', '여자', '남자'],  "나이" : [40, 20, 50] }, index=['가', '나', '다'], columns = ['성별', '나이', '이름'])  print(df) |

[실행 결과]

텍스트, 테이블이(가) 표시된 사진

자동 생성된 설명

### 파일 불러오기

데이터프레임을 직접 만드는 경우는 거의 없습니다. 데이터프레임을 만드는 경우는 대부분 데이터 크롤링을 통해 인터넷에서 데이터를 수집해왔거나 다양한 시리즈를 데이터프레임을 만드는 경우가 아니면 대부분은 엑셀 파일이나 csv파일 등을 가져와서 데이터분석 하는 경우가 대부분입니다. 이번에는 파일을 읽어와서 데이터프레임으로 만들어보도록 하겠습니다. csv파일을 하나 불러와보겠습니다. csv파일을 불러오려면 판다스 모듈의 read\_csv() 함수를 사용하면 쉽게 가져올 수 있습니다. read\_csv() 함수에 가져오고자 하는 파일의 경로를 입력해주면 됩니다. 필자는 같은 폴더에 concat\_1.csv 파일을 위치시키고 파일을 불러와보겠습니다. 같은 폴더에 있는 파일을 불러올 때는 따로 경로 설정을 해줄 필요가 없습니다.

예시 코드 csv파일 불러오기

|  |
| --- |
| import pandas as pd  df = pd.read\_csv("concat\_1.csv") print(df) |

[실행 결과]

텍스트, 전자기기, 키보드이(가) 표시된 사진

자동 생성된 설명

데이터 분석을 할 때, csv파일 말고도 엑셀 파일, tsv파일 등 다양합니다. 만약 엑셀 파일을 불러오고자 할 때는 판다스 모듈의 read\_excel() 함수를 사용하면 엑셀 파일을 불러올 수 있습니다. 그런데 tsv파일을 불러올 때는 따로 read\_tsv() 함수는 없습니다. 그럼 어떻게 tsv파일을 불러올 수 있을까요? 먼저 csv파일과 tsv파일의 차이를 알아야 합니다. csv파일을 데이터를 콤마(,)로 구분 지어서 만든 파일 파일입니다. csv파일을 실제로 열면 아래와 같은 모양입니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

그림 csv파일

그러면 tsv파일은 어떻게 생겼을까요? tsv파일은 콤마가 아닌 탭(Tab)을 기준으로 데이터들을 구분합니다. tsv파일을 열어보면 아래와 같습니다.

테이블이(가) 표시된 사진

자동 생성된 설명

그림 tsv파일

csv파일과 tsv파일은 단순히 데이터를 콤마로 구분하느냐 탭으로 구분하느냐의 차이입니다. 따라서 매개변수만 잘 조정하면 read\_csv() 함수로도 tsv파일을 불러올 수 있다는 뜻이죠. read\_csv() 함수 안에는 기본 값으로 sep=’,’이 설정되어있습니다. 이 설정값을 탭으로만 바꾸어주면 됩니다. 탭키는 “\t” 문자로 표현할 수 있습니다. read\_csv() 함수로 tsv파일을 불러와보겠습니다.

예시 코드 tsv파일 불러오기

|  |
| --- |
| import pandas as pd  df = pd.read\_csv("gapminder.tsv", sep = '\t') print(df) |

[실행 결과]

테이블이(가) 표시된 사진

자동 생성된 설명

## 누락 값 처리하기

누락 값이란 말 그대로 누락된 데이터를 말합니다. 누락 값을 처리하는 일은 매우 중요합니다. 만약 누락 값이 있다면 제대로 된 연산이 힘들 것이고, 뒤에서 다룰 머신 러닝에서도 누락 값이 있으면 학습이 불가능하기 때문입니다. 누락 값은 NaN, nan, NAN과 같은 방법으로 표기할 수 있습니다. 이 책에서는 누락 값을 NaN이라고 표기하겠습니다.

누락 값을 다룰 때 유의해야할 점이 있습니다. 학창 시절에 무한대에 대해서 배웠다면, 무한대끼리 비교가 불가능하다는 것을 알 것입니다. 같은 무한대지만 어느 무한대가 더 큰 무한대인지 확실하게 이야기할 수 없기 때문이죠. 누락 값도 마찬가지입니다. 존재하지 않는 것이기 때문에 누락 값 역시 비교가 불가능합니다. 하지만 많은 사람들이 누락 값을 처리하기 위해서 누락 값인지 IF문을 통해서 비교하는 실수를 저지릅니다. 누락 값은 0이나 “”와 같은 값과는 다른 개념입니다. 누락 값은 말 그대로 데이터 자체가 없다는 것을 의미합니다. 그래서 “같다” 라는 개념도 없습니다. 아래 코드는 누락 값과 True, False, 0, “”을 비교한 결과입니다.

예시 코드 누락 값 비교하기

|  |
| --- |
| print(nan == True) print(nan == False) print(nan == nan) print(nan == 0) print(nan == "") |

[실행 결과]

False

False

False

False

False

### 누락 값 확인하기

값이 누락 값인지 누락 값이 아닌지 확인하려면 판다스의 isnull() 함수를 사용해야만 누락 값인지 확인할 수가 있습니다.

예시 코드 isnull() 함수 사용해서 누락 값 확인하기

|  |
| --- |
| print(pd.isnull(nan)) print(pd.isnull(5)) |

[실행 결과]

True

False

누락 값이 아닌 경우도 확인할 수 있습니다. 누락 값이 아닌지 확인하려면 isnull() 함수의 반대인 notnull() 함수도 있습니다.

예시 코드 notnull() 함수 사용해서 누락 값이 아닌 값 확인하기

|  |
| --- |
| print(pd.notnull(nan)) print(pd.notnull(5)) |

[실행 결과]

False

True

### 누락 값 개수 구하기

데이터프레임의 각 열의 누락 값의 개수를 구할 때는 info() 함수를 사용하면 됩니다. info() 함수는 누락 값 뿐만 아니라 각 열의 타입도 확인할 수 있기 때문에 데이터프레임을 불러오면 사실 가장 먼저 실행해봐야 하는 함수입니다.

예시 코드 info() 함수를 통해 누락 값 개수 확인하기

|  |
| --- |
| import pandas as pd  df = pd.read\_csv("country\_timeseries.csv") df.info() |

[실행 결과]

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 122 entries, 0 to 121

Data columns (total 18 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 Date 122 non-null object

1 Day 122 non-null int64

2 Cases\_Guinea 93 non-null float64

...중략...

15 Deaths\_UnitedStates 18 non-null float64

16 Deaths\_Spain 16 non-null float64

17 Deaths\_Mali 12 non-null float64

dtypes: float64(16), int64(1), object(1)

memory usage: 17.3+ KB

결과의 Column을 보면 각 열들이 있고 그 옆에 Non-Null Count 열에 데이터의 개수가 있습니다. 그리고 그 옆에 Dtype열에는 각 열들의 타입을 확인할 수 있는데, object는 객체가 아니라 문자열이라는 의미입니다. int64는 정수로 된 데이터를 말하며, float64 실수 형태의 데이터를 말합니다.

불러온 country\_timeseries.csv 파일에 누락 값이 아주 많이 있는 것을 확인했습니다.

예시 코드 country\_timeseries.csv 파일 확인하기

|  |
| --- |
| import pandas as pd  df = pd.read\_csv("country\_timeseries.csv") print(df.iloc[:5, :5]) |

[실행 결과]

Date Day Cases\_Guinea Cases\_Liberia Cases\_SierraLeone

0 1/5/2015 289 2776.0 NaN 10030.0

1 1/4/2015 288 2775.0 NaN 9780.0

2 1/3/2015 287 2769.0 8166.0 9722.0

3 1/2/2015 286 NaN 8157.0 NaN

4 12/31/2014 284 2730.0 8115.0 9633.0

### 누락 값 채우기

누락 값은 누락 값을 임의의 값으로 변경하거나 데이터프레임에 이미 있는 값으로 대신 채우는 방법 등으로 처리할 수 있습니다. 누락 값을 채우려면 판다스의 fillna() 함수를 사용하면 됩니다. fillna() 함수 안에 값을 넣으면 모든 누락 값을 입력한 값으로 채울 수 있습니다.

예시 코드 누락 값 0으로 채우기

|  |
| --- |
| import pandas as pd  df = pd.read\_csv("country\_timeseries.csv") print(df.fillna(0).iloc[:5, :5]) |

[실행 결과]

Date Day Cases\_Guinea Cases\_Liberia Cases\_SierraLeone

0 1/5/2015 289 2776.0 0.0 10030.0

1 1/4/2015 288 2775.0 0.0 9780.0

2 1/3/2015 287 2769.0 8166.0 9722.0

3 1/2/2015 286 0.0 8157.0 0.0

4 12/31/2014 284 2730.0 8115.0 9633.0

fillna() 함수 안 매개변수 method에 “ffill”을 설정하면 누락 값이 나타나기 전의 값으로 즉, 바로 앞에 있는 값으로 누락 값을 처리할 수 있습니다. 하지만 앞에 데이터가 없다면, 채울 수 없다는 단점이 있습니다.

예시 코드 앞에 있는 값으로 누락 값 채우기

|  |
| --- |
| import pandas as pd  df = pd.read\_csv("country\_timeseries.csv") print(df.fillna(method='ffill').iloc[:5, :5]) |

[실행 결과]

Date Day Cases\_Guinea Cases\_Liberia Cases\_SierraLeone

0 1/5/2015 289 2776.0 NaN 10030.0

1 1/4/2015 288 2775.0 NaN 9780.0

2 1/3/2015 287 2769.0 8166.0 9722.0

3 1/2/2015 286 2769.0 8157.0 9722.0

4 12/31/2014 284 2730.0 8115.0 9633.0

fillna() 함수의 method 옵션 값을 “bfill”로 설정하면 “ffill”과는 반대로 뒤에 있는 값을 가져옵니다. 이 역시 뒤에 데이터가 없다면 값을 채울 수 없다는 단점이 있습니다.

예시 코드 뒤에 있는 값으로 누락 값 채우기

|  |
| --- |
| import pandas as pd  df = pd.read\_csv("country\_timeseries.csv") print(df.fillna(method='bfill').iloc[:5, :5]) |

[실행 결과]

Date Day Cases\_Guinea Cases\_Liberia Cases\_SierraLeone

0 1/5/2015 289 2776.0 8166.0 10030.0

1 1/4/2015 288 2775.0 8166.0 9780.0

2 1/3/2015 287 2769.0 8166.0 9722.0

3 1/2/2015 286 2730.0 8157.0 9633.0

4 12/31/2014 284 2730.0 8115.0 9633.0

만약 각각의 열의 평균값으로 각각의 누락 값을 채울 때는 fillna 함수에 df.mean 함수를 넣어주면 됩니다.

예시 코드 각 열의 평균 값으로 각 열의 누락 값 채우기

|  |
| --- |
| import pandas as pd  df = pd.read\_csv("country\_timeseries.csv") print(df.fillna(df.mean()).iloc[:5, :5]) |

[실행 결과]

Date Day Cases\_Guinea Cases\_Liberia Cases\_SierraLeone

0 1/5/2015 289 2776.000000 2335.337349 10030.000000

1 1/4/2015 288 2775.000000 2335.337349 9780.000000

2 1/3/2015 287 2769.000000 8166.000000 9722.000000

3 1/2/2015 286 911.064516 8157.000000 2427.367816

4 12/31/2014 284 2730.000000 8115.000000 9633.000000

interpolate 함수는 누락 값을 데이터의 흐름에 맞게 채울 수 있습니다. 흐름에 맞게 채우는 방법에는 두 가지가 있는데, 데이터의 흐름에 맞게 채우는 방법 또는 시간의 흐름에 맞게 채우는 방법이 있습니다.

예시 코드 누락 값을 가진 시리즈 만들기

|  |
| --- |
| import pandas as pd from numpy import nan  date = ['2022/12/01', '2022/12/02', '2022/12/09', '2022/12/10'] date = pd.to\_datetime(date, format = '%Y/%m/%d')  values = [1, nan, nan, 10]  series = pd.Series(values, index = date) print(series) |

[실행 결과]

2022-12-01 1.0

2022-12-02 NaN

2022-12-09 NaN

2022-12-10 10.0

dtype: float64

1과 10 사이에 누락 값이 2개가 있습니다. 단순히 누락 값을 평균 값 5.5으로 채우기에는 흐름에 맞지 않아 보입니다. interpolate() 함수를 사용하면 데이터의 흐름에 맞게 자동으로 채울 수 있습니다.

예시 코드 흐름에 맞게 누락 값 채우기

|  |
| --- |
| import pandas as pd from numpy import nan  date = ['2022/12/01', '2022/12/02', '2022/12/09', '2022/12/10'] date = pd.to\_datetime(date, format = '%Y/%m/%d')  values = [1, nan, nan, 10]  series = pd.Series(values, index = date) series = series.interpolate() print(series) |

[실행 결과]

2022-12-01 1.0

2022-12-02 4.0

2022-12-09 7.0

2022-12-10 10.0

dtype: float64

위의 결과처럼 interpolate() 함수를 사용하면 데이터의 흐름에 맞게 자동으로 채워줍니다. 하지만 날짜를 자세히 보아보면, 12월 1일에는 값이 1이고, 12월 10일에는 값이 10입니다. 그렇다면 왠지 12월 2일에는 값이 2일 것 같고, 12월 9일에는 값이 9일 것 같습니다. 정말 시간에 따른 데이터가 맞다면 데이터의 흐름에 맞게 누락 값을 채우면 안 됩니다. 이런 경우 interpolate() 함수 안에 옵션 값으로 **method=”time”**을 입력해주면 됩니다.

예시 코드 시간에 흐름에 맞게 누락 값 채우기

|  |
| --- |
| import pandas as pd from numpy import nan  date = ['2022/12/01', '2022/12/02', '2022/12/09', '2022/12/10'] date = pd.to\_datetime(date, format = '%Y/%m/%d')  values = [1, nan, nan, 10]  series = pd.Series(values, index = date) series = series.interpolate(method='time') print(series) |

[실행 결과]

2022-12-01 1.0

2022-12-02 2.0

2022-12-09 9.0

2022-12-10 10.0

dtype: float64

### 누락 값 삭제하기

데이터프레임의 행에 있는 값들 중 누락 값이 단 한개라도 있으면 그 행 자체를 삭제해버리는 dropna() 함수가 있습니다.

예시 코드 누락 값 있는 행 삭제하기

|  |
| --- |
| import pandas as pd  df = pd.read\_csv("country\_timeseries.csv") print(df.dropna()) |

[실행 결과]

Date Day ... Deaths\_Spain Deaths\_Mali

19 11/18/2014 241 ... 0.0 6.0

[1 rows x 18 columns]

만약 조건을 만족하는 값을 가지고 있는 행을 없애려면 해당하는 열 값에 인덱싱 한 후, 조건을 적고 index 기능을 이용해 위치 값을 가져와서 drop() 함수에 얻은 인덱스 값을 넣어주면 됩니다. 예를 들어 데이터프레임 df에서 Day열의 값들 중 100이 넘어가는 값을 가진 모든 행을 삭제해보겠습니다.

예시 코드 Day열의 값이 100을 초과하는 행 삭제하기

|  |
| --- |
| import pandas as pd  df = pd.read\_csv("country\_timeseries.csv") idx = df[df['Day'] > 100].index  print(df.drop(idx).iloc[:, :3]) |

[실행 결과]

Date Day Cases\_Guinea

79 6/30/2014 100 413.0

80 6/22/2014 92 NaN

81 6/20/2014 90 390.0

...중략

119 3/25/2014 3 86.0

120 3/24/2014 2 86.0

121 3/22/2014 0 49.0

## 데이터 시각화 하는 방법

파이썬 에서는 데이터를 시각화 하는 아주 많은 모듈들을 제공하고 있습니다. 인기가 많은 모듈로는 matplotlib, seaborn 등이 있습니다. matplotlib 모듈은 그래프를 그리는 데에 있어서 유용한 함수들을 많이 제공하고 있어서 원하는 형태의 그림을 자유자재로 그릴 수 있으며, seaborn 모듈은 사실 데이터분석 연습용 모듈인데, 예쁜 그림을 그려주는 고급 함수들을 많이 제공하고 있어서 seaborn 모듈 역시 많이 사용되고 있습니다.

seaborn 모듈에는 데이터 분석 연습용 모듈 답게 데이터분석을 연습할 수 있는 연습용 데이터도 많이 제공하고 있습니다. seaborn 모듈 안에 있는 tips 데이터 집합을 가져와보겠습니다. 여기서 tips 데이터 집합은 어떤 식당에서 팁을 지불한 손님의 정보를 모아둔 데이터입니다.

이번 실습에서는 tips 데이터 집합을 사용해 다양한 그래프를 그려보도록 하겠습니다. 그래프를 이해하는 데 필요한 개념은 그래프를 그리기 전에 설명하면서 진행하겠습니다.

tips 데이터 집합을 불러와 변수 tips에 저장하겠습니다. tips 데이터프레임은 음식 지불 금액(total\_bill), 팁은 얼마 주었는지(tip), 어떤 성별이 계산했는지(sex), 흡연하는 테이블인지 아닌지(smoker), 어떤 요일에 왔는지(day), 점심에 왔는지 저녁에 왔는지(time), 몇 명이서 왔는지(size)에 대한 정보가 담겨있습니다.

예시 코드 tips 데이터프레임 가져오기

|  |
| --- |
| import seaborn as sns  tips = sns.load\_dataset("tips") print(tips) |

[실행 결과]

테이블이(가) 표시된 사진

자동 생성된 설명

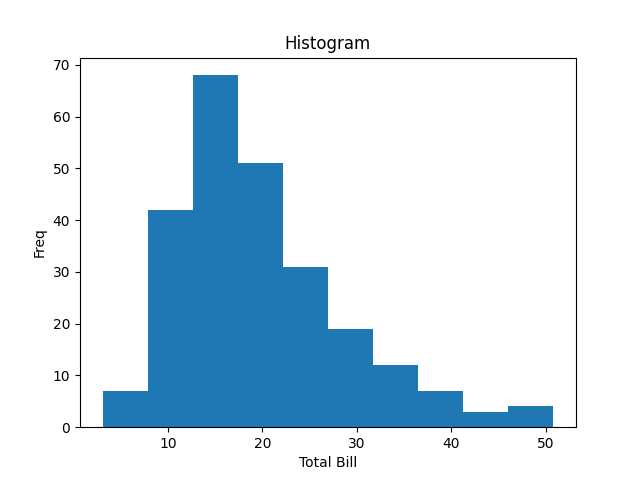
### matplotlib 모듈을 활용해서 그래프 그리기

히스토그램은 데이터프레임의 열 데이터 분포와 빈도를 살펴보는 용도로 자주 사용하는 그래프입니다. 이 때 데이터프레임의 total\_bill, tip 등의 열을 변수라고 부르기도 합니다. 그리고 변수를 하나만 사용해서 그린 그래프를 “일변량 그래프” 라고 합니다. matplotlib 모듈을 사용해서 히스토그램을 그려봅시다.

예시 코드 matplotlib 모듈로 히스토그램 그리기

|  |
| --- |
| import matplotlib.pyplot as plt import seaborn as sns  tips = sns.load\_dataset("tips")  fig = plt.figure() ax = fig.add\_subplot(1, 1, 1)  ax.hist(tips['total\_bill'], bins=10)  ax.set\_title("Histogram") ax.set\_xlabel("Total Bill") ax.set\_ylabel("Freq")  plt.show() |

[실행 결과]



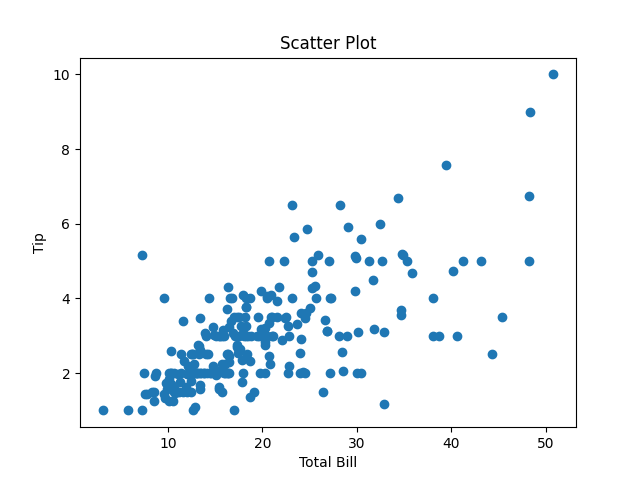
hist() 함수 안 매개변수 bins는 막대기 개수를 설정하는 옵션 값입니다. 숫자를 조정해서 막대기의 개수를 조정할 수 있습니다.

이번에는 산점도 그래프를 그려보겠습니다. 산점도 그래프는 변수 2개를 사용해서 만드는 그래프입니다. 2개의 특성을 사용하기 때문에 통계 용어로 “이변량 그래프”라고 부릅니다. tips 데이터프레임에서 x축은 total\_bill을 표현하고, y축은 tip을 표현해서 산점도를 그려보겠습니다.

예시 코드 matplotlib 모듈로 산점도 그리기

|  |
| --- |
| import matplotlib.pyplot as plt import seaborn as sns  tips = sns.load\_dataset("tips") fig = plt.figure() ax = fig.add\_subplot(1, 1, 1)  ax.scatter(tips['total\_bill'], tips['tip']) ax.set\_title("Scatter Plot") ax.set\_xlabel("Total Bill") ax.set\_ylabel("Tip")  plt.show() |

[실행 결과]

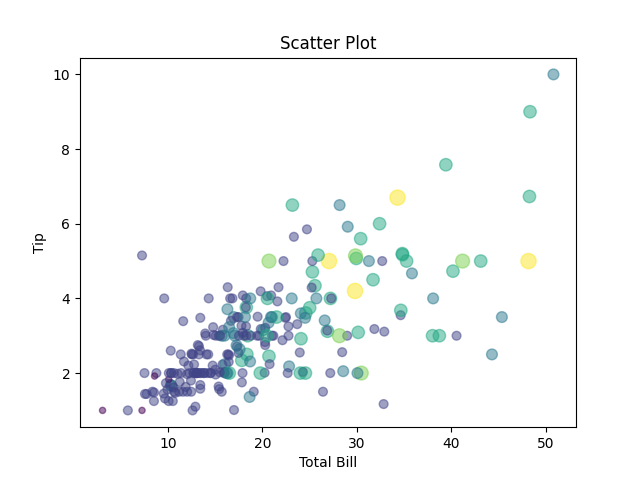


matplotlib 모듈은 원하는 대로 그림을 그릴 수 있는 많은 유용한 함수와 옵션들을 제공하고 있습니다. 앞서 그린 산점도도 몇 가지 매개변수를 더 추가하면 더 예쁜 그림을 그릴 수 있습니다.

예시 코드 산점도 꾸미기

|  |
| --- |
| import matplotlib.pyplot as plt import seaborn as sns  tips = sns.load\_dataset("tips")  fig = plt.figure() ax = fig.add\_subplot(1, 1, 1)  ax.scatter(tips['total\_bill'], tips['tip'], c = tips['size'], s = tips['size'] \* 20, alpha = 0.5) ax.set\_title("Scatter Plot") ax.set\_xlabel("Total Bill") ax.set\_ylabel("Tip")  plt.show() |

[실행 결과]



scatter() 함수 안에 매개변수 c, s, alpha를 추가했습니다. x축은 Total Bill, y축은 Tip, 색깔로 Size 특성을 표현했기 때문에 3가지 특성을 표현했습니다. 2개의 특성을 넘어가는 이러한 그래프를 “다변량 그래프” 라고 부릅니다. 여기서 c는 색깔을 의미하고, s는 점의 사이즈, alpha는 투명도를 의미합니다. matplotlib() 함수에 대해서 조금 더 자세히 알고 싶은 분들은 matplotlib 공식 사이트를 참고해보는 것을 추천합니다. (<https://matplotlib.org/>)

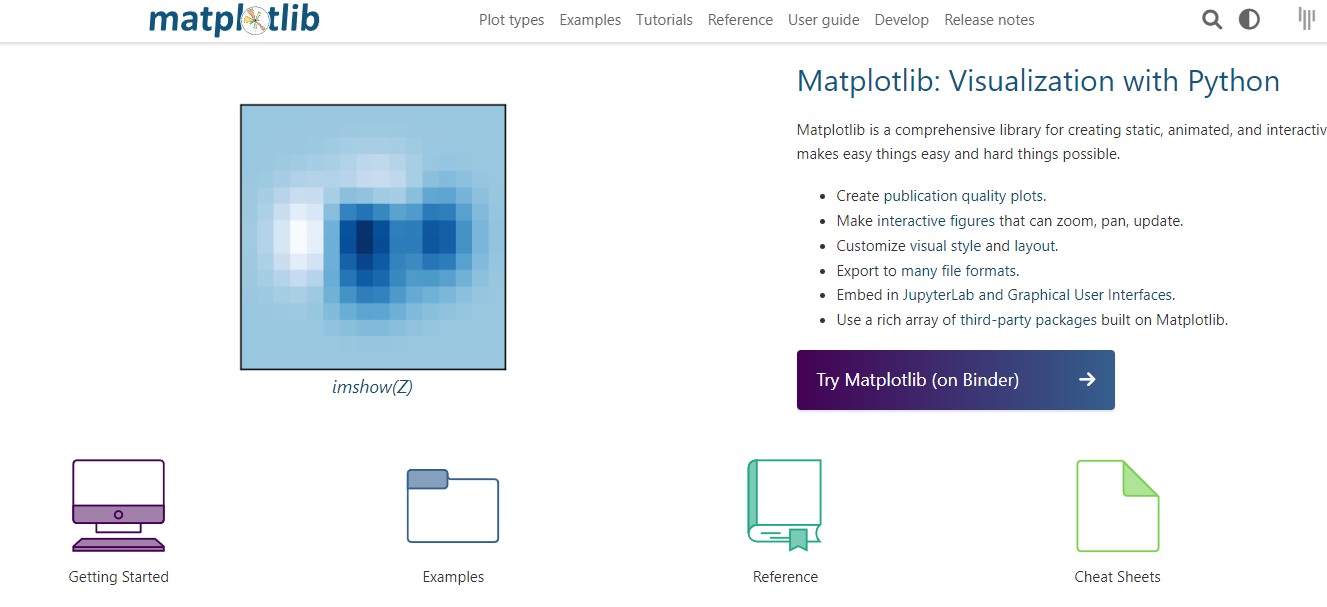


그림 matplotlib 공식 사이트

### seaborn 모듈을 활용해서 그래프 그리기

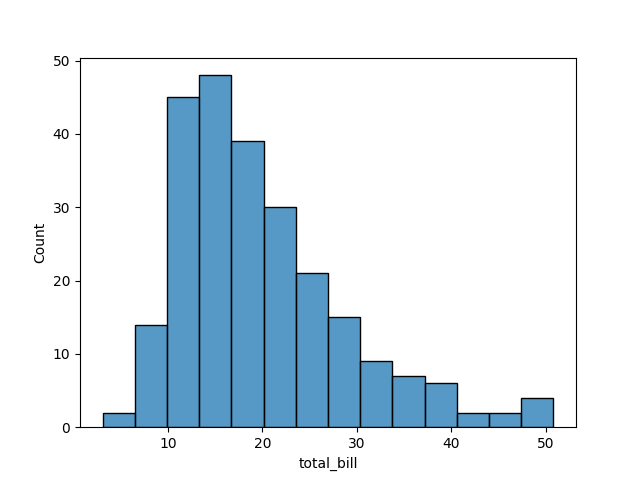
seaborn 모듈을 사용해서 tips 데이터 집합만 사용했지만, 이번에는 seaborn 모듈 자체를 활용해서 그래프를 그려보겠습니다. (seaborn 모듈은 쉬운 짧은 코드로 화려한 그래프를 그릴 수 있어서 필자가 애용하는 모듈입니다.)

seaborn 모듈은 대부분 “sns” 라는 이름으로 줄여서 사용합니다. seaborn 모듈로 히스토그램을 그리려면 histplot()을 사용하면 됩니다.

예시 코드 seaborn 모듈로 히스토그램 그리기

|  |
| --- |
| import matplotlib.pyplot as plt import seaborn as sns  tips = sns.load\_dataset("tips") sns.histplot(tips['total\_bill']) plt.show() |

[실행 결과]

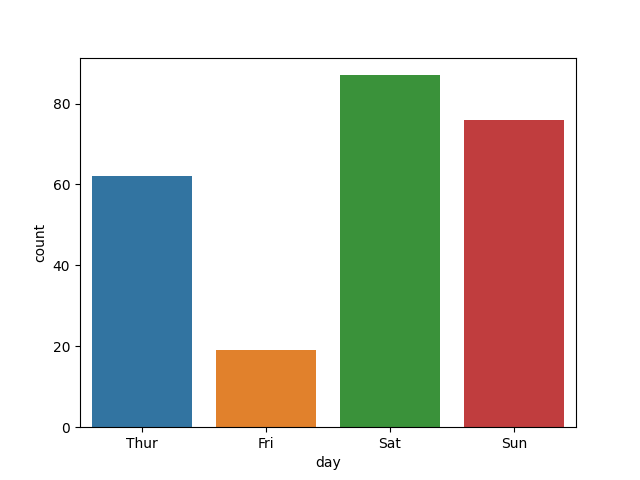


이번에는 히스토그램과 비슷하게 생긴 count 그래프를 그려보겠습니다. count 그래프는 특정 열 안에 있는 값이 몇 개인지 세어주는 그래프입니다.

예시 코드 seaborn 모듈로 countplot 그리기

|  |
| --- |
| import matplotlib.pyplot as plt import seaborn as sns  tips = sns.load\_dataset("tips") sns.countplot(x=tips['day'], data=tips) plt.show() |

[실행 결과]

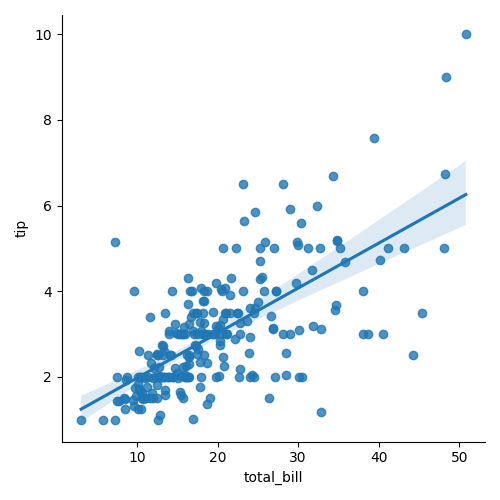


seaborn 모듈에서는 matplotlib 모듈보다 훨씬 더 쉬운 방법으로 산점도를 그릴 수 있고 게다가 회귀선까지 함께 그릴 수가 있습니다.

예시 코드 seaborn 모듈로 countplot 그리기

|  |
| --- |
| import matplotlib.pyplot as plt import seaborn as sns  tips = sns.load\_dataset("tips") sns.lmplot(x='total\_bill', y='tip', data=tips) plt.show() |

[실행 결과]

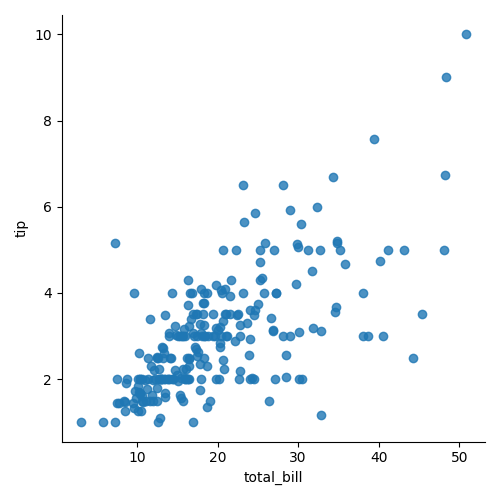


만약 seaborn 모듈의 산점도에서 회귀선을 제거하고 싶다면, lmplot() 함수 내에 있는 매개변수에 **fit\_reg=False**를 입력해주면 됩니다.

예시 코드 회귀선 지우기

|  |
| --- |
| import matplotlib.pyplot as plt import seaborn as sns  tips = sns.load\_dataset("tips") sns.lmplot(x='total\_bill', y='tip', data=tips, fit\_reg=False) plt.show() |

[실행 결과]

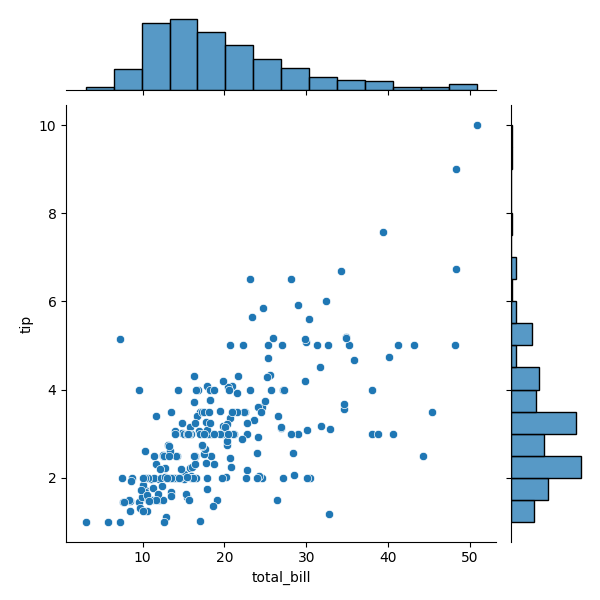


seaborn 모듈에는 산점도 그래프와 히스토그램을 같이 한 번에 그려주는 jointplot 그래프도 있습니다. 사용법은 산점도 그래프를 그리는 것과 같습니다.

예시 코드 seaborn 모듈로 jointplot 그래프 그리기

|  |
| --- |
| import matplotlib.pyplot as plt import seaborn as sns  tips = sns.load\_dataset("tips") sns.jointplot(x='total\_bill', y='tip', data=tips) plt.show() |

[실행 결과]

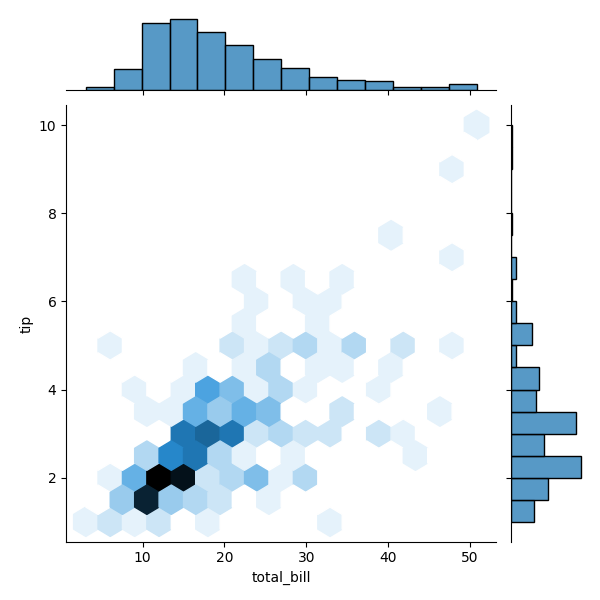


산점도 그래프는 점이 겹쳐 보일 경우 점이 몇 개가 있는지 구분하기 어렵다는 단점이 있습니다. 만약 산점도 그래프의 데이터를 구분하기 쉽게 그리고 싶다면 육각 그래프(hexbin)를 사용하면 됩니다. 육각 그래프는 2차원 표면에 육각형으로 데이터를 쌓아 표현하는 그래프입니다. 그래서 특정 데이터의 개수가 많아지면 점점 진한 색으로 표현됩니다. 육각 그래프는 jointplot 그래프를 그릴 때 설정해줄 수 있습니다. joinplot() 그래프 함수 안에 kind=”hex” 로 설정하면 됩니다.

예시 코드 jointplot 그래프에 육각 그래프 그리기

|  |
| --- |
| import matplotlib.pyplot as plt import seaborn as sns  tips = sns.load\_dataset("tips") sns.jointplot(x='total\_bill', y='tip', data=tips, kind='hex') plt.show() |

[실행 결과]

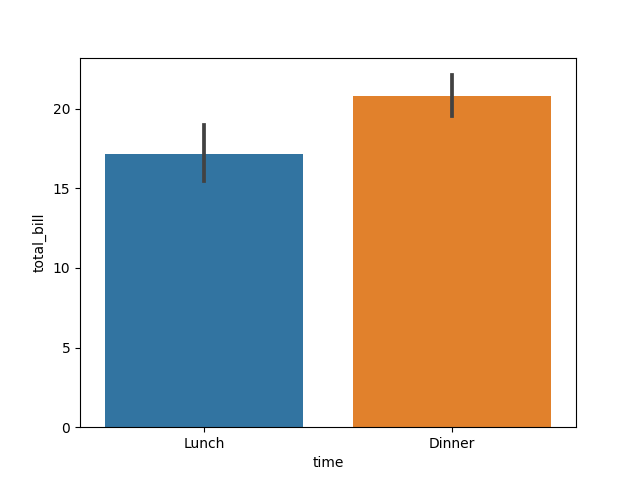


이번에는 바 그래프를 그려보겠습니다. 바 그래프는 지정한 변수의 평균을 계산해서 그릴 수 있습니다. 시간에 따라 지불한 비용의 평균을 바 그래프로 그려보겠습니다.

예시 코드 seaborn 모듈로 jointplot 그래프 그리기

|  |
| --- |
| import matplotlib.pyplot as plt import seaborn as sns  tips = sns.load\_dataset("tips") sns.barplot(x='time', y='total\_bill', data=tips) plt.show() |

[실행 결과]

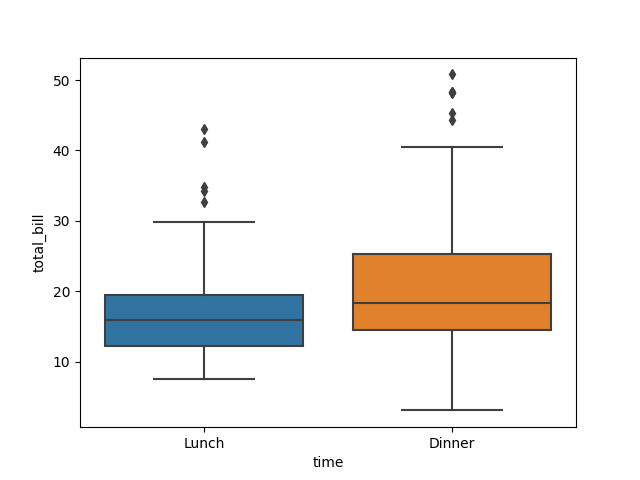


이번에는 박스 그래프를 그려보겠습니다. 박스 그래프는 최솟값, 1분위수, 중앙값, 3분위수, 최댓값, 이상치 등을 알 수 있습니다. 다양한 통계량을 한 번에 표현하는 그래프이며, 특히 이상치가 있는지 판단하기 좋은 그래프이기 때문에 가장 사랑받고 있는 그래프 중 하나입니다.

예시 코드 seaborn 모듈로 jointplot 그래프 그리기

|  |
| --- |
| import matplotlib.pyplot as plt import seaborn as sns  tips = sns.load\_dataset("tips") sns.boxplot(x='time', y='total\_bill', data=tips) plt.show() |

[실행 결과]

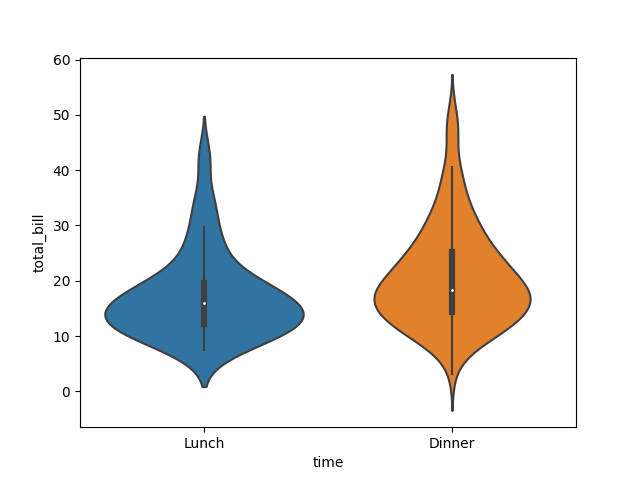


박스 그래프는 다양한 통계 수치를 확인하기 위해 자주 사용되는 그래프이지만, 단순히 네모 모양으로 그래프를 표현하기 때문에 데이터 분산이 모호하게 표현이 돼서 데이터들이 어디에 밀집이 되어있는지 자세한 내용을 알기 어렵습니다. 이런 경우에는 커널 밀도를 추정한 바이올린 그래프를 사용해서 문제를 해결할 수 있습니다. 하지만 이상치를 확인하기가 어렵다는 단점이 있습니다.

예시 코드 seaborn 모듈로 jointplot 그래프 그리기

|  |
| --- |
| import matplotlib.pyplot as plt import seaborn as sns  tips = sns.load\_dataset("tips") sns.violinplot(x='time', y='total\_bill', data=tips) plt.show() |

[실행 결과]

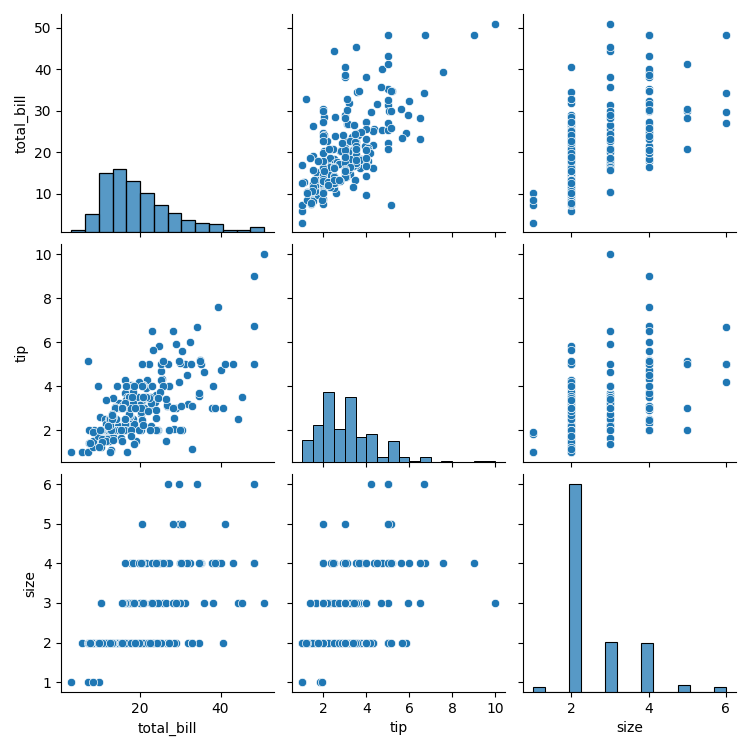


마지막으로 관계 그래프를 그려보겠습니다. 관계 그래프는 여러 개의 특성의 상관관계를 한 번에 알아보기 쉽게 산점도로 비교할 수 있는 모든 열을 비교해서 보여줍니다.

예시 코드 seaborn 모듈로 관계 그래프 그리기

|  |
| --- |
| import matplotlib.pyplot as plt import seaborn as sns  tips = sns.load\_dataset("tips") sns.pairplot(tips) plt.show() |

[실행 결과]

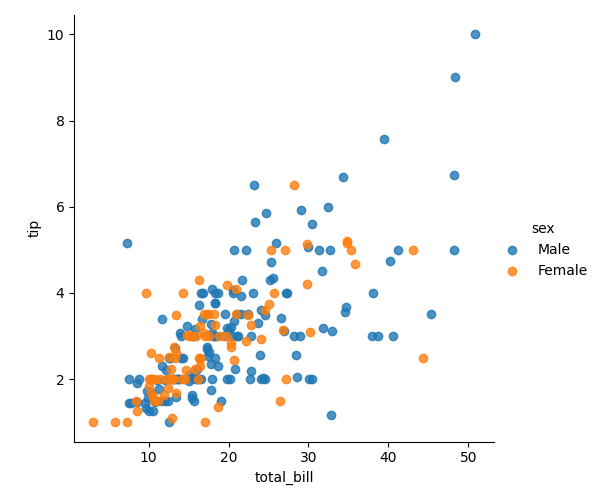


이번에는 seaborn 모듈을 활용해서 이변량 이상의 다변량 그래프를 그려보겠습니다. 2차원 그래프를 그릴 때, 2개의 특성을 넘어가는 그래프를 그리려면 색깔을 조정해주는 방법을 사용할 수 있습니다. 특성을 추가할 때 색으로 표현하려면 그래프 함수 안에 매개변수 hue에 추가하고 싶은 특성을 입력해주면 됩니다.

예시 코드 그래프에 색상 추가하기1

|  |
| --- |
| import matplotlib.pyplot as plt import seaborn as sns  tips = sns.load\_dataset("tips") sns.lmplot(x='total\_bill', y='tip', data=tips, hue='sex', fit\_reg=False) plt.show() |

[실행 결과]

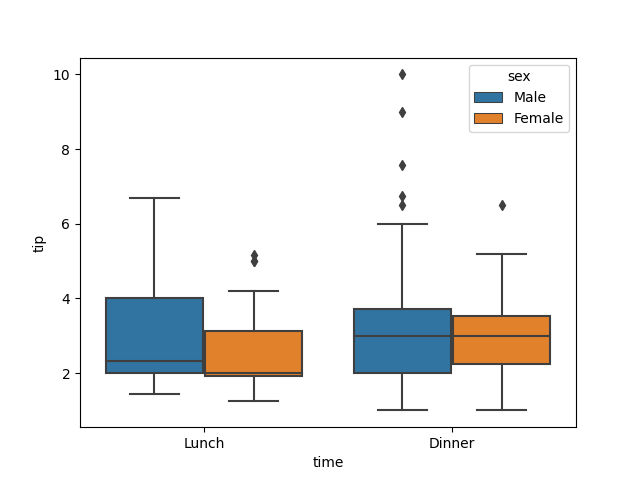


다른 그래프 역시 hue 매개변수를 통해서 색깔로 데이터를 더 추가해서 표현할 수 있습니다.

예시 코드 그래프에 색상 추가하기2

|  |
| --- |
| import matplotlib.pyplot as plt import seaborn as sns  tips = sns.load\_dataset("tips") sns.boxplot(x='time', y='tip', data=tips, hue='sex') plt.show() |

[실행 결과]

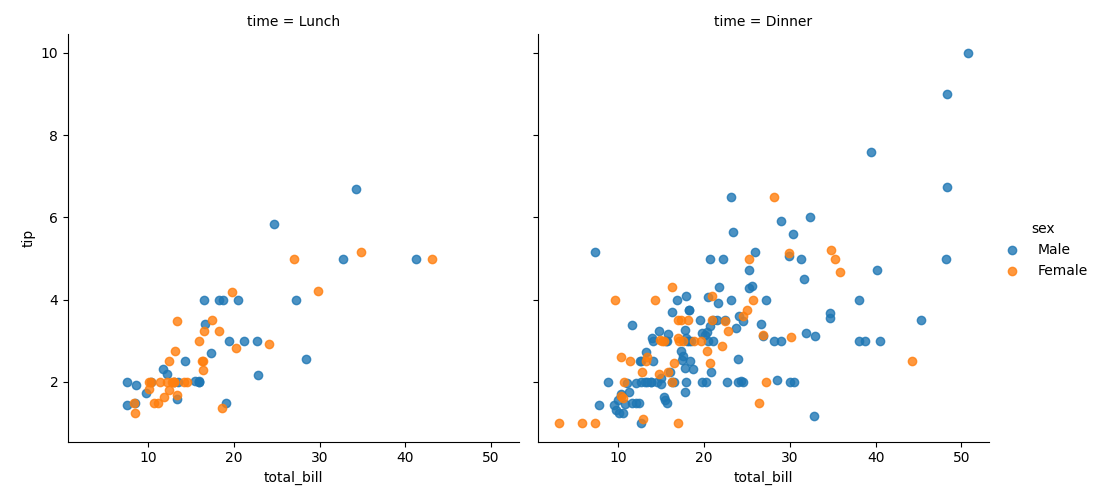


색깔을 추가하고 2차원 그래프에 3가지의 특성을 표현할 수 있습니다. 이제 x축도, y축도, 색깔도 사용했는데, 4가지 특성을 2차원 그래프에 표현이 가능할까요? 매개변수 col에 특성을 추가하여 열 단위로 그래프를 분할해서 그리는 방법으로 특성을 추가할 수 있습니다.

예시 코드 매개변수 col을 사용해서 그래프 열 방향으로 분할하기

|  |
| --- |
| import matplotlib.pyplot as plt import seaborn as sns  tips = sns.load\_dataset("tips") sns.lmplot(x='total\_bill', y='tip', data=tips, fit\_reg=False, hue='sex', col = 'time') plt.show() |

[실행 결과]

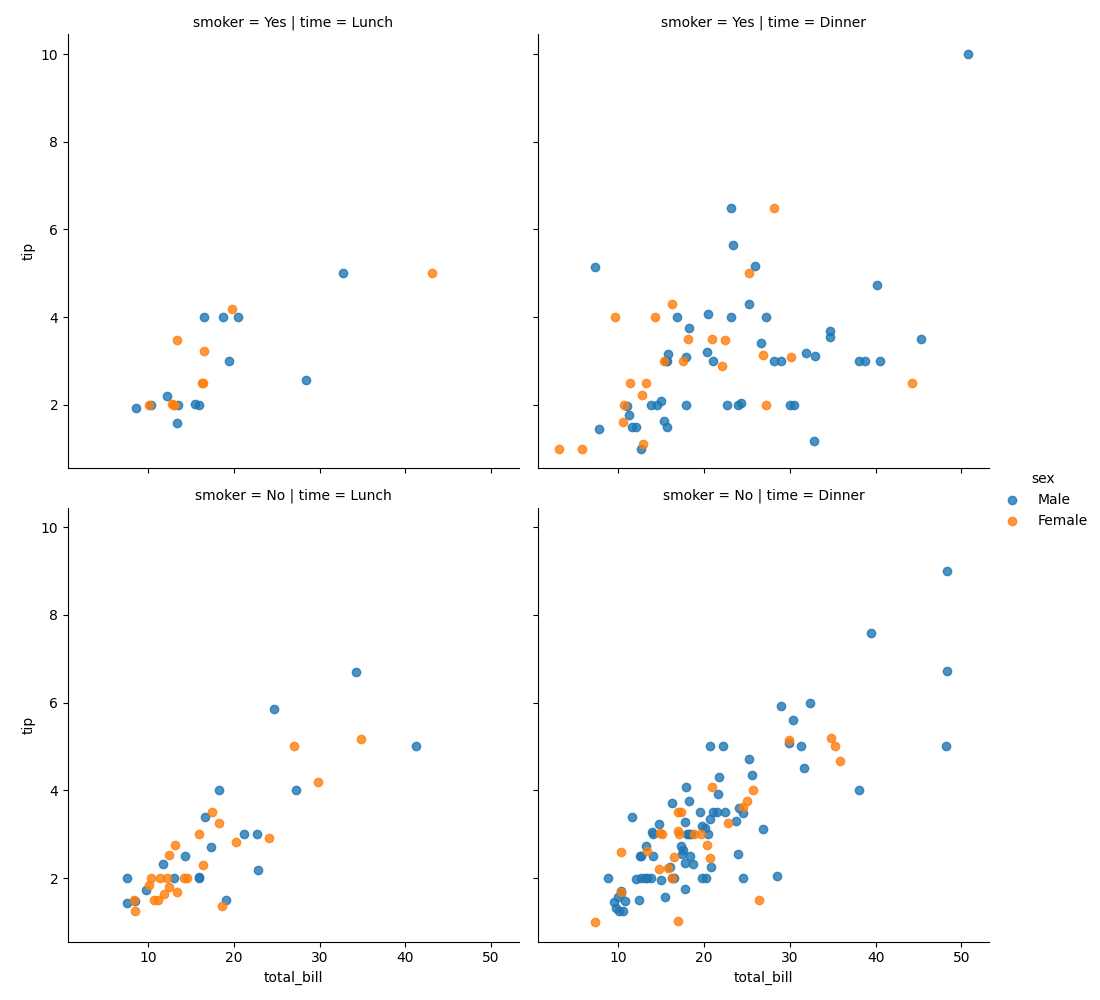


위의 결과처럼 특성마다 그래프를 열 방향으로 분할해서 특성을 추가할 수 있습니다. 그렇다면 2차원 그래프에 5가지 특성을 표현하려면 어떻게 해야 할까요? 열 방향에 그래프를 분할해서 그린 것처럼 행 방향으로 데이터를 분할해서 그리면 됩니다.

예시 코드 매개변수 row를 활용해서 행 방향으로 그래프 분할하기

|  |
| --- |
| import matplotlib.pyplot as plt import seaborn as sns  tips = sns.load\_dataset("tips") sns.lmplot(x='total\_bill', y='tip', data=tips, fit\_reg=False, hue='sex', col = 'time', row = 'smoker') plt.show() |

[실행 결과]



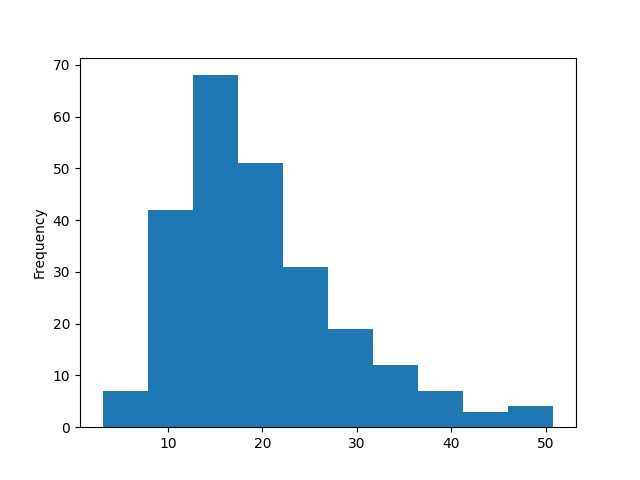
### pandas 모듈을 활용해서 그래프 그리기

우리가 데이터 분석에 활용하는 판다스 모듈을 통해서도 그래프를 그릴 수 있습니다. 판다스 모듈을 사용해서 그래프를 그릴 때는 주로 간단한 그래프를 그릴 때 사용합니다. 이번에는 판다스 모듈을 사용해서 몇 가지 그래프를 그려보겠습니다. 이번에도 seaborn 모듈에서 제공하는 tips 데이터를 사용해서 그래프를 그려봅시다.

예시 코드 시리즈를 활용해서 히스토그램 그래프 그리기

|  |
| --- |
| import matplotlib.pyplot as plt import seaborn as sns import pandas as pd  tips = sns.load\_dataset("tips") tips['total\_bill'].plot.hist() plt.show() |

[실행 결과]

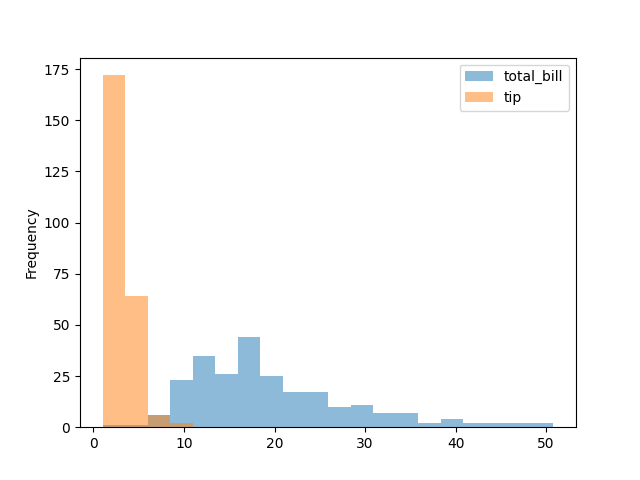


데이터프레임의 특정 시리즈에 접근해서 시리즈 뒤에 점(.)을 찍고, 그래프 함수 이름을 적어서 그래프를 그릴 수 있습니다. 한 개의 시리즈 뿐만 아니라 여러 개의 시리즈를 통해서 위와 같이 히스토그램을 그릴 수도 있습니다.

예시 코드 여러 개의 시리즈 값을 히스토그램으로 그리기

|  |
| --- |
| import matplotlib.pyplot as plt import seaborn as sns import pandas as pd  tips = sns.load\_dataset("tips") tips[['total\_bill', 'tip']].plot.hist(bins=20, alpha = 0.5) plt.show() |

[실행 결과]



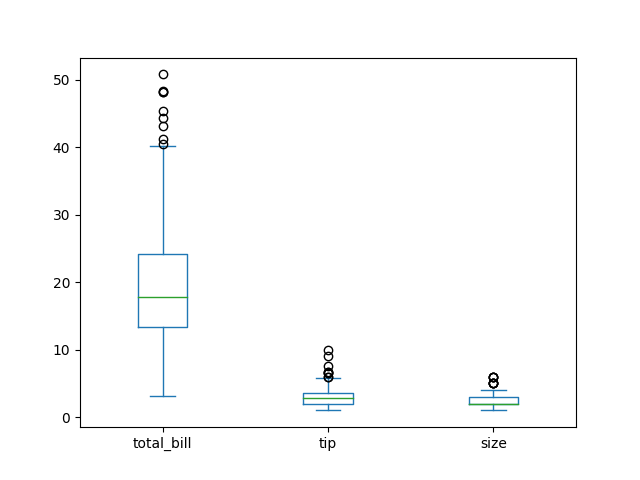
판다스를 통해 히스토그램을 그릴 때, 매개변수로 bins값을 설정하면 히스토그램에 표현되는 막대기 개수를 표현할 수 있으며, alpha값으로 투명도를 설정할 수도 있습니다.

특정 시리즈를 선택해서 그래프를 그릴 수 있는 것뿐만 아니라 데이터프레임 그 자체에서 그림을 그릴 수도 있습니다.

예시 코드 데이터프레임으로 박스 그래프 그리기

|  |
| --- |
| import matplotlib.pyplot as plt import seaborn as sns import pandas as pd  tips = sns.load\_dataset("tips") tips.plot.box() plt.show() |

[실행 결과]



간단하게 판다스 모듈을 통해서 시리즈를 활용한 히스토그램, 데이터프레임을 활용한 박스 그래프를 그려보았습니다. 판다스 모듈은 그 외에 다양한 그래프를 그릴 수 있는 함수들을 제공하고 있습니다. 판다스 모듈을 통해서 그래프를 그릴 수 있는 자세한 내용을 알고 싶은 독자가 있다면, 판다스 공식 사이트(<https://pandas.pydata.org/docs/reference/index.html>)에 가보면 자세하게 코드 예시와 그림이 나와있으므로 관심있으신 분들은 참고해주세요.

# 데이터 크롤링

데이터 크롤링이란 원하는 데이터들을 수집하는 기술을 의미합니다. 인터넷에 있는 정보를 손으로 하나하나 적는 것보다는 마우스로 하나하나 복사해서 붙여넣기 하는 방식으로 데이터를 가져오는 것이 더 좋을 것이고 더 나아가서 자동으로 컴퓨터가 가져와서 예쁘게 엑셀로 정리해주면 훨씬 더 작업이 편해질 것입니다.

그런데 인터넷은 파이썬 프로그래밍 언어로 되어있지 않습니다. 웹은 여러 가지 프로그래밍 언어가 결합되어 있습니다. 하지만 크게 봤을 때는 HTML이라는 언어로 되어있습니다. 그럼 우리가 파이썬 말고도 HTML까지 더 배워야 할까요? 다행히 그렇지 않습니다. HTML 문법만 약간 이해하면 됩니다. 파이썬 라이브러리에는 다양한 모듈들이 존재합니다. 그 중 파이썬으로 HTML을 다루는 모듈도 존재합니다. 파이썬으로 웹과 HTML을 다루는 다양한 모듈을 소개하면서 파이썬으로 데이터 크롤링을 하면 얼마나 쉽게 할 수 있는지 알아봅시다.

먼저 네이버 금융의 시가총액 데이터프레임을 크롤링해보겠습니다. 네이버에 접속해주신 후에 네이버 검색창에 “네이버 금융”을 검색해주세요.

텍스트이(가) 표시된 사진

자동 생성된 설명

그림 네이버 - 네이버금융 검색

화면 맨 위에 보이는 “네이버 증권”을 클릭해서 네이버 증권 페이지에 접속합니다. 그 후에 페이지가 나오면 화면 상단에 국내증시를 클릭합니다.



그림 국내증시 클릭

국내 증시 페이지에 접속하면, 왼쪽 탭에 보이는 “시가 총액”을 클릭합니다.



그림 시가총액 클릭

시가 총액 페이지에 접속하면 화면 가운데에 표가 보일 것입니다. 표에는 코스피 또는 코스닥에 대해서 시가총액이 1순위부터 오름차순으로 정리되어 있습니다.

테이블이(가) 표시된 사진

자동 생성된 설명

그림 시가총액

시가 총액 페이지는 코스피, 코스닥에 대해서 시가총액이 순위별로 나와서 보기 편하지만 우리 마음대로 필터링을 해서 볼 수는 없고 검색 또한 할 수가 없다는 단점이 있습니다. 하지만 이 데이터들을 모두 가져와서 엑셀로 표현한다면 쉽게 필터링할 수 있고 우리가 보고싶은 주식도 검색해서 볼 수 있겠죠?

이 페이지를 보면 각 기업마다 확인 가능한 항목은 20개가 넘지만 동시에 확인 가능한 항목은 최대 여섯 개입니다. 이 정보들은 내려 받을 수 없는 제약이 있지만 우리는 파이썬 데이터 크롤링을 통하여 제공되는 모든 항목을 얻어오겠습니다. 우리가 웹 브라우저에 접속하여 웹 페이지 정보를 눈으로 볼 수 있는 것처럼 파이썬 코드로 해당 웹 페이지를 정보로 받아올 수 있습니다. 웹 크롤링을 하는데 있어서 반드시 필요한 모듈은 아래 2가지입니다.

1. requests
2. BeautifulSoup

requests 모듈은 파이썬을 통해서 웹에게 페이지를 보여 달라고 요청할 수 있는 라이브러리입니다. 그리고 BeautifulSoup모듈은 파이썬을 통해 HTML을 제어할 수 있습니다. 현재 우리가 접속하고 있는 시가 총액 페이지의 웹 주소는 웹 주소창에 나와있습니다.

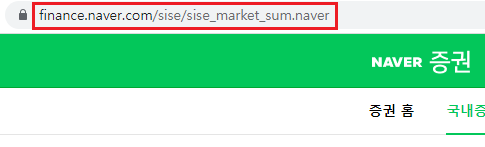


그림 시가총액 페이지 주소

해당 페이지 주소를 파이썬을 통해서 불러와보도록 하겠습니다.

예시 코드 페이지 요청하기

|  |
| --- |
| from bs4 import BeautifulSoup  import requests  import pandas as pd  import time  url = requests.get("https://finance.naver.com/sise/sise\_market\_sum.naver")  url |

[실행 결과]

<Response [200]>

requests 모듈의 get() 함수에 요청하고자 하는 페이지 주소를 문자열로 입력해주면 해당 페이지 정보를 받아올 수 있습니다. 받아온 정보는 변수 url에 할당했습니다. 그리고 url을 출력해주면 <Response [200]>이 출력되는 것을 볼 수 있습니다. 이 의미는 정상적으로 잘 가져왔다고 이해해주시면 됩니다. 그 밖에 400번대가 나오면 존재하지 않는 페이지를 가져왔다는 의미이며, 500번대가 나오면 로그인이 필요하거나 비공개인 페이지에 접속했다는 의미라고 이해해주시면 됩니다.

페이지 정보는 가져온 url변수의 text를 실행하면 페이지 정보를 볼 수 있습니다.

예시 코드 페이지 정보 보기

|  |
| --- |
| from bs4 import BeautifulSoup  import requests  import pandas as pd  import time  url = requests.get("https://finance.naver.com/sise/sise\_market\_sum.naver")  url.text |

[실행 결과]

텍스트이(가) 표시된 사진

자동 생성된 설명

가져온 정보를 보면 외계어 같은 이상한 문장들이 굉장히 지저분하게 출력됩니다. 이것이 HTML코드입니다. 우리가 항상 보고 있는 웹 사이트는 이러한 문장들이 모여 예쁘게 표현되는 것입니다. 하지만 실행 결과를 보면 출력된 HTML정보는 문자열로 이루어져 있습니다. 이 문자열로 이루어진 정보를 HTML로 변환해야하는데 그 역할은 bs4모듈 안에 있는 BeautifulSoup() 함수가 해줍니다. BeautifulSoup() 함수를 사용해서 가져온 정보를 HTML로 변환해보겠습니다.

예시 코드 페이지 정보 HTML로 변환하기

|  |
| --- |
| from bs4 import BeautifulSoup  import requests  import pandas as pd  import time  url = requests.get("https://finance.naver.com/sise/sise\_market\_sum.naver")  html = BeautifulSoup(url.text)  html |

[실행 결과]

텍스트이(가) 표시된 사진

자동 생성된 설명

실행 결과를 보면 크게 달라진 것을 잘 못 느끼실 수도 있습니다. 하지만 자세히 보면 문자열로 이루어져 있지 않은 것을 알 수 있습니다. (맨 앞과 뒤에 따옴표가 없습니다.) HTML로 잘 변환되었다고 보시면 됩니다. 이제부터 파이썬으로 자유롭게 HTML을 제어할 수 있게 된 것이죠.

크롤링으로 얻어 온 웹 페이지 정보는 HTML 형태 그대로 보이기 때문에 <p>, <a> 등 태그와 글자가 섞여 있습니다. (HTML에서 <> 문자로 되어있는 정보를 태그라고 합니다. 태그 안에 맨 앞에 있는 문자를 “태그 명”이라고 하며, 그 뒤에 class나 id등 값들이 적혀 있을 때가 있는데 이 것들은 태그의 “속성 값” 입니다. 여기서 우리는 수집하고 싶은 데이터가 있는 요소를 추출하여 필요한 데이터를 얻어 올 수 있습니다. 그렇다면 추출하고 싶은 요소를 확인하는 방법을 알아보겠습니다. (필자는 구글 크롬을 기준으로 했습니다.) 추출하고 싶은 정보가 있는 곳에 마우스를 갖다 대고 마우스 오른쪽 버튼을 누른 뒤 맨 아래에 있는 “검사” 버튼을 누릅니다.

테이블이(가) 표시된 사진

자동 생성된 설명

그림 마우른 오른쪽 버튼 – 검사

“검사” 버튼을 누르면 웹 사이트 화면 오른쪽에 해당 웹 페이지의 HTML 정보를 볼 수 있습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

그림 시가총액 페이지 HTML정보

시가 총액 페이지에서 시가 총액이 나와있는 표를 긁어오려면 표가 있는 곳의 HTML 태그명과 속성 값을 찾으면 됩니다. 그러면 가져오고자 하는 표에 마우스를 대고 검사 버튼을 눌러서 표의 태그명과 속성 값을 찾아보겠습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

그림 시가 총액 표 HTML 정보

시가 총액 표의 HTML 정보를 찾아보니 태그 명은 “table” 이며 class 속성 값은 “type\_2” 인 것을 알 수 있습니다. 그럼 우리가 가져온 변수 html에 해당하는 태그와 속성 값에 접근해보겠습니다. 원하는 태그와 속성에 접근할 때는 html의 find() 함수를 사용하면 됩니다. 함수 안에 태그 명과 class\_ 매개변수에 속성 값을 입력해줍니다.

예시 코드 시가총액 표 데이터 가져오기

|  |
| --- |
| from bs4 import BeautifulSoup  import requests  import pandas as pd  import time  url = requests.get("https://finance.naver.com/sise/sise\_market\_sum.naver")  html = BeautifulSoup(url.text)  html.find('table', class\_ = 'type\_2') |

[실행 결과]

텍스트이(가) 표시된 사진

자동 생성된 설명

맨 앞에 있는 태그 명이 “table” 인 것을 보아 해당하는 정보를 잘 가져왔습니다. 이제 이 표를 데이터프레임으로 변환해보겠습니다. 가져온 html 정보를 데이터프레임으로 변환하려면 판다스 모듈의 read\_html() 함수를 사용해서 문자열로 해당 정보를 입력해주면 됩니다.

예시 코드 read\_html() 함수로 데이터프레임 만들기

|  |
| --- |
| from bs4 import BeautifulSoup  import requests  import pandas as pd  import time  url = requests.get("https://finance.naver.com/sise/sise\_market\_sum.naver")  html = BeautifulSoup(url.text)  table = html.find('table', class\_ = 'type\_2')  table = pd.read\_html(str(table))[0]  table |

[실행 결과]

테이블이(가) 표시된 사진

자동 생성된 설명

판다스 모듈의 read\_html() 함수는 읽어온 HTML을 정보를 데이터프레임으로 변환해서 리스트로 반환해줍니다. 따라서 리스트 안에 있는 정보를 가져오기 위해서 리스트의 0번째에 있는 데이터프레임을 꺼내서 다시 table 변수에 넣어주었습니다. 그리고 출력해보니 예쁘게 데이터 프레임으로 출력하는 것을 볼 수 있습니다. 그런데 중간중간에 NaN으로 누락 값이 많이 있습니다. 누락 값은 당연히 존재할 수 있습니다. 하지만 데이터프레임에 종목명에 있는 값에는 누락 값이 존재하면 안 됩니다. 종목명이 누락 값이 아닌 행들만 가져오도록 하겠습니다.

예시 코드 종목명이 누락값이 아닌 행들만 가져오기

|  |
| --- |
| from bs4 import BeautifulSoup  import requests  import pandas as pd  import time  url = requests.get("https://finance.naver.com/sise/sise\_market\_sum.naver")  html = BeautifulSoup(url.text)  table = html.find('table', class\_ = 'type\_2')  table = pd.read\_html(str(table))[0]  table = table[table['종목명'].notnull()]  table |

[실행 결과]

테이블이(가) 표시된 사진

자동 생성된 설명

데이터프레임이 담긴 변수 table에 접근해서 table의 “종목 명” 열에 있는 값 중에서 notnull() 함수를 통해 누락 값이 아닌 행에만 접근했습니다. 결과를 보면 종목명에는 누락 값이 존재하지 않습니다.

표를 깔끔하게 가져오는 방법을 알았으니, 모든 코스피와, 코스닥 정보를 표로 가져와서 하나로 합쳐보겠습니다. 가장 먼저 해야 할 것은 코스피가 총 몇 페이지로 구성되어 있는지, 코스닥이 총 몇 페이지로 구성되어 있는지 알아야 합니다. 밑에 페이지 번호들을 누르면 위 주소창에 페이지에 대한 정보가 추가가 되며, 표 상단에 코스피 또는 코스닥 버튼을 클릭하면 역시 주소 창에 코스피, 코스닥에 대한 정보가 주소 창에 추가됩니다.

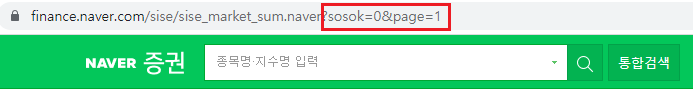


그림 주소 창에 페이지정보와 소속 정보가 추가된 모습

확인해 본 결과 sosok이 0이면 코스피, 1이면 코스닥인 것을 알 수 있습니다. 그리고 페이지 정보도 추가되었으니 for문을 사용하면 쉽게 모든 페이지에 접근할 수 있겠군요.

이번에는 코스피, 코스닥에 가장 마지막 페이지가 몇 페이지인지 정보를 가져오도록 하겠습니다. 페이지 밑에 페이지 번호가 있는 곳에서 “맨 뒤” 버튼을 마우스 오른쪽 버튼을 통해 “검사” 버튼을 눌러 태그명과 속성 값을 찾아보겠습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

그림 "맨 뒤" 버튼 HTML 정보

“맨 뒤” 버튼의 HTML 정보를 보니 “td” 태그 명의 class 속성 값이 “pgRR” 이며, 그 안에 “a” 태그 명에 우리가 찾고 있는 정보가 있는 것을 알 수 있습니다. 코스피 페이지의 가장 맨 뒷 페이지는 40페이지 라는 것을 알 수 있습니다. 해당 태그에 접근해서 40이라는 숫자를 가져오겠습니다.

예시 코드 "맨 뒤" HTML 정보에 접근하기

|  |
| --- |
| url = requests.get("https://finance.naver.com/sise/sise\_market\_sum.naver?sosok=0&page=1")  html = BeautifulSoup(url.text)  html.find("td", class\_ = 'pgRR').find('a') |

[실행 결과]

<a href="/sise/sise\_market\_sum.naver?sosok=0&amp;page=40">맨뒤

<img alt="" border="0" height="5" src="https://ssl.pstatic.net/static/n/cmn/bu\_pgarRR.gif" width="8"/>

</a>

“맨 뒤” 버튼의 HTML 정보를 가져왔습니다. 이제는 숫자 40을 가져와야 합니다. 숫자 40이 들어있는 곳의 정보는 a 태그의 href 라는 곳 안에 있는 것을 알 수 있습니다. 속성 값에 접근할 때는 인덱스와 마찬가지로 대괄호를 사용해서 접근해주면 됩니다.

예시 코드 맨 뒷 페이지 정보 가져오기1

|  |
| --- |
| url = requests.get("https://finance.naver.com/sise/sise\_market\_sum.naver?sosok=0&page=1")  html = BeautifulSoup(url.text)  kospi\_page = html.find("td", class\_ = 'pgRR').find('a')['href']  kospi\_page |

[실행 결과]

'/sise/sise\_market\_sum.naver?sosok=0&page=40'

맨 뒤에 있는 숫자를 가져오기 위해서는 “=”를 기준으로 문자열을 분할해서 가장 뒤에 있는 정보를 가져오는 것이 가장 편리할 것 같습니다.

예시 코드 맨 뒷 페이지 정보 가져오기2

|  |
| --- |
| url = requests.get("https://finance.naver.com/sise/sise\_market\_sum.naver?sosok=0&page=1")  html = BeautifulSoup(url.text)  kospi\_page = int(html.find("td", class\_ = 'pgRR').find('a')['href'].split("=")[-1])  kospi\_page |

[실행 결과]

40

split() 함수를 사용해서 “=”를 기준으로 문자열 데이터를 분할하면 리스트로 분할된 정보를 보여주는데 가장 뒤에 페이지 번호가 있는 것을 알 수 있습니다. 여기서 가져온 페이지 번호는 문자열이기 때문에 int() 함수를 사용해서 정수로 바꿔준 후에 kospi\_page변수에 할당했습니다.

그렇다면 이제 모든 코스피, 코스닥을 데이터프레임으로 가져와서 하나의 데이터프레임으로 합쳐보겠습니다.

예시 코드 모든 코스피 코스닥 정보 가져와서 데이터프레임으로 만들기

|  |
| --- |
| from tqdm import tqdm  total = []  url = requests.get("https://finance.naver.com/sise/sise\_market\_sum.naver?sosok=0&page=1")  html = BeautifulSoup(url.text)  kospi\_page = int(html.find("td", class\_ = 'pgRR').find('a')['href'].split("=")[-1])  for n in tqdm(range(1, kospi\_page + 1)):  url = requests.get("https://finance.naver.com/sise/sise\_market\_sum.naver?sosok=0&page={}".format(n))  html = BeautifulSoup(url.text)  table = html.find('table', class\_ = 'type\_2')  table = pd.read\_html(str(table))[0]  table = table[table['종목명'].notnull()]  table['소속'] = "코스피"  total.append(table)  time.sleep(1)    url = requests.get("https://finance.naver.com/sise/sise\_market\_sum.naver?sosok=1&page=1")  html = BeautifulSoup(url.text)  kosdaq\_page = int(html.find("td", class\_ = 'pgRR').find('a')['href'].split("=")[-1])  for n in tqdm(range(1, kosdaq\_page + 1)):  url = requests.get("https://finance.naver.com/sise/sise\_market\_sum.naver?sosok=1&page={}".format(n))  html = BeautifulSoup(url.text)  table = html.find('table', class\_ = 'type\_2')  table = pd.read\_html(str(table))[0]  table = table[table['종목명'].notnull()]  table['소속'] = "코스닥"  total.append(table)  time.sleep(1)    stock = pd.concat(total, ignore\_index=True)  del stock['N']  del stock['토론실']  stock |

[실행 결과]

테이블이(가) 표시된 사진

자동 생성된 설명

# 머신러닝

머신러닝(Machine Learning)은 말 그대로 학습하는 기계를 말합니다. 데이터가 충분히 있지만 데이터의 특징을 찾거나 패턴을 찾기가 힘들 때 머신러닝은 큰 효과를 발휘합니다.

머신러닝이 인공지능과 같다고 생각하는 사람들이 많습니다. 우리가 흔히 생각하는 인공지능은 영화 “아이언 맨”에 나오는 “자비스” 라는 캐릭터나 또는 영화 “Her”에 나오는 연애OS를 주로 떠올릴 것입니다. 필자 생각에는 머지않아 정말로 영화에 나왔던 인공지능이 현실화될 것이라고 생각합니다.

## 인공지능, 머신러닝, 딥러닝의 관계

우리가 상상하는 폭넓은 개념을 인공지능이라고 한다면, 그 중 컴퓨터가 데이터를 보고 스스로 학습하고 판단하는 부분을 머신러닝이라고 합니다. 즉 인공지능이라는 개념 안에 일부가 머신러닝인것이죠.

텍스트이(가) 표시된 사진

자동 생성된 설명

그림 인공지능, 머신러닝, 딥러닝의 관계 (그림 출처: 네이버 블로그, LEE&JANG ECO ENG)

그리고 아마 딥러닝이라는 단어를 들어 보신 분들이 많으실 겁니다. 머신러닝과 딥러닝을 따로 보시는 분들도 많습니다. 머신러닝에는 다양한 통계 방법을 사용하는 방식 또는 사람과 같은 신경망 형태를 본따 만든 방식 등 다양한 모델들이 있습니다. 딥러닝은 다양한 머신러닝 방법들 중에 하나라고 보시면 됩니다. 즉 머신러닝의 일부가 딥러닝인것이죠. 또는 딥러닝이 머신러닝 모델들 중 가장 성능이 좋다고 생각하시는 분들도 많은데, 이는 반은 맞고, 반은 틀립니다.

## 정형데이터와 비정형데이터

테이블이(가) 표시된 사진

자동 생성된 설명

그림 정형데이터와 비정형데이터

데이터에는 크게 2가지 종류가 있습니다. 정형 데이터와 비정형 데이터가 있습니다. 정형 데이터는 표(데이터 프레임) 형태로 되어있는 정형화된 데이터들을 말합니다. 비정형 데이터는 위의 그림처럼 이미지, 텍스트, 음성과 같은 정형화 되어있지 않은 데이터들이 비정형 데이터입니다. 딥러닝은 비정형 데이터에서는 정말 좋은 성능을 보여줍니다. 현재 존재하는 머신러닝 모델들 중 압도적으로 성능이 좋다고 할 수 있습니다. 하지만 정형 데이터에서는 딥러닝이 그닥 성능을 좋게 보여주는 경우는 많이 없습니다. 오히려 다른 통계적인 방법을 사용한 머신러닝 모델이 더 큰 성능을 보일 때가 많습니다. 따라서 머신러닝 모델을 선택할 때 무조건 딥러닝을 선택하면 안 됩니다. 데이터의 형태를 보고 적합한 모델을 찾아야 합니다.

## 인공지능을 배워야 하는 이유

정확하게 말하면 사실 머신러닝을 배워야하는 이유이지만, 대부분 인공지능이라고 표현하니 인공지능을 배워야하는 이유라고 표현하겠습니다. 왜 인공지능을 배워야 할까요? 앞으로 우리는 인공지능으로 인해서 할 수 있는 일의 가짓수가 늘어날 것이기 때문입니다. 필자는 초등학생 시절부터 선생님에게 항상 들었던 이야기가 있습니다. (참고로 필자는 92년생입니다.)

“미래에는 컴퓨터가 발전하고 보급되며, 태블릿PC라는 것이 등장하면서 종이가 필요 없어지고 기술이 환경을 보호해줄거야!”

필자도 선생님의 그 말씀을 듣고 동의했습니다. 그 당시에는 아날로그에서 디지털로 넘어가는 시대여서 디지털이 빠르게 발전하고 있었고, 액정을 통해 글을 읽는 사람들이 점차 늘어나고 있었습니다. 아마 대부분도 저와 같은 생각이실 겁니다. 그런데 정말 지금 종이가 필요가 없어졌나요? 또는 종이 사용량이 정말 줄었을까요?

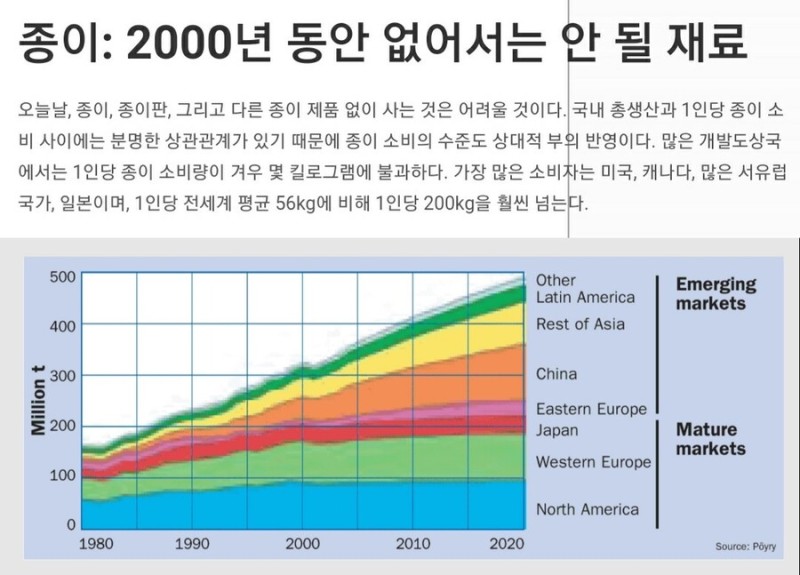


그림 종이 소비량

위 그림은 1980년에서 2020년도까지의 종이 소비량을 측정한 것입니다. 분명히 빠르게 디지털 기술이 발전하고 있는데, 그 만큼 종이 소비량도 급격하게 오르고 있습니다. 왜 이럴까요? 왜냐하면 사람이 할 수 있는 일의 가짓수가 증가했기 때문입니다. 예를 들어 한 사람이 하루에 10가지 일을 할 수 있고, 1가지 일을 하는데 종이가 1장 필요하다고 가정해보겠습니다. 그렇다면 이 사람은 하루에 필요한 종이는 10장이 될 것입니다. 그리고 컴퓨터가 보급되면서 컴퓨터로 인해서 종이 소비량이 90%나 줄었고 사람이 할 수 있는 일의 가짓수가 200가지로 늘었다고 고 또 가정해보겠습니다. 그렇다면 컴퓨터 덕분에 하루에 10가지 일을 할 수 있는 사람이 하루에 200가지 일을 할 수 있게 되었고, 아무리 종이 소비량을 90%나 줄였다고 해도 200가지 일의 10%는 20장입니다. 즉 10장의 종이가 필요했던 사람은 이제 20장의 종이가 필요하게 된 것이죠. 즉 기술이 발전하면 발전할 수록 사람이 할 수 있는 일의 가짓수가 증가하고 있다는 것을 추론해볼 수 있습니다. 이제는 4차 산업혁명 시대이고 핵심은 인공지능입니다. 인공지능으로 인해서 우리는 앞으로 인공지능 덕분에 할 수 있는 일의 가짓수는 또 크게 늘어날 것입니다.

텍스트, 선화이(가) 표시된 사진

자동 생성된 설명

그림 러다이트 운동

1차 산업혁명 기계의 등장으로 인해 기계 장치가 산업 현장에 본격적으로 도입되면서 기계가 사람의 일자리를 빼앗는 것처럼 보였습니다. 위의 그림은 1차 산업혁명 기계의 등장으로 일자리를 잃어버린 노동자들이 일자리를 뺏기 위해서 벌였던 “러다이트” 라는 시위 운동입니다. 하지만 기계를 부순다고 해서 일자리를 다시 되찾을 수 있을까요? 기술의 발전은 모두 알고 있겠지만 절대로 막을 수가 없습니다. 결국 지금 우리는 기계와 아주 밀접하게 함께 살아가고 있으니까요. 하지만 기계가 일자리를 없애기만 한 것은 아닙니다. 사람들이 기계에 대해서 공부하고 이해하기 시작하면서 기계로 인해서 새로운 일자리들도 많이 생겨났습니다. 반대로 기계에 대해서 공부하지 않고 이해하지 않으려고 했던 사람들은 자연스럽게 도태되었습니다. 여기서 기계에 대해서 공부하고 이해했다는 것은 전문가 수준으로 공부했다는 것이 아닙니다. 우리도 TV를 보다가 리모컨이 작동하지 않으면 건전지를 바꿔야 한다는 사실을 압니다. 스마트 폰 배터리가 부족하면 충전 케이블을 꽂아야 한다는 것도 알고 있습니다. 이 모든 것이 우리가 기계에 대해서 어느정도 이해를 하고 있기 때문에 가능한 것입니다. 만약 아프리카 원주민에게 스마트 폰을 충전해달라고 한다면 이해할 수 있을까요? 그 뒤에 3차 산업혁명 컴퓨터가 등장하면서 뉴스에 컴퓨터로 인해서 일자리가 사라지고 있다는 뉴스를 항상 나왔습니다. 실제로 컴퓨터로 인해서 많은 일자리들이 사라졌었죠. 하지만 컴퓨터가 일자리를 사라지게만 한 것은 아닙니다. 우리가 인터넷 검색을 통해서 지식을 습득할 수 있게 되었고, 파워포인트나 워드를 통해서 보고서를 쓸 수 있게 되는 등 컴퓨터에 대해 공부하고 이해하기 시작하면서 새로운 일자리들도 많이 생겨났습니다. 반대로 컴퓨터에 대해 공부하지 않은 사람들은 자연스럽게 도태되었습니다. 만약 인터넷 검색을 통해 자료를 얻지 못하고 한글이나 워드를 통한 보고서를 작성하는 방법도 모른다면 받아주는 회사가 있을까요? 없을 것입니다. 이제는 4차 산업혁명 시대입니다. 4차 산업혁명 시대의 핵심은 인공지능입니다. 그렇다면 역사적으로 봤을 때 우리가 인공지능에 대해서 공부하고 이해하려는 노력을 하지 않는다면 자연스럽게 도태될 것입니다. 따라서 이제는 인공지능에 대해서 관심이 없다고 하더라도 미래에 대해 적응하고 대비하기 위해서 인공지능에 대해서 어느정도 공부하고 이해해야 합니다. 이 역시 우리가 인공지능에 대해서 전문가가 될 정도로 공부해야 한다는 의미는 아닙니다. 인공지능을 어느정도 다룰 수 있고 인공지능을 사용해서 무언가를 예측할 수 있는 정도라면 충분하다고 생각합니다.

## IT와 DT

IT(Information Technology)와 컴퓨터를 동일시하는 사람들이 많습니다. 아래 그림은 컴퓨터 공학과 IT, DT의 범위를 구분해 놓은 그림입니다.

텍스트, iPod이(가) 표시된 사진

자동 생성된 설명

그림 컴퓨터 공학과 IT와 DT의 관계

여기서 A부터 E에 대한 내용은 아래와 같습니다.

1. 자동제어, 통신, 전기공학
2. 빅 데이터, 인공지능
3. PC, 스마트 폰 등
4. 인쇄술, 도서관 자료 분류 체계
5. 컴퓨터 공학의 도움 없이 데이터를 처리하는 방법

자동제어, 통신, 전기 공학(A)은 컴퓨터 공학과 관련된 영역이지만 IT라고 할 수는 없습니다. 반면 도서관 분류 체계와 인쇄술(D)은 IT이지만 컴퓨터 공학이 존재하지 않던 때부터 있었던 영역입니다. 만약 도서관에서 아무 체계나 기준도 없이 책들이 비치되어 있다면 원하는 책을 찾는 일 자체가 불가능했을 것입니다. 문헌 정보학이라는 것이 컴퓨터가 존재하기 훨씬 전부터 책의 분류를 체계적으로 다루었습니다. 문헌 정보학이 했던 일이 바로 IT라고 할 수 있습니다. 컴퓨터가 없어도 그 역할을 다 할 수 있었기 때문입니다. 이렇게 IT라고 해서 모두 컴퓨터와 관련된 것은 아닙니다. 그리고 IT가 발전한 것을 DT(Data Technology)라고 합니다. DT는 현재 우리가 마주친 4차 산업혁명과 그리고 인공지능과 밀접하게 관련되어 있습니다. 그래서 인공지능을 이해하기 위해서는 먼저 빅 데이터가 무엇인지 알아야 하고, 빅 데이터를 이해하기 위해서는 DT에 대해서 이해해야 합니다. 먼저 DT는 IT가 발전한 형태를 말합니다. IT는 어떤 데이터가 있다고 가정했을 때, 전문가가 데이터로부터 의미 있는 정보를 선별해서 활용하는 것입니다. 3차 산업혁명 때까지는 전문가가 데이터로부터 의미 있는 정보를 선별해서 가공하고 체계적으로 관리해서 정보들을 활용했습니다. 그러다가 1990년대 후반부터 인터넷이 대중화되었고 2000년대부터 인터넷이 더 발전을 해서 사람들이 인터넷을 활용하게 되면서 제3자에게 정보를 공유하거나 글을 남기기도 하면서 데이터의 양이 폭발적으로 증가하게 됩니다. 그렇게 수집할 수 있는 데이터의 양이 증가하게 되면서 IT는 DT로 확대가 됩니다. 여기서 DT란 데이터를 전문가가 수집해서 의미 있는 정보를 선별하는 것이 아니라 데이터 자체를 모두 저장해서 활용하는 것이 DT입니다. 조금 더 이해하기 쉽게 설명해보자면 100개의 데이터가 발생했을 때 발생한 데이터로부터 의미 있는 정보를 10개 선별한다면 이 10개의 데이터는 information이라고 부릅니다. 하지만 information을 선별하지 않고 그 자체를 모두 저장하고 활용하는 기술을 DT라고 합니다. 이것이 가능해진 이유는 과거에 비해서 데이터 저장 장치가 아주 저렴해졌고, 클라우드 컴퓨팅의 대중화로 굳이 데이터를 선별할 필요가 없어졌기 때문입니다. 그리고 이렇게 저장한 데이터를 전문가가 분석하고 활용하는 것이 아니라 데이터 분석이나 인공지능을 활용해서 데이터를 분석하는 것이 DT입니다. 즉 IT와 DT의 차이는 인간의 판단력이 중심이 되느냐, 인간의 판단력에 대한 과신을 내려놓고 데이터 안에서 의미 있는 패턴을 찾느냐의 차이입니다.

## 빅 데이터

필자는 대학원에 입학하기 전에는 빅 데이터라는 것이 막연히 아주 많은 데이터를 의미하는 줄 알았습니다. 빅 데이터에 대한 흔한 오해는 데이터의 양이 많으면 빅 데이터이고, 그렇지 않으면 빅 데이터가 아니라고 생각한다는 것입니다. 빅 데이터는 양이 중요한 것이 아닙니다. 한가지 예를 들어보겠습니다.

제철소 용광로에 온도 센서가 있고, 이 온도 센서가 10분 마다 온도 값을 송출하는 경우를 생각해보겠습니다. 그러면 10분마다 1개의 온도 값을 측정하니 1시간에 6개의 온도 데이터가 수집될 것입니다. 그리고 하루에는 총 144개의 데이터가 수집될 것입니다. 여기서 이 144개의 온도 데이터를 information으로 선별하지 않고 모두 저장한다면 이 144개의 데이터는 빅 데이터라고 부릅니다. 반면 100억개의 데이터 중 10억개의 데이터를 선별했다면 이 데이터가 10억개나 되는 데이터라고 하더라도 information이며 빅 데이터가 아닙니다.

### 전문가보다 인공지능으로 분석해야 하는 이유

전문가가 데이터를 분석하는 것보다 인공지능으로 데이터를 분석해야 합니다. 인공지능이 데이터를 분석하는 이유에는 크게 2가지가 있습니다.

1. 사람의 판단으로 데이터를 처리하기에는 데이터의 양이 너무 많다.
2. 사람의 논리로 빅 데이터를 해석하면 일부의 데이터만 해석될 위험이 있다.

이제는 데이터의 양이 너무 많아서 사람이 분석하기에는 무리가 있으며, 전체 데이터를 두고 전문가가 의미 있는 데이터만 수집한다면 전문가가 미처 알아차리지 못한 중요한 데이터를 거를 가능성이 있기 때문입니다. 예를 들어보겠습니다.

고등학생의 성적과 관련된 요인을 연구하는 연구자가 있다고 가정해보겠습니다. 이 연구자는 성적의 관계를 찾아내기 위해 학생들의 데이터를 수집할 것입니다. 학생들의 수면시간, 통학 시간, 자율학습 시간, IQ 등 성적과 관련된 정보를 수집하려고 하겠죠. 그리고 학생의 신발 사이즈나 학생이 어떤 연예인을 좋아하는지에 대한 정보를 선별하지 않았을 가능성이 높습니다. 성적과 관련이 없다고 판단했기 때문입니다. 그런데 이것은 모르는 겁니다. 학생의 신발 사이즈가 성적에 영향을 크게 미칠 수도 있을 수도 있으니까요. 만약 정말로 학생의 신발 사이즈가 성적에 큰 영향을 미친다면 이 정보를 선별하지 않았을 때 그 만큼 정확한 성적을 예측하기에는 한계가 있을 것입니다. 이렇게 정보를 수집하는 IT는 이러한 가능성이 놓칠 위험이 있습니다. 하지만 데이터를 모두 수집해서 분석하는 DT는 이러한 모든 가능성을 놓치지 않는다는 장점이 있습니다.

## 인공지능과 관련된 학문과 기술 분야

인공지능이 컴퓨터 공학과만 관련되어 있다고 생각하는 사람도 아주 많습니다. 하지만 절대 그렇지 않습니다. 인공지능은 아주 다양한 학문과 긴밀하게 연결되어 있습니다.

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

그림 인공지능과 관련된 학문

위의 그림은 인공지능과 관련된 학문들입니다. 위에서 나온 학문들보다 훨씬 많은 학문들과 연결 되어있을 것입니다. 철학과 연결시켜보겠습니다. 미래에는 자동차를 사람이 운전하지 않고 인공지능이 운전하는 자율 자동차가 나올 것이라는 것은 대부분 예측하고 있을 것입니다. 자율 주행 기술이 언제쯤 완벽하게 완성이 되어서 상용화가 될까요? 사실 자율 주행 기술은 기술적으로만 보면, 상용화 될 정보로 세밀하고 완벽하게 구현이 되어있습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

그림 자율주행차 미국 횡단 성공(출처: 아이뉴스24)

위의 뉴스 기사는 미국의 자율주행 자동차가 미국 동서횡단에 성공했다는 뉴스입니다. 심지어 이 뉴스 기사는 2018년에 나왔습니다. 지금은 훨씬 더 안정화되었고 발전되었겠죠? 사실 자율 주행 자동차가 상용화되지 않고 있는 이유는 기술적인 문제보다 철학적인 문제 때문입니다. 사람 운전자와 달리 자율 주행 자동차는 각종 센서에서 입수한 정보를 인공지능이 순식간에 처리하므로 언제나 현재 처한 상황을 객관적으로 파악할 수 있습니다. 그럼에도 차가 움직이는 것은 물리적인 현상이기 때문에 사고를 피할 수 있는 이상적인 대응이 매번 가능하지는 않을 것입니다. 만약 반대편 차선에서 갑자기 차가 중앙선을 넘어온다던 지, 아이가 갑자기 도로로 뛰어든다던 지 하는 상황에서는 인공지능으로도 사고를 피하기 어려울 것입니다. 이러한 경우 사람 운전자는 상황을 온전히 파악하지 못한 채 사실상 반사행동이라고 볼 수 있는 대응을 하지만 자율 주행 자동차는 실행 가능한 선택 중에 가장 위험도 낮은 선택을 할 수 있습니다. 피해가 불가피한 경우 피해를 최소화하는 방향으로 결정하는 것이죠. 사람으로 치면 사고 직전의 상황이 슬로우 모션으로 돌아가 어떻게 사고를 내야 할지 판단할 시간이 인공지능에게는 있는 셈입니다.

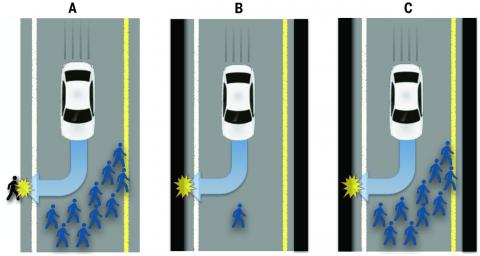


그림 자율주행차의 윤리적 딜레마(출처: The ScienceTimes)

위 그림을 보겠습니다. 연구자들은 인터넷을 통해 1,928명을 대상으로 자율 주행 자동차의 행동 지침에 대한 설문조사를 했는데, 전방에 사람 10명이 갑자기 나타났을 때 그대로 차가 직진하면 모두 사망할 것입니다. 이런 상황에서 사람들 다수는 공리주의에 따른 결정을 선호했습니다. 76%가 보행자 10명 대신 탑승자 1명을 희생하는 쪽이 더 도덕적이라고 판단한 것입니다. 사람들은 자율 주행 자동차의 인공지능이 이런 식으로 즉, 자기 희생 모드로 프로그래밍 되는 것이 바람직하다고 평가했습니다. 그런데 이를 법적으로 강제해야 한다는 지점에 이르자 동의하는 빈도가 뚝 떨어졌습니다. 그리고 자기 희생 모드와 자기보호 모드 두 가지로 차가 나올 경우 대다수는 후자인 모드를 가지고 있는 자율 자동차를 구매하겠다고 대답했습니다. 그러면서도 다른 사람들은 전자인 모드로 사야 한다고 대답했습니다. 차를 구매하는 입장에서는 사고가 날 경우 내가 생존하는 쪽으로 세팅이 되어야 한다는 거죠.

그렇다면 이런 상황에서 자율 주행 자동차의 인공지능이 어떻게 결정해야 할지를 정하는 주체는 누가 되어야 할까요? 사실 전자든 후자든 중요하지 않습니다. 중요한 것은 인공지능이 누구를 죽여야할지 선택한다는 것이 문제입니다. 인공지능이 누구를 죽여야할지 선택하는 부분에서 인간의 존엄성이 훼손될 수 있다는 점에서 철학적인 문제가 발생합니다. 아마 이러한 철학 문제는 간단히 해결될 문제가 아니라서 앞으로 자율 주행차의 상용화의 큰 걸림돌이 될 것이라고 생각됩니다.



그림 인공지능 기술 분야

인공지능을 활용할 수 있는 기술적인 분야도 다양하게 존재합니다. 우리가 흔히 알고 있는 삼성의 스마트 폰 빅스비, 구글의 구글 어시스턴트, 애플의 시리 등 인간의 언어를 학습시켜서 활용하는 자연어 처리, 확률적 추론을 통한 최적화 모델, 검색, 의사 결정 등 인공지능을 활용한 기술적인 분야가 다양합니다. 따라서 누군가가 만약 인공지능 전문가라고 한다면 사실 의심의 눈초리로 볼 필요가 있습니다. 따라서 인공지능을 배울 때도 정확히 어떤 분야를 공부하고 싶은 지 미리 방향을 설정하고 단계적으로 배우는 것이 좋습니다.

# 머신 러닝

머신 러닝의 개념은 다양하게 표현할 수 있으나 일반적으로는 애플리케이션을 수행하지 않고도 데이터를 기반으로 패턴을 학습하고 결과를 예측하는 알고리즘 기법을 통칭합니다. 현실 세계의 매우 복잡한 조건으로 인해 기존의 소프트웨어 코드만으로는 해결하기 어려웠던 많은 문제점들을 이제 머신 러닝을 이용해 해결해 나가고 있습니다. 가령 금융 사기 거래를 적발하는 프로그램을 만든다고 가정해보겠습니다. 복잡한 금융 거래에서 발생하는 수많은 변수에 대해 수십 년에 걸쳐 발생한 다양한 금융 사기 거래 조건을 감안해 수천~수만 라인의 소스 코드로 된 프로그램을 작성하더라도 금융 사기 전문가들은 경험을 통해 교묘하게 이 로직을 뚫어냅니다. 또한 금융 환경, 정부 정책, 소비자 성향 등이 수시로 변하기 때문에 이러한 외부 조건에 맞춰서 기존 로직을 다시 수정하고 검증하는 프로세스는 많은 시간과 비용이 요구됩니다. 프로그램 로직을 통해 모든 조건과 다양한 환경 변수, 규칙을 반영해 사기 거래 적발의 정확성을 높일 수 있다면 이러한 시간과 비용의 희생을 감당할 수 있겠지만, 기대와는 다르게 오히려 소스 코드가 복잡해지면서 예측의 정확성 향상이 이뤄지지 않는 경우가 현실입니다. 업무적으로 복잡한 조건, 규칙들이 다양한 형태로 결합하고 시시각각 변하면서 도저히 소프트웨어 코드로 로직을 구성하여 이들을 관통하는 일정한 패턴을 찾기 어려운 경우에 머신 러닝은 훌륭한 솔루션을 제공합니다. 가령 스팸 메일 필터링 프로그램을 만든다고 가정해보겠습니다. 단순히 특정 단어가 메일 내용에 포함되어 있다고 이를 스팸 메일로 분류할 수는 없습니다. 언어란 문맥에 의해 판단해야 하므로 인간의 언어에서 이러한 패턴을 규정하기란 매우 어렵습니다. 머신 러닝은 이러한 문제를 데이터를 기반으로 숨겨진 패턴을 인지해 해결합니다. 머신 러닝 알고리즘은 데이터를 기반으로 통계적인 신뢰도를 강화하고 예측 오류를 최소화하기 위한 다양한 수학적 기법을 적용해 데이터 내의 패턴을 스스로 인지하고 신뢰도 있는 예측 결과를 도출해 냅니다. 머신 러닝은 데이터를 관통하는 패턴을 학습하고, 이에 기반한 예측을 수행하면서 데이터 분석 영역에 새로운 혁신을 가져왔습니다. 데이터 분석 영역은 재빠르게 머신 러닝 기반의 예측 분석으로 재편되고 있습니다. 많은 데이터 분석가와 데이터 과학자가 머신 러닝 알고리즘 기반의 새로운 예측 모델을 이용해 더욱 정확한 예측 및 의사 결정을 도출하고 있으며, 데이터에 감춰진 새로운 의미와 인사이트를 발굴해 놀랄 만한 이익으로 연결시키고 있습니다.

일반적으로 머신 러닝은 지도학습, 비지도학습, 강화학습으로 나뉩니다. 지도학습의 대표적인 머신 러닝은 분류와 회귀로 나눌 수 있습니다.