

1. Python

Understanding of Python



1. 프로그래밍 개요

최희운 강사

프로그래밍이란?



✓ 프로그램 (Program)

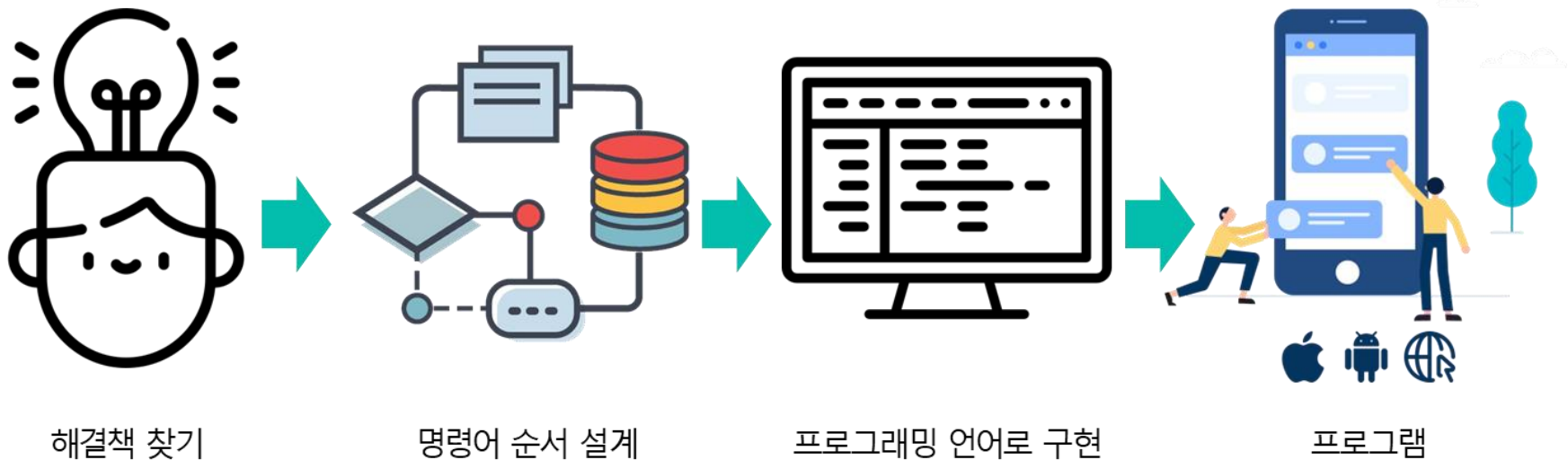
- 어떤 문제를 해결하기 위하여 그 처리 방법과 순서를 기술하여 컴퓨터에게 주어지는 일련의 명령문 집합체
- 컴퓨터가 이해하고 실행할 수 있는 명령 집합
- 컴퓨터에게 내리는 명령어의 모음

✓ 프로그래밍 (Programming)

- 컴퓨터가 작업을 수행하기 위해 따를 수 있는 프로그램이라는 명령어 순차 조합을 구성하는 일
- 프로그램 만드는 일
- 프로그램이 모여서 소프트웨어가 되고, 이러한 소프트웨어를 만드는 과정

프로그래밍이란?

- ✓ 프로그래밍 (Programming): 현실 문제를 해결하기 위해 프로그래밍 언어로 구현하는 모든 행위.



코딩과 소스코드

- ✓ 코딩(Coding)은 소스 코드를 작성하는 것을 의미
- ✓ 소스 코드 (Source Code)는 프로그래밍 언어로 작성한 텍스트 파일을 의미

메모장으로 프로그래밍
언어를 작성한 파일도
소스 코드인가요?



파이썬.py

네. 텍스트 편집기의
종류와는 상관 없이
프로그래밍 언어를 작성한
파일은 소스 코드입니다.

프로그래밍 언어란?

- ✓ 프로그래밍 언어 (Programming Language)
 - 기계에게 명령이나 연산을 시킬 목적으로 설계되어 기계와 의사소통을 할 수 있게 해주는 언어
 - 소프트웨어를 만들 때 사용하는 도구

프로그래밍 언어의 발전

```

MONITOR FOR 6802 1.4          9-14-80  TSC ASSEMBLER  PAGE    2

C000                          ORG      ROM+$0000 BEGIN MONITOR
C000 8E 00 70  START  LDS      #STACK

*****
* FUNCTION: INITA - Initialize ACIA
* INPUT: none
* OUTPUT: none
* CALLS: none
* DESTROYS: acc A

0013      RESETA EQU      %00010011
0011      CTLREG EQU      %00010001

C003 86 13      INITA  LDA A  #RESETA  RESET ACIA
C005 B7 80 04          STA A  ACIA
C008 86 11          LDA A  #CTLREG   SET 8 BITS AND 2 STOP
C00A B7 80 04          STA A  ACIA

C00D 7E C0 F1          JMP      SIGNON  GO TO START OF MONITOR

*****
* FUNCTION: INCH - Input character
* INPUT: none
* OUTPUT: char in acc A
* DESTROYS: acc A
* CALLS: none
* DESCRIPTION: Gets 1 character from terminal

```

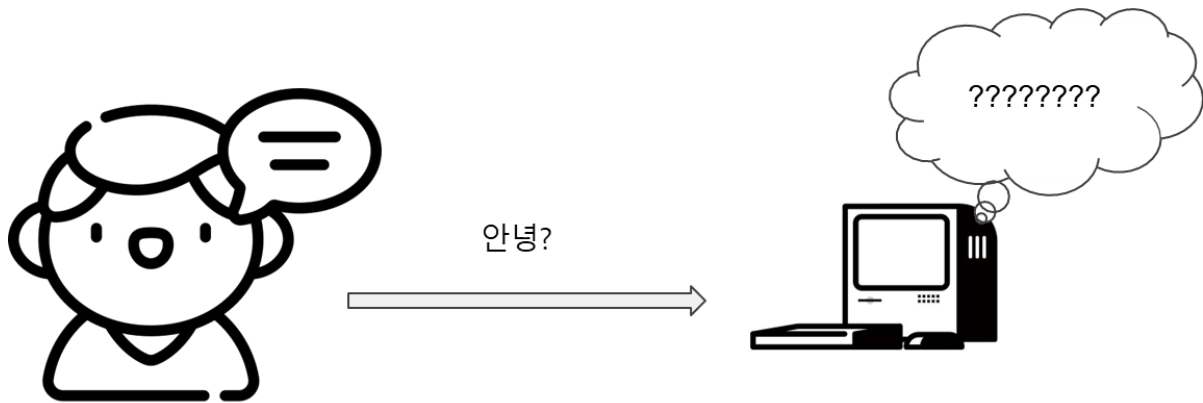
1. 초기에는 0과 1의 조합인 기계어로 개발
2. 어셈블리어 같은 저급언어 탄생
3. 파이썬과 같은 인간의 언어에 가까운 고급언어 탄생

최근 논리력을 기르기위한 학습용으로 프로그래밍 언어 사용

과거에 비해 쉬어져 다양한 전공에서 프로그래밍 언어 사용

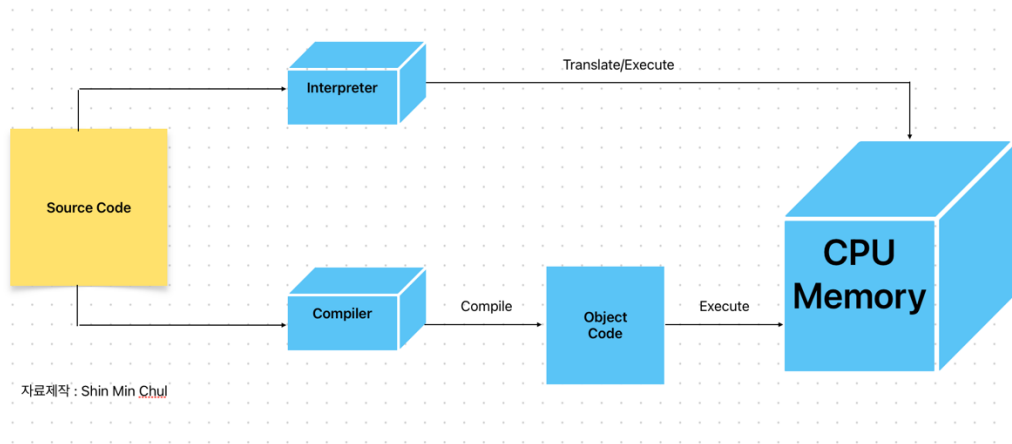
컴파일러 vs 인터프리터

- ✓ 컴파일러와 인터프리터는 기본적으로 프로그래밍 언어를 컴퓨터가 어떻게 이해할지에 대한 대답
 - 컴파일러: 미리 기계어로 변환해두었다가 사용하는 방식의 언어
 - 인터프리터: 실행 시점에서 기계어로 변환하는 방식의 언어이며 동적 컴파일러라고도 부름



컴파일러 vs 인터프리터

- ✓ 컴파일러: C, C++, JAVA
 - 전체 파일을 한 번에 스캔하여 변환, 초기 속도는 느리지만 실행 파일이 만들어지면 속도가 빠름
- ✓ 인터프리터: Python, JavaScript
 - 프로그램 실행 시 한 번에 한 문장씩 번역, 메모리 효율은 좋지만 속도가 느림



프로그래밍 언어 종류

✓ C

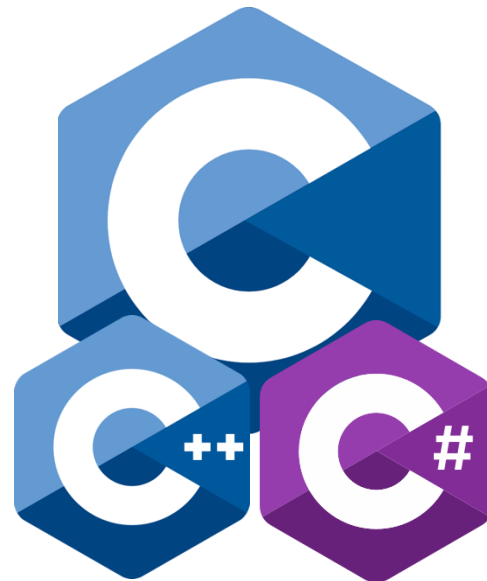
- 프로그램을 간단하게 설계할 수 있도록 하는 운영체제 공통 언어
 - Unix 운영체제를 작성하는 데 사용됨
- 메모리와 하드웨어에 대한 직접 접근이 가능해 매우 빠르고 효율적인 프로그램 작성 가능
- 절차적 프로그래밍 언어: 함수와 절차 중심으로 프로그래밍 하여 데이터와 함수를 별도로 관리
- 시스템 프로그래밍, 운영체제, 임베디드 시스템, 고성능 어플리케이션 등의 개발에 사용 됨

✓ C++

- C 언어에 객체지향을 추가해 기능 확장
 - 즉, 클래스를 사용하여 데이터와 함수를 캡슐화하여 재사용할 수 있음
- 절차적, 객체 지향적 프로그래밍을 모두 지원해 다양한 스타일로 코드 작성 가능
- C의 성능을 이어받아 그래픽 처리, 게임, 고성능 어플리케이션 개발에 많이 사용됨

✓ C#

- C, C++의 장점을 도입하면서도 자바의 영향을 받음
- 자동 메모리 관리가 지원되어 개발자가 직접 메모리 관리를 할 필요가 없음



프로그래밍 언어 종류

✓ 자바 (Java)

- 완전한 컴파일 언어와 인터프리터 언어의 중간 형태인 하이브리드 언어 (바이트코드로 컴파일된 후 JVM으로 이를 실행하는 방식)
 - JVM(Java Virtual Machine): java프로그램이 다양한 운영체제에서 동일하게 실행 될 수 있도록 하는 가상컴퓨터
- 객체 지향적 언어로, 기존의 코드를 재사용할 수 있고 모듈식 프로그램을 개발할 수 있음
- 독립적이며 고성능 애플리케이션을 개발할 수 있다는 점에서 개발자들로부터 높은 호응을 얻고 있음
- 강력한 디버깅 제공

✓ 자바스크립트 (JavaScript)

- 웹 개발에서 사용되는 프로그래밍 언어
- 웹 브라우저에서 한 줄씩 실행되는 인터프리터 언어
- 웹페이지에 동적 기능을 추가하여, 버튼 클릭, 입력 폼 처리, 애니메이션 등과 같은 상호작용 구현



프로그래밍 언어 종류

✓ 파이썬 (Python)

- 인터프리터 언어 (한 줄씩 실행가능한 언어)
- 웹사이트 및 서비스 개발 그리고 데이터 분석 등 다양한 분야에 활용 됨
- C 및 C++과 같은 프로그래밍 언어와의 상호운용성 덕분에 초보자 뿐만 아니라 숙련된 개발자 모두가 가장 쉽게 사용할 수 있는 언어
- 최종 실행 속도는 컴파일 언어보다 상대적으로 느린 편
 - 컴파일 언어: 전체 소스코드가 한 번에 컴파일 되어 실행 파일로 변환된 후 실행됨
 - 컴파일: 프로그래밍 언어로 작성된 소스코드를 컴퓨터가 이해할 수 있는 기계어로 변환하는 과정



프로그래밍 언어 사용 순위

# Ranking	Programming Language	Percentage (YoY Change)	YoY Trend
1	Python	16.389% (+0.347%)	
2	Java	11.164% (-0.986%)	^
3	JavaScript	10.605% (-4.400%)	v
4	C++	10.323% (+1.008%)	^
5	TypeScript	10.222% (+3.891%)	^

2022년 3분기 Github 프로그래밍 언어별 비율(출처: [Github 2.0](#))

프로그래밍 언어 사용 순위

- ✓ 목적에 따라 다양한 프로그래밍 언어가 존재하기 때문에 목적에 맞는 프로그래밍 언어를 선정해야 함
- ✓ 가장 많이 사용한다고 반드시 좋은 언어는 아니다.



Bigdata



iOS



Android



Frontend



Backend

2. Python 개요

최희운 강사

파이썬 개요

✓ 창시자: 귀도 반 로섬 (Guido Van Rossum)

✓ 파이썬 명칭의 유래

귀도 반 로섬이 즐겨보던 영국 BBC 코미디 프로그램의
6인조 코미디 그룹 몬티 파이썬에서 따옴

✓ 파이썬 개발 철학

- 아름다운 게 추한 거 보다 낫다
- 명시적인 것이 암시적인 것 보다 낫다
- 단순함이 복잡함 보다 낫다
- 복잡함이 난해한 것 보다 낫다
- 가독성은 중요하다



파이썬 개요

- ✓ 출시
 - 1989년에 개발 시작
 - 1990년 첫 버전 공개
- ✓ 사용 현황
 - 대형 글로벌 기업부터 스타트업까지 다양하게 활용
 - 구글, 유럽입자물리연구소, NASA, 핀인사이트, 등



파이썬이란?

- ✓ 사람이 이해하기에 문법이 쉽다 -> 사람의 언어와 흡사하다
- ✓ 데이터를 다루기 위한 NumPy, Pandas 등 제공되는 라이브러리가 많다.
- ✓ 손쉽게 데이터를 시각화할 수 있는 Matplotlib, Seaborn 라이브러리
- ✓ Django, Flask 프레임워크를 활용하여 웹 서비스를 만들 수 있다.
- ✓ SciPy를 활용하여 과학기술 계산 및 알고리즘을 활용할 수 있다.
- ✓ Tensorflow, Keras, PyTorch를 활용하여 Deep Learning 모델을 구현할 수 있다.

파이썬이란?

- ✓ 파이썬으로 개발하는 기업 [CWN 뉴스](#)

Instagram

Google

NETFLIX



Spotify®



Meta

파이썬 특징

✓ 단순, 간단

```
public class hello {  
    public static void main(string[] args) {  
        System.out.println("Hello World");  
    }  
}
```



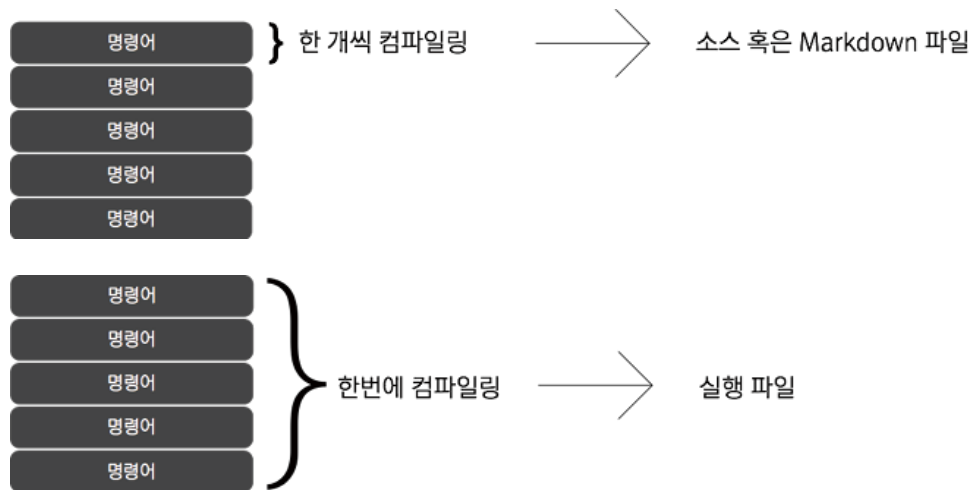
```
print("Hello World")
```

파이썬 특징

- ✓ 방대한 라이브러리
웹 개발 , 게임 개발, 데이터 과학 등 범용 언어로 쓰임
- ✓ 무료
- ✓ 어느 운영체제에도 사용 가능
윈도우, 리눅스, 맥 모두 사용 가능

파이썬 특징

- ✓ 인터프리터(Interpreter) 방식으로 동작하는 언어
- ✓ 프로그래밍 언어로 된 코드를 컴퓨터가 바로 처리할 수 있도록 즉시 변환
- ✓ 다른 방식으로 컴파일 방식이 존재



파이썬 특징

- ✓ 파이썬 소스 파일의 기본 인코딩은 UTF-8
- ✓ 인코딩이란 컴퓨터에서 문자를 처리하기 위한 방식으로 각 문자마다 0/1로 매핑이 되어있음
- ✓ 이러한 약속은 표준이 여럿일 수 있는데, 그 중 하나가 ASCII(아스키)이고 다른 하나는 UTF-8
- ✓ ASCII는 영어권에서 컴퓨터가 만들어져서 그 때 사용되던 표준으로 $2^7=128$ 개를 가지고 있음
- ✓ UTF-8은 다양한 언어의 표현을 위해 만든 표준

파이썬을 들어가기 전 지켜야 할 사항

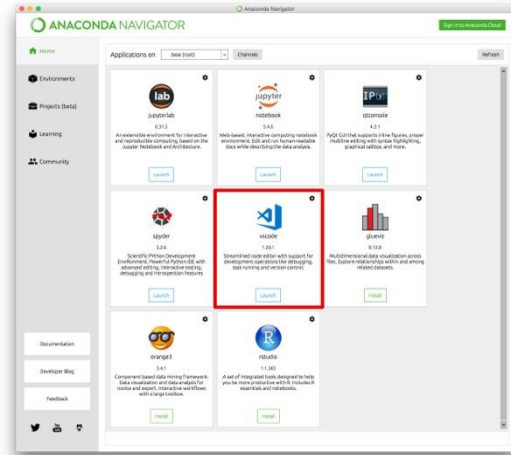
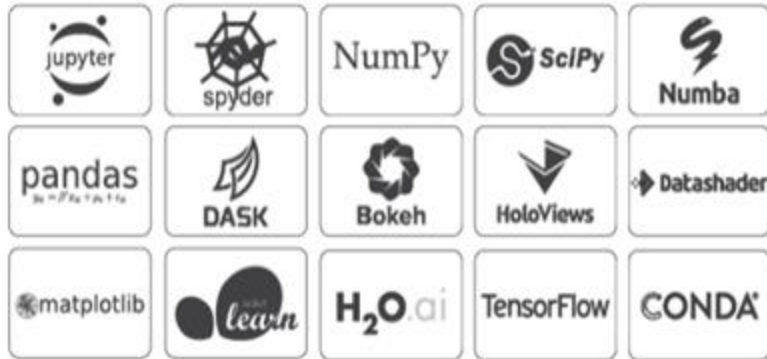
- ✓ 세미콜론은 옵션입니다.
- ✓ 파이썬의 경우는 생략이 가능. 한 줄에 여러 명령어를 작성할 경우에만 구분자로 사용
 - java: `System.out.println('hello world');`
 - python: `print('hello')`
- ✓ 파이썬은 띄어쓰기(공백)를 통해 코드의 범위를 구분
 - 들여쓰기를 통해 구분한다는 의미
- ✓ 즉, 띄어쓰기를 할 땐 늘 주의해서 구획(코드 블록)이 섞이지 않도록 해야 함
- ✓ 범위를 나눌 때 띄어쓰기 4회를 권장하고 있음

3-1. 환경세팅 - 윈도우

최희윤 강사

아나콘다란?

- ✓ 파이썬으로 데이터 과학, 기계학습을 수행하기 위한 여러 도구와 라이브러리 제공



아나콘다란?

- ✓ 패키지 관리
 - 파이썬과 이와 관련된 다양한 패키지 설치, 업데이트, 관리하기 용이
 - 프로젝트 별로 파이썬 및 라이브러리 버전 충돌 최소화 가능
- ✓ 라이브러리 및 도구
 - 데이터 과학 및 머신러닝에 필요한 라이브러리들이 미리 설치되어 제공됨
(ex. Numpy, pandas, Matplotlib, Scikit-learn, TensorFlow, Pytorch, 등)
- ✓ 다양한 플랫폼 지원
 - 윈도우, 맥, 리눅스 등 모든 운영체제 지원
- ✓ 설치 용이

아나콘다 설치

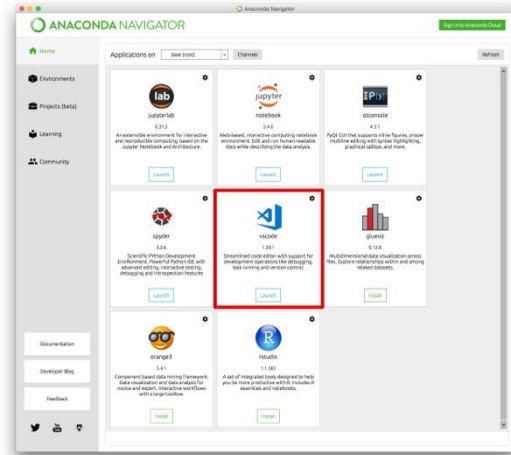
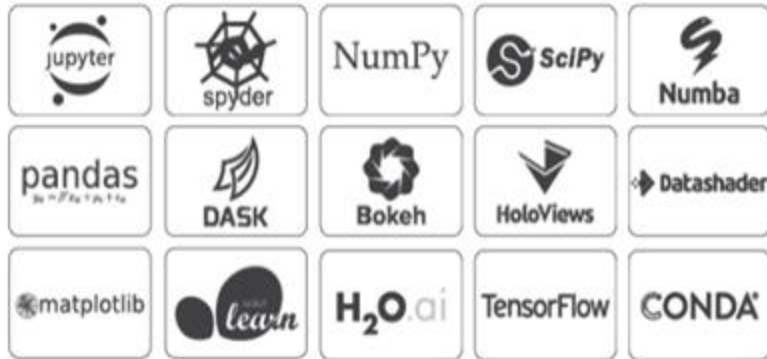
<https://alluring-parent-4dd.notion.site/14ed791a37c6807080dffc3aa61a4a4?pvs=4>

3-1. 환경세팅 - ios

최희운 강사

아나콘다란?

- ✓ 파이썬으로 데이터 과학, 기계학습을 수행하기 위한 여러 도구와 라이브러리 제공

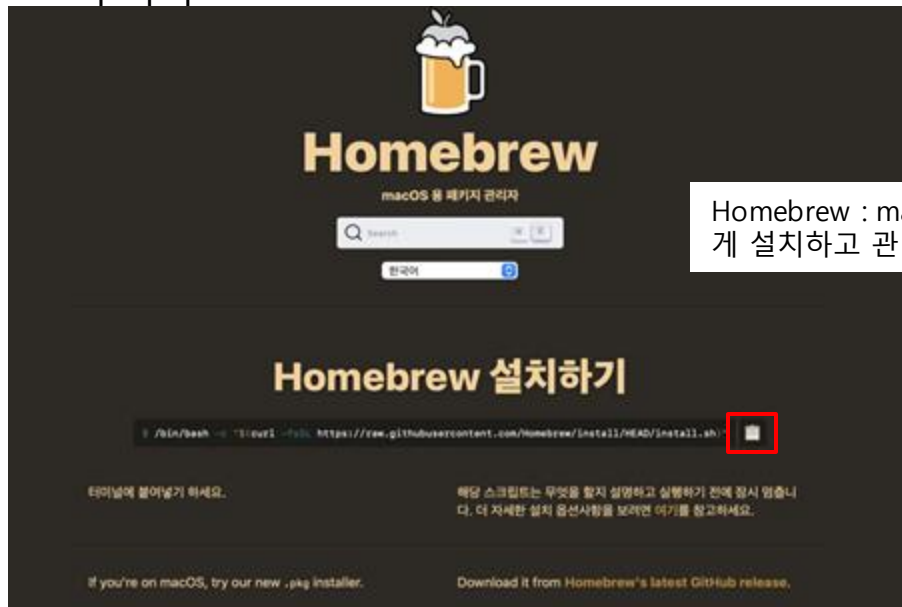


아나콘다란?

- ✓ 패키지 관리
 - 파이썬과 이와 관련된 다양한 패키지 설치, 업데이트, 관리하기 용이
 - 프로젝트 별로 파이썬 및 라이브러리 버전 충돌 최소화 가능
- ✓ 라이브러리 및 도구
 - 데이터 과학 및 머신러닝에 필요한 라이브러리들이 미리 설치되어 제공됨
(ex. Numpy, pandas, Matplotlib, Scikit-learn, TensorFlow, Pytorch, 등)
- ✓ 다양한 플랫폼 지원
 - 윈도우, 맥, 리눅스 등 모든 운영체제 지원
- ✓ 설치 용이

Homebrew 설치

1. [Homebrew](#) 홈페이지에 들어가서 터미널 명령어 복사



Homebrew : macOS에서 소프트웨어 패키지를 쉽게 설치하고 관리할 수 있는 패키지 관리자

Homebrew 설치

2. **brew --version** 으로 homebrew 설치 확인

```
[(base) fininsight@fininsightui-MacBookPro ~ % brew --version  
Homebrew 4.2.15
```

✓ 만약 "zsh: command not found" 가 뜬다면 zsh셸에 path를 추가해줘야 한다.

3. 다음의 명령어를 순차적으로 입력

- 1) Homebrew PATH 추가 (Intel Mac 경우)

```
echo 'export PATH="/usr/local/homebrew/bin:$PATH"' >> ~/.zshrc
```

- 2) Homebrew PATH 추가 (Apple Silicon Mac 경우)

```
echo 'export PATH="/opt/homebrew/bin:$PATH"' >> ~/.zshrc
```

```
source ~/.zshrc
```

Anaconda 설치

4. 아나콘다 설치 및 경로 확인

1) brew install --cask anaconda 설치

```
(base) fininsight@fininsightui-MacBookPro ~ % brew install --cask anaconda
```

2) 패스워드 입력 후 PREFIX(아나콘다 설치경로) 확인

[Password:

PREFIX=/usr/local/anaconda3

✓ 패스워드를 입력하면 PREFIX가 뜨는데 이 경로를 잘 기억 해야 한다.

Anaconda 설치

5. PATH 설정 및 conda 명령어 확인

1) 아나콘다를 설치하고 conda —version 명령어를 치면 오류가 발생

✓ 환경변수를 설정해줘야 정상적으로 아나콘다를 사용할 수 있다.

2) 터미널을 종료하고 다시 열어서 conda —version로 확인

```
zsh: command not found: conda
```

3) 환경변수 설정을 위해 다음의 명령어를 순차대로 입력 (Prefix 경로 확인!)

```
echo 'export PATH="/usr/local/anaconda3/bin:$PATH"' >> ~/.zshrc
```

```
source ~/.zshrc
```

4) conda —version으로 확인하면 정상적으로 설치된 것 확인 가능

```
[fininsight@fininsightui-MacBookPro ~ % conda --version  
conda 24.5.0
```

vscode 설치

<https://www.codeit.kr/tutorials/67/vscode-tutorial-macos>

3-2.환경 세팅

최희운 강사

실습 파일 저장 경로 세팅

✓ 윈도우

- c:/sesac/01_python
- C 드라이브 하위에 sesac 폴더 생성 후 01_python 폴더 생성

✓ 맥

- /Users/user/Documents/sesac/01_python
- Home 디렉토리 (/Users/user/) 하위의 Documents 폴더에 sesac 폴더 생성 후 01_python 폴더 생성
- (Documents 하위 외에 원하는 위치에 sesac폴더를 생성해도 좋습니다.)

터미널 vs 셸 vs 커널

✓ 터미널 (Terminal)

- 사용자가 명령어를 입력하여 컴퓨터와 상호작용할 수 있는 인터페이스
- 입출력이 가능한 하드웨어 (console이라고도 부름)
- 터미널에 실행되는 '셸(Shell)'을 통해 명령어를 입력하고 전달할 수 있음

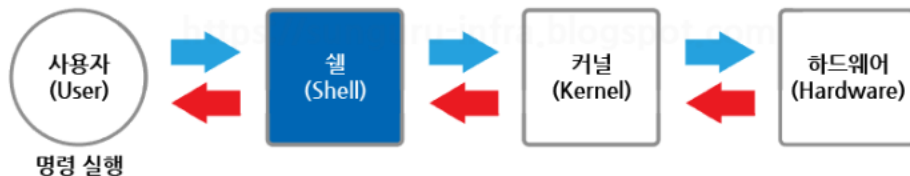
✓ 셸 (Shell)

- Command Line Interface (CLI)로 구현되는 프로그램
- 주로 사용자가 입력하는 명령어들을 해석하는 해석기의 역할 수행
- 즉, 사용자가 셸을 통해 명령어를 입력하고 해당 명령어가 셸에 의해 해석되어 커널(Kernel)에 전달 됨
- 커널에서는 각 명령을 실행하게 되고, 우리는 출력되는 결과를 하드웨어를 통해 확인하게 됨

터미널 vs 셸 vs 커널

✓ 커널 (Kernel)

- 운영체제의 핵심 구성요소로, 하드웨어와 소프트웨어의 중간다리 역할 (Linux Kernel, Windows NT Kernel, 등)
- 하드웨어 관리: CPU, 메모리, 디스크 드라이브 등의 자원을 관리
- 프로세스 관리: 여러 프로세스의 실행을 조율하고, 프로세스 간의 충돌 방지
- 보안: 사용자와 프로그램이 하드웨어를 직접 조작하지 못하게 보호
- 시스템 호출 처리: 사용자 프로그램이 운영 체제 기능 (예: 파일 읽기, 쓰기)을 요청할 때 이를 처리
- 항상 메모리에 상주하며, 컴퓨터가 부팅되면 가장 먼저 로드됨



<http://sunguru-infra.blogspot.com/2019/03/bash-1-shell-programming.html>

대화형 vs 스크립트

✓ 대화형

- 사용자가 명령어를 직접 입력하고, 입력한 즉시 결과를 확인하며 작업을 수행하는 방식
- 작업을 즉시 실행하고 확인할 수 있어서 테스트나 디버깅에 적합
- 반복적인 작업을 자동화하기 어렵고, 사용자 입력에 의존적

```
python 📄 코드 복사  
  
>>> x = 5  
>>> x + 3  
8
```

대화형 vs 스크립트

✓ 노트북(Notebook): 대화형 개발환경

- 코드, 설명(문서화), 시각화, 및 실행 결과를 하나의 문서에서 관리할 수 있는 대화형 개발 환경
- 코드와 결과를 문서화 하여 공유하기 쉬움
- 코드 셀(cell)과 마크다운 셀을 함께 사용하여 설명과 코드가 동시에 존재 가능
- 데이터 분석, 머신러닝 모델 실험 등에서 작업 과정을 기록하고 재현 가능
- Jupyter Notebook, Google Colab, VS Code의 노트북 인터페이스 등이 대표적

VS Code의 대화형 개발환경

```
print('Hello, World!')
```

[30] ✓ 0.1s

Python

... Hello, World!

대화형 vs 스크립트

✓ 스크립트

- 명령어들을 파일에 작성하여, 파일을 실행하면 여러 명령이 순차적으로 자동으로 처리되는 방식
- 동일한 작업을 반복적으로 수행할 때 적합 (유지보수 쉬움)
- 사용자 입력 없이 한 번에 실행 됨
- 작성과 디버깅에 시간이 걸릴 수 있으며, 즉각적인 피드백이 부족할 수 있음
- 따라서, 명령어를 미리 작성하고 자동으로 실행하는 방식으로 반복적이고 구조화된 작업에 적합

```
test.py > ...  
1  import numpy as np  
2  
3  my_array = np.array([[0, 1, 2], [3, 4, 5]])  
4  print(my_array)
```

```
● ualization/실습 파일 /test.py  
[[0 1 2]  
 [3 4 5]]
```

가상환경 생성

✓ 가상환경 생성

- 파이썬이나 다른 프로그래밍 환경 언어에서 특정 프로젝트에 필요한 의존성 (패키지와 라이브러리)을 다른 프로젝트와 분리하여 관리할 수 있도록 도와주는 독립적인 작업환경
- 가상환경을 커널로 등록하여 주피터 노트북이 해당 가상환경을 사용할 수 있도록 설정할 수 있음
 - 주피터 노트북에서 커널은 파이썬 인터프리터를 실행하는 환경
 - 즉, 주피터 노트북에서 커널은 프로그래밍 언어의 실행 환경으로, 코드 실행을 담당
 - 주피터 노트북에서는 코드 셀을 실행할 때 사용하는 파이썬 인터프리터를 커널이라고 함

가상환경 생성

✓ 가상환경 생성

- conda create -n sesac python=3.10

가상환경이름 파이썬 버전

```
(base) C:\Users\>conda create -n test python=3.10
Channels:
 - defaults
Platform: win-64
Collecting package metadata (repodata.json): done
Solving environment: done

## Package Plan ##

environment location: C:\Users\>.conda\envs\test

added / updated specs:
 - python=3.10

The following packages will be downloaded:
```

package	build	
bzip2-1.0.8	h2bbff1b_6	90 KB
ca-certificates-2024.3.11	haa95532_0	128 KB
libffi-3.4.4	hd77b12b_1	122 KB
openssl-3.0.13	h2bbff1b_1	7.5 MB
pip-24.0	py310haa95532_0	2.9 MB
python-3.10.14	he1021f5_1	15.9 MB
setuptools-69.5.1	py310haa95532_0	1013 KB
sqlite-3.45.3	h2bbff1b_0	973 KB
tk-8.6.14	h0416ee5_0	3.5 MB
tzdata-2024a	h04d1e81_0	116 KB

✓ 생성한 가상환경 목록 보기

- conda env list

```
(base) C:\Users\>conda info --envs
# conda environments:
#
base                * C:\ProgramData\anaconda3
test                C:\Users\>.conda\envs\test
```

가상환경 관련 명령어

✓ 가상환경 활성화

- `conda activate sesac`

가상환경이름

✓ 가상환경 비활성화

- `conda deactivate`

✓ 가상환경 삭제

- `conda remove --name [가상환경 이름] --all`

4. 파이썬 자료형

최희윤 강사

자료형이란?

✓ 자료형: 데이터의 종류 Data Type

자료형의 종류

- ✓ 숫자 (number)
- ✓ 시퀀스 (sequence)
- ✓ 매핑 (mapping)
- ✓ 불린 (boolean)
- ✓ 세트 (set)

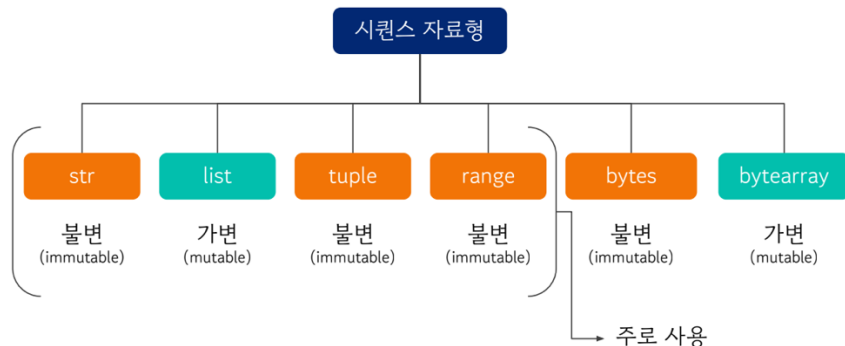
자료형의 종류

✓ 숫자 (number)

- int (integer): 정수
- float: 실수

✓ 시퀀스 (sequence)

- str (string): 문자열
- list: 리스트 (대괄호 []에 한 개 이상의 데이터가 묶여있는 형태)
- tuple: 튜플 (소괄호 ()에 한 개 이상의 데이터가 묶여있는 형태)



자료형의 종류

- ✓ 매핑 (mapping): 종괄호에 key와 value 쌍으로 구성 돼있는 데이터
 - dict (dictionary): 딕셔너리
 - key
 - value
- ✓ bool (boolean): 참과 거짓 두 가지의 값만 가질 수 있는 데이터 타입
 - True: 참
 - False: 거짓
- ✓ 세트 (set)
 - 순서가 유지되지 않으며 중복 원소가 없는 데이터의 집합 (종괄호로 요소를 묶음)
 - 집합 연산 사용 가능

Print() 함수

✓ print() 함수란?

- print()에서 괄호 안에 들어가는 값을 화면에 출력 하는 함수
- print는 함수 이름으로, '함수'는 소괄호 안에 인자(arguments)를 넣어 결과를 도출할 수 있음
- print 함수의 인자값: 화면에 출력하고자 하는 값
- 함수는 input으로 인자(arguments)를 넘겨주고, 내부 로직을 처리한 뒤 output을 반환
- 함수 호출에는 함수명과 소괄호가 필수이며 이 둘은 반드시 붙여 씀 (ex. print())

print('Hello, World')

함수명

인자(arguments)

자료형 확인 방법

- ✓ 자료형(데이터 타입) 확인하는 방법: type(값)

```
print(type( 1 ))
```

```
>> <class 'int'>
```

```
print(type('Hello, World'))
```

```
>> <class 'str'>
```

```
print(type([1, 2, 3]))
```

```
>> <class 'list'>
```

```
print(type(0.1))
```

```
>> <class 'float'>
```

```
print(type((1, 2, 3)))
```

```
>> <class 'tuple'>
```

```
print(type({'key': 'value'}))
```

```
>> <class 'dict'>
```

자료형 확인 실습 01

✓ 각 데이터의 데이터타입(자료형)을 확인해주세요.

1. 15827

2. 7.0

3. {1: 'a'}

4. "Hi, My name is 'Daisy'"

5. (812.324992, 'orange', ('test'))

6. ['apple', 3.513, {'test': 1}]

5. 산술 연산

최희윤 강사

산술 연산

구분	예제	결과
덧셈	$1 + 2$	3
뺄셈	$1 - 2$	-1
곱셈	$5 * 2$	10
나눗셈	$5 / 2$	2.5
나눗셈 (몫)	$5 // 2$	2
나눗셈 (나머지)	$9 \% 2$	1
제곱	$2 ** 3$	8
괄호 (우선순위)	$(2 + 3) * 2$	10

정수와 실수의 표현 범위

- ✓ 실수가 정수보다 표현 범위가 넓음
- ✓ 즉, 실수를 정수로 변환하게 되면 소수점 아래 값의 손실이 발생
- ✓ 소수점 아래 값의 손실 없이 정수와 실수를 계산할 수 있는 방법
정수를 실수로 만들고 실수와 계산하는 것
- ✓ 따라서, 실수와 정수의 계산은 결과값이 실수가 됨

실수

-1.1, -1.0, 0.9999, 0, 1.111111,
99.2192, 128.28182, ...

정수

..., -3, -2, -1, 0, 1, 2, 3, ...

정수와 실수의 표현 범위

$$2 + 2.0 = 4.0 \quad \blacktriangleright \text{실수}$$

$$3 - 2.0 = 1.0 \quad \blacktriangleright \text{실수}$$

$$4 * 5.0 = 20.0 \quad \blacktriangleright \text{실수}$$

$$4 / 2.0 = 2.0 \quad \blacktriangleright \text{실수}$$

실습 01

✓ 산술 연산 실습

- [문제1] $13 + (22 - 3) \times 4$ 의 결과를 구해보세요.
- [문제2] $13 + ((22 - 3) \times 4) \div 5$ 의 결과를 구해보세요.

숫자형 변환

✓ 자료형 변환

- `int()`: 정수로 변환
- `float()`: 실수로 변환

```
print(int( 9.2 ))  
>> 9
```

```
print(int( 0.6 ))  
>> 0
```

```
print(int( 2.0 ))  
>> 2
```

```
print(float( 9 ))  
>> 9.0
```

6. 변수

최희운 강사

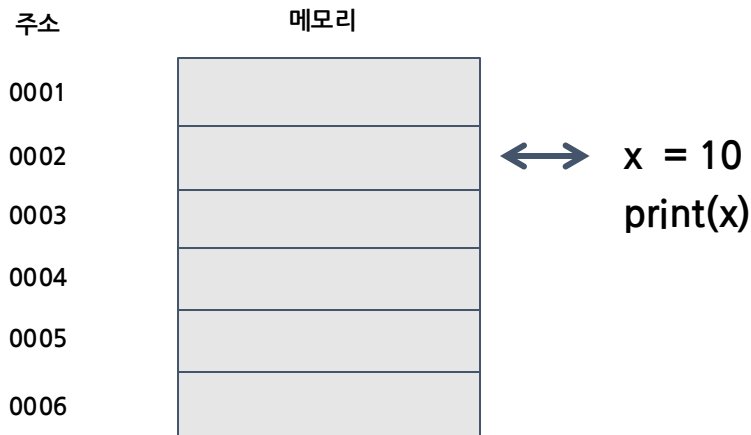
변수란?

하나의 데이터를 저장할 수 있는 메모리 공간, 변할 수 있는 수
변할 수 있는 값을 메모리상에 저장하고 활용할 수 있도록 하는
공간

$$\begin{array}{ccc} \underline{X} & = & \underline{10} \\ \text{변수} & & \text{값} \end{array}$$

변수란?

메모리상에 저장한다 = 프로그램이 실행되어 있는 동안에만
저장된다



리터럴(literal)이란?

값 자체

숫자, 문자열, True, False와 같은 값을 의미
(변수와는 달리 고정된 값을 가진 데이터)

$X = 10$

숫자형 리터럴

변수 사용 방법

✓ 대입 연산자

수학에서 '같다' 라는 의미로 쓰이는 등호 기호는

프로그래밍에서 변수 생성 후 리터럴을 메모리에 대입(할당, assignment)한다는 의미

<u>X</u>	=	<u>10</u>
변수		숫자형 리터럴

동시 할당(unpacking)

한 줄에 여러 변수를 초기화 하는 방식
좌변에 있는 변수들과 우변에 있는 리터럴 요소 개수가 일치하면,
각각의 변수에 값을 순서대로 할당

$$x, y, z = 10, 20, 30$$

변수

소괄호가 생략된
튜플

동시 할당(unpacking)

튜플 외에 다른 시퀀스 자료형도 unpacking 가능

```
x, y, z = [10, 20, 30]
```

```
print(x, y, z)
```

```
>> 10 20 30
```

동시 할당(unpacking)

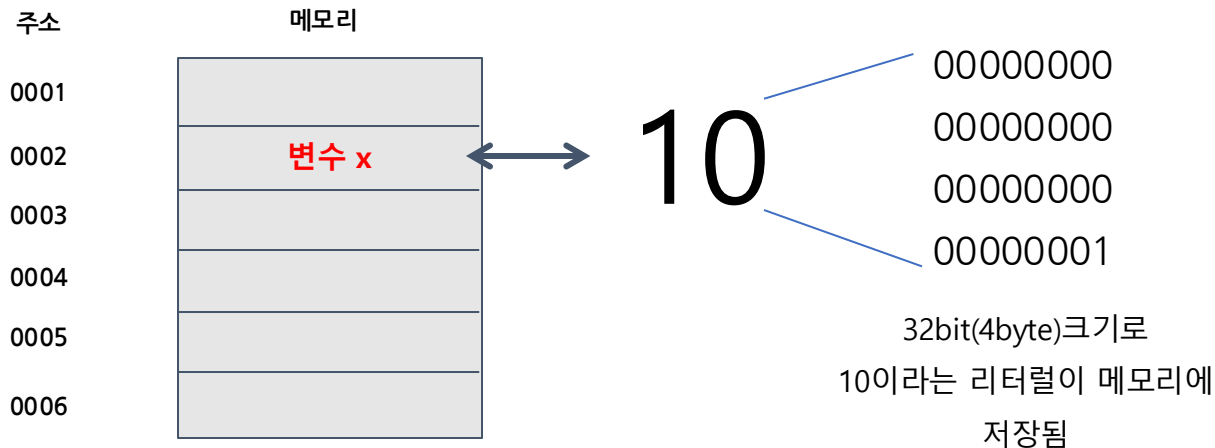
튜플 외에 다른 시퀀스 자료형도 unpacking 가능

`x, y, z = '문자열'`

```
print(x, y, z)
```

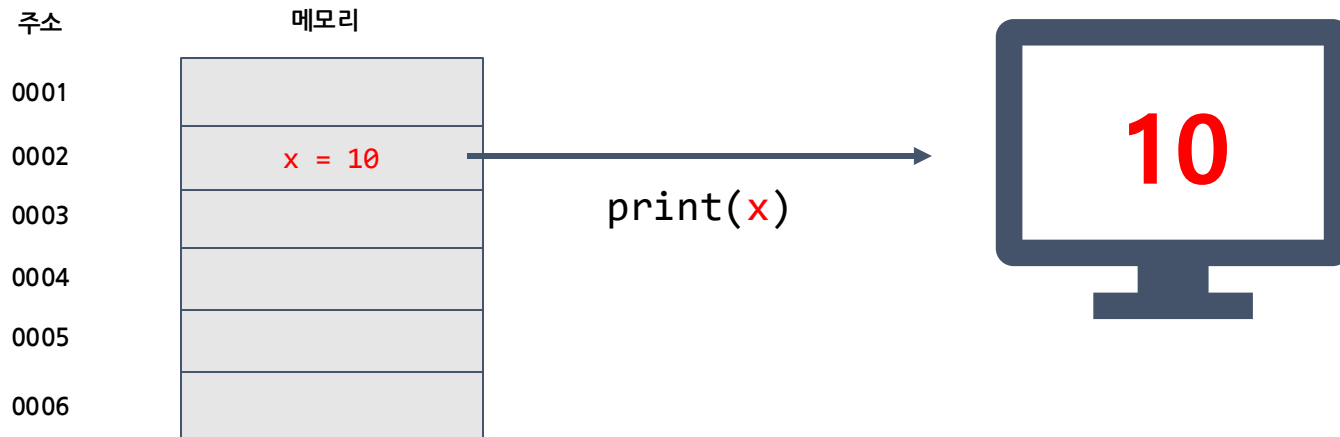
```
>> 문자열
```

변수에 리터럴을 저장하는 것에 대한 도식화



- x라는 변수는 메모리의 특정 위치를 참조
- 리터럴 값(10)이 변수 x에 할당(assign)되면, 메모리의 해당 위치에 값이 저장됨
- 변수는 메모리 내 주소 역할을 함
- 리터럴(10)은 32비트의 이진수로 변환되어 저장 됨
- 메모리는 저장되는 데이터의 크기, 종류(타입), 값의 정보를 함께 관리

변수에 리터럴을 저장하는 것에 대한 도식화



변수명 생성 조건

- 문자, 숫자, 언더바(_)를 사용하여 생성
- 대소문자 구분 가능
- 숫자부터 시작할 수 없음
- 공백 포함 불가
- 특수 문자 사용 불가능
- 파이썬의 키워드(예약어) 사용 불가
- snake_case를 활용하여 명명하도록 권장됨

None, True, False, and, as, assert, break, class, continue, def, del, elif, else, except, finally, for, from, global, if, import, in, is, lambda, nonlocal, not, or, pass, raise, return, try, while, with, yield

변수명 생성 조건 - 식별자 (Identifier)

- ✓ 식별자(Identifier)는 변수, 상수, 함수, 사용자 정의 타입 등에서 다른 것들과 구분하기 위해서 사용되는 변수의 이름, 상수의 이름 등을 일반화 해서 지칭하는 용어
- ✓ 사용자가 임의로 지정하지만 직관적이고 가독성있게 정의하는 것이 좋음

변수명 생성 조건 - 식별자 (Identifier)

- ✓ **snake_case**

변수명/함수명/파일명을 작성할 때, 단어를 소문자로만 작성하고 단어와 단어 사이를 언더바(_)로 연결하는 표기법

- ✓ **PascalCase**

모든 단어의 첫 글자를 대문자로 작성하며, 주로 클래스 이름에 사용

- ✓ **kebab-case**

모든 단어를 소문자로 작성하고, 단어 사이를 하이픈(-)으로 연결 하며, 주로 URL 경로, HTML/CSS 파일 이름에 사용

- ✓ **camelCase**

단어 첫 글자를 대문자로 작성하되, 첫 단어는 소문자로 유지하며, 주로 JavaScript, Java, C#에서 사용

변수명 생성 조건 - 식별자 (Identifier)

- ✓ 대부분 카멜 표기법과 파스칼 표기법을 적절하게 조합하여 사용.
 - ✓ Python에서는 "[Style Guid for Python Code](#)"를 기술하고 있음
 - 코드 배치, 표현식과 구문의 공백등 다양한 코드 스타일을 기술하고 있음.
 - 이름 지정 규칙을 아래와 같이 정의하고 있음.
1. 변수명에서 _(밑줄)은 위치에 따라 다음과 같은 의미가 있습니다.
 - 1개의 밑줄로 시작 : 내부적으로 사용되는 변수
 - 1개의 밑줄로 종료 : 파이썬 기본 키워드와 충돌을 피하려고 사용
 - 2개의 밑줄로 시작 : 클래스 속성으로 사용
 - 2개의 밑줄로 종료 : 사용자가 조정할 수 있는 네임스페이스 안의 속성
 2. 소문자 l, 대문자 O, 대문자 I 는 변수명으로 사용하지 마세요.
 3. 모듈(Module) 명은 짧은 소문자로 구성되며 필요하다면 밑줄로 나눕니다.
 - 모듈은 파이썬 파일(.py)에 대응하기 때문에 파일 시스템의 영향을 받으니 주의하세요.
 - C/C++ 확장 모듈은 밑줄로 시작합니다.
 4. 클래스명은 카멜케이스(CamelCase)로 작성합니다.
 - 내부적으로 쓰이면 밑줄을 앞에 붙입니다.
 - 예외(Exception)는 실제로 에러인 경우엔 "Error"를 뒤에 붙입니다.
 5. 함수명은 소문자로 구성하되 필요하다면 밑줄로 나눕니다.

실습 01

✓ 변수 통한 리터럴 출력 실습

➤ [문제1] 실행 결과가 아래와 같이 출력되도록 하나의 셀에 코드를 작성하세요.

```
>> Hello, World!
```

```
>> Hello, World!
```

```
>> Python Programming
```

변수로 계산하기

```
x = 20
y = 30
print( x + y )
>> 50
```

```
x = 20
y = 30
c = x + y
print( c )
>> 50
```

```
x = 10
print( x + 10 )
>> 20
```

```
x = 10
x = x + 10
print( x )
>> 20
```

```
x = 10
x += 10
print( x )
>> 20
```



x와의 계산을 다시 x에 할당하려고 할 때

복합 할당 연산자

✓ 변수를 업데이트할 때 자주 사용되는 연산과 할당을 하나의 연산자로 결합

```
x = 10
```

```
x -= 10
```

```
print( x )
```

```
>> 10
```

```
x = 10
```

```
x *= 10
```

```
print( x )
```

```
>> 100
```

```
x = 10
```

```
x /= 10
```

```
print( x )
```

```
>> 1.0
```

```
x = 2
```

```
x **= 3
```

```
print( x )
```

```
>> 8
```

비어 있는 변수 (None)

- None: '아무 값도 없는'
- 값이 없음을 나타내는 특별한 상수
- None Type이라는 고유한 타입을 가짐
- 논리연산(Bool)에서는 False로 간주 됨
(ex. if None: 에서 None은 False로 간주되어 조건문을 실행하지 않음)
- 변수나 함수에서 반환할 값이 없거나 값이 존재하지 않을 때 None을 사용
- 다른 프로그래밍 언어에서는 None대신 null이 쓰임
(Java Script, SQL, JSON, C#, 등)

비어 있는 변수 (None)

```
x = None
```

```
print(x)
```

```
>> None
```

```
print(type(x))
```

```
>> <class 'None Type'>
```

변수 삭제 (del)

- del: 파이썬에서 변수를 삭제하거나, 리스트, 딕셔너리 등의 특정 요소를 제거할 때 사용
- 메모리에서 변수를 제거하여 더이상 접근할 수 없도록 만들

➤ 기본 사용법:

```
del 변수명
```

➤ 사용 예시:

```
x = 10
```

```
del x
```

```
print(x)
```

```
>> NameError: name 'x' is not defined
```


실습02

- ✓ [문제] Input()함수를 사용하여 사용자로 부터 변수 x, 변수 y에 값을 입력 받아 더하고 곱하는 프로그램을 작성해주세요.

➤ 실행:

x 값을 입력해주세요: 20

y값을 입력해주세요: 30

➤ 결과:

50

600

키워드(Keyword)

- ✓ 키워드(Keyword)는 파이썬에서 이미 예약되어 있는 문자열
- ✓ 따라서, 다른 용도로 사용 불가능
- ✓ 아래의 코드를 실행해보면, False, True, ... 등과 같은 문자열은 파이썬 고유의 키워드들임

```
import keyword  
  
print(keyword.kwlist)
```

상수 (Constant)

- ✓ 상수(Constant)는 항상 똑같은 값을 저장
- ✓ 즉, 사전에 미리 정의 되어있음
- ✓ constant1.py -> 파일 이름

```
PI = 3.14
```

```
GRAVITY = 9.8
```

- ✓ python_basic1.ipynb -> 파일 이름

```
import constant1
```

```
constant.PI = 3.141592 # 값을 변경하므로 상수가 역할이 사라짐
```

7. Bool과 비교, 논리 연산자

최희윤 강사

Boolean

True

참

False

거짓

참,거짓 두 가지의 값만 가질 수 있는 데이터 타입
반드시 첫 글자를 대문자로 작성해야 함

Boolean

- true, false처럼 소문자로 작성 시에는 변수명으로 사용 가능
- 단, 굳이 헛갈리게 변수명으로 쓰이는 경우는 잘 없음

```
true = '참입니다.'
```

```
print(true)
```

```
>> 참입니다.
```

```
false = 0
```

```
print(false)
```

```
>> 0
```

비교 연산자

>

크다

>

=

크거나 같다

<

작다

<

=

작거나 같다

= =

같다

!=

같지 않다

비교 연산자

✓ 연산 결과는 Boolean

```
print(3 > 1)
```

```
>> True
```

```
print(3 < 1)
```

```
>> False
```

```
print(type(10 != 10))
```

```
>> <class 'bool'>
```

```
print(10 == 10)
```

```
>> True
```

```
print(10 != 10)
```

```
>> False
```


논리 연산자

and

모두가 True 면 True
하나라도 False 면 False

or

하나만 True 여도 True
모두가 False 면 False

not

반대

논리 연산자

✓ 연산 결과는 Boolean

```
print( True and True )  
>> True
```

```
print( True or True )  
>> True
```

```
print( Not True )  
>> False
```

```
print( True and False )  
>> False
```

```
print( True or False )  
>> True
```

```
print( Not True )  
>> True
```

```
print( False and False )  
>> False
```

```
print( False or False )  
>> False
```

비교 + 논리 연산자

5 > 10 **and** 20 ==

20

False

True

False

5 == 5 **or** 10 !=

10

True

False

True

실습01

- ✓ [문제] 사용자로 점수를 3개 입력 받아
모든 점수가 65점 보다 클 경우 True, 점수가 하나라도 65점이 안 넘을 경우 False를 출력하세요.

➤ 실행:

첫 번째 점수: 20

두 번째 점수: 70

세 번째 점수: 80

➤ 결과:

False

➤ 실행:

첫 번째 점수: 70

두 번째 점수: 80

세 번째 점수: 90

➤ 결과:

True

8. 문자형

최희윤 강사

문자열

✓ 문자열 자료형(str)

문자나 단어, 문장 등의 텍스트 데이터 표현하는 자료형

✓ 생성 방식

- 작은 따옴표(' ')나 큰 따옴표(" ")로 생성 가능
- 작은 따옴표와 큰 따옴표 혼용 가능
- 따옴표 세 개를 연달아 사용하여(삼중 따옴표) 줄바꿈 가능

변수 = "Hello, World"

변수 = 'Hello, World'

문자열

✓ 문자열 생성 방식

작은 따옴표와 큰 따옴표를 통한 문자열 생성
방식

```
a = 'Hello, World!'
```

```
b = "Hello, World!"
```

작은 따옴표와 큰 따옴표 혼용

```
c = 'Hello, "This is my world!"'
```

```
d = "Hello, 'This is my world!'"
```

이스케이프문자 역슬래시(\) 활용

```
e = 'Hello, \\'This is my world!\\'
```

문자열

✓ 문자열 생성 방식

```
# \n 통한 줄바꿈
c = 'This is my example:\nHello, World!'
d = "This is my example:\nHello, World!"

print(c) # c와 d의 결과는 동일

>> This is my example:
    Hello, World!
```


문자열

✓ 문자열 생성 방식

```
# raw string 통해 \n를 문자열로 그대로  
출력하기
```

```
e = r'c:\name'  
print(e)  
>> c:\name
```

```
# 이스케이프문자 역슬래시(\) 활용
```

```
d = 'c:\\name'  
print(d)  
>> c:\name
```

문자열

✓ 문자열 생성 방식

```
# 삼중 따옴표를 통한 줄바꿈
a = '''This is my example:
Hello, World!'''
b = """ This is my example:
Hello, World!"""

print(a) # a와 b의 결과는 동일

>> This is my example:
Hello, World!
```

문자열

✓ 문자열 생성 방식

```
# 삼중 따옴표를 통한 줄바꿈
a = '''
This is my example:
Hello, World!'''

print(a)

>>
This is my example:
Hello, World!
```

<- 줄바꿈이 적용 됨

<- 줄바꿈이 적용 됨

문자열

✓ 문자열 생성 방식

```
# 삼중 따옴표를 통한 줄바꿈
a = ''' \                                # 역슬래시로 줄바꿈 방지
가능

This is my example:
Hello, World!'''

print(a)

>> This is my example:
Hello, World!
```

실습01

✓ [문제1] 아래 문장을 리터럴로 만든 후 출력(print)하세요.

➤ 주어진 문장:

“안녕?”이라는 그 인사를 듣고 나는 ‘누구지?’라고 생각 했다.

➤ 결과:

“안녕?”이라는 그 인사를 듣고 나는 ‘누구지?’라고 생각 했다.

문자열 연산

✓ 문자열 연산

덧셈

```
a = 'Hello' + 'Hello'
```

```
print(a)
```

```
>> HelloHello
```

곱셈

```
b = 'Hello' * 3
```

```
print( b )
```

```
>> HelloHelloHello
```

실습02

(1) 다음의 글에서 '너무'를 총 100회 출력되도록 하세요.

➤ 실행 결과:

너무 너무 너무 너무 너무 너무 너무 ... 너무 너무 너무 파이썬 공부가 즐겁습니다.

❖ '너무'는 100번 쓰임

문자열 함수

✓ 문자열 함수1

```
# len(값): 문자 길이
```

```
word = 'apple banana'
```

```
len(word)
```

```
>> 12
```

```
# replace(바꿀 문자, 새 문자): 문자열 바꾸기
```

```
word = 'apple banana'
```

```
print(word.replace('a', '2'))
```

```
>> 2pple b2n2n2
```


문자열 함수

✓ 문자열 함수2

```
# upper(): 소문자를 대문자로 바꿈
```

```
word = 'apple banana'
```

```
print(word.upper())
```

```
>> APPLE BANANA
```

```
# lower(): 대문자를 소문자로 바꿈
```

```
word = 'APPLE BANANA'
```

```
print(word.lower())
```

```
>> apple banana
```

문자열 인덱싱

✓ 인덱스(index)

- 시퀀스 자료형은 순서를 갖기때문에 각 요소마다 순서를 통해 접근 가능
- 이 때 이 순서를 **인덱스**라고 함
- 인덱스는 0부터 시작
- -1은 마지막 항목
- 아래는 시퀀스 인덱스를 지정하여 항목을 가져오는 방법

변수명[인덱스 숫자]

문자열 인덱싱

✓ 문자열 인덱싱

- 문자열 리터럴은 각 하나의 문자들이 합쳐진 시퀀스
- 인덱스를 활용하여 해당 위치에 있는 각각의 문자를 가져올 수 있음

```
word = 'python'
```

```
print(word[0])
```

```
>> p
```

p	y	t	h	o	n
0	1	2	3	4	5
-6	-5	-4	-3	-2	-1

인덱스 에러

✓ 인덱스 에러

- 인덱스는 시퀀스 자료형이 갖는 범위 내에서만 사용 가능
- 이 범위를 넘는 경우엔 IndexError가 발생

```
word = 'python'  
print(word[6])
```

```
>> IndexError: string index out of range
```

p	y	t	h	o	n
0	1	2	3	4	5
-6	-5	-4	-3	-2	-1

실습03

✓[문제]

- sentence라는 명칭의 변수를 선언하고 '중요한 것은 꺾이지 않는 마음'의 문자열 리터럴을 할당하세요.
- 그리고 인덱싱을 활용하여 아래와 같은 결과물을 출력하세요.

➤ 실행결과:

>> 꺾

문자열 슬라이싱

✓ 슬라이싱(slicing)

- 슬라이싱은 총 세 개의 파트로 구성 됨
- 구분자는 콜론(:)
- 형태: [시작 숫자(이상) : 끝 숫자(미만) : 증감 크기]
- 첫 요소부터 포함하는 범위일 때, 시작 숫자는 생략 가능
- 마지막 요소까지 포함하는 범위일 때, 끝 숫자 생략 가능
- 증감 크기가 1일 때, 증감 크기 생략 가능

변수명[시작 : 끝 : 증감]

문자열 슬라이싱

✓ 문자열 슬라이싱(slicing)

- 문자열 리터럴에서 부분 문자열(substring)을 얻기 위한 방법
- 범위에 해당하는 요소들을 가져오는 방법
- 인덱스와 동일하게 대괄호 안에 범위 표현

```
word = 'python'  
print(word[::2])  
>> Pto
```

p	y	t	h	o	n
0	1	2	3	4	5
-6	-5	-4	-3	-2	-1

문자열 슬라이싱

✓ 문자열 슬라이싱 표현 범위

- 슬라이싱은 인덱스 범위를 벗어나도 유연하게 처리됨
- 만약 인덱스 에러처럼 존재하는 인덱스에 대해서만 표현 가능하다면, 값을 포함하지 않는 '끝 숫자'의 범위 표현이 불가능해짐

```
sentence = '중요한 것은 꺾이지 않는 마음'
print(sentence[42:])
print(sentence[2:42])
>>
>> 한 것은 꺾이지 않는 마음
```


실습 04

✓[문제1]

- sentence라는 명칭의 변수를 선언하고 '중요한 것은 꺾이지 않는 마음'의 문자열 리터럴을 할당하세요.
- 그리고 슬라이싱을 활용하여 아래와 같은 결과를 출력해보세요.

➤ 실행결과:

>> 꺾이지 않는 마음

실습 04

✓[문제2]

- sentence라는 명칭의 변수를 선언하고 '중요한 것은 꺾이지 않는 마음'의 문자열 리터럴을 할당하세요.
- 그리고 슬라이싱을 활용하여 아래와 같은 결과를 출력해보세요.

➤ 실행결과:

>> 한요중

서식지정자

✓ 서식 지정자

- 문자열 내에 변수를 포함하거나 포매팅 하는 방법
- 문자열에 값을 삽입하는 방법

✓ 서식 지정자 종류

- %s: 문자열(string) - 모든 객체를 문자열로 변환하여 삽입
- %d: 정수(integer) - 십진수 형식으로 정수 삽입
- %f: 실수(float) - 기본 소수점 이하 6자리까지 표현
- %.nf: 소수점 이하 n자리까지 표현하는 실수

✓ 서식 지정자 방법

- % 포매팅
- str.format() 메서드
- f-string 포매팅

서식지정자1

✓ % 포매팅

“문자열 %s 문자열” % “추가문자”



서식지정자1

✓ %s 기본 사용 방식

%s 포매팅 사용 안 한 경우

```
name = 'tom'
```

```
print('I am ' + name + '!')
```

```
>> I am tom!
```

%s 포매팅 사용

```
name = 'tom'
```

```
print('I am %s!' % name)
```

```
>> I am tom!
```

서식지정자1

✓ %s 응용1

```
# %(정수)s: 공백 추가
name = 'tom'
print('I am %10s!' % name) # 양의 정수: 왼쪽으로 숫자만큼 공백 추가
print('I am %-10s!' % name) # 음의 정수: 오른쪽으로 숫자만큼 공백 추가
>> I am          tom!
>> I am tom      !
```

서식지정자1

✓ %s 응용2

```
# 여러개의 문자 추가
string1 = 'an apple'
string2 = 'banana'
print('I like %s and %s!' %(string1, string2))
>> I like an apple and a banana
```

서식지정자2

✓ %d 기본 사용 방식

```
# %d 포매팅 사용
number = 3
print('Number %d is my favorite number.' % number)
>> Number 3 is my favorite number.
```


서식지정자3

✓ %f 기본 사용 방식

```
# %f 포매팅 사용 (기본적으로 소수점 이하 6자리까지 출력)
number = 3.2323
print('Float %f is my favorite number.' %number)
>> Float 3.232300 is my favorite number.
```

서식지정자3

✓ %f 응용1

```
# %(.정수)f: 출력할 소수점 이하 자리수 지정
number = 3.232323
print('Float %.2f is my favorite number.' % number)
>> Float 3.23 is my favorite number.
```


서식지정자3

✓ %f 응용2

```
# %(0)(채울 너비 수).(소수점 이하 출력 수)f: 남은 너비 만큼 0으로  
채우기  
number = 3.2323  
print('Float %03.6f is my favorite number.' %number)  
>> Float 3.232300 is my favorite number.
```

format 함수

"문자열 {0}, {1} 문자열".format(값, 값)



format함수의 인자값의 인덱스 번호를
문자열 내에 원하는 위치에 넣어줘야 함

format 함수

✓ format 함수 기본 사용 방식

기본 사용 방식

```
print('I like {0} and {1}.'.format("an apple", "a banana"))
```

```
>> I like an apple and a banana
```

변수 활용 방식

```
fruit1, fruit2 = "an apple", "a banana"
```

```
print('I like {0} and {1}.'.format(fruit1, fruit2))
```

```
>> I like an apple and a banana
```

format 함수

✓ format 함수 기본 사용 방식

기본 사용 방식2

```
print('Number {0}, {1}, {2}'.format(4, 5, 6))
```

```
>> Number 4, 5, 6
```

```
print('Number {0}, {0}, {1}'.format(4, 5, 6))
```

```
>> Number 4, 4, 5
```

```
print('Number {}, {}, {}'.format(4, 5, 6))
```

```
>> Number 4, 5, 6
```

f-string

f"문자열 {변수 명} 문자열"

f-string

✓ f-string 기본 사용 방식

```
name = 'Tom'
age = 10
height = 144.5

print(f"My name is {name} and I'm {age} years old")
print(f"My height is {height}cm")
print(f"I will be {age + 1} years old next year")

>> My name is Tom and I'm 10 years old
>> My height is 144.5
>> I will be 11 years old next year
```


실습05

✓ [문제1] 문자를 입력 받아 공백을 모두 제거하고 출력하세요

➤ 실행:

아무 문자나 입력해주세요 : 안녕하세요 반갑습니다.

➤ 결과:

안녕하세요반갑습니다.

실습05

✓ [문제2] 이름과 점수 세 개를 입력 받아 아래와 같이 출력하세요.

➤ 실행:

이름을 입력해주세요: **홍길동**

첫 번째 점수를 입력해주세요: **80**

두 번째 점수를 입력해주세요: **70**

세 번째 점수를 입력해주세요: **60**

저의 이름은 홍길동이고 총점은 210 입니다.

9. 리스트와 튜플

최희운 강사

시퀀스 자료형

✓ 시퀀스 자료형

데이터를 순서대로 나열하여 저장할 수 있는 자료형

✓ 시퀀스 자료형 종류

str(문자열), list(리스트), tuple(튜플), range 등

✓ 시퀀스 특징

- 인덱싱
- 슬라이싱
- 반복 가능 (iterable)
- 길이 확인 가능 (len() 함수 사용 가능)
- 멤버십 테스트 가능 (in 키워드 통해 특정 값 포함 여부 확인 가능)
- +와 *를 통해 연결과 반복 가능

변수 = [요소1, 요소2, 요소3]

리스트 (list)

✓ 리스트 자료형

여러 값을 순서대로 저장할 수 있는 자료형

대괄호 [] 로 생성하며, 콤마(,)로 구분된 여러 개의 값을 담을 수 있음

✓ 리스트 자료형 특징

- 순서가 있음: 순서가 있는 자료형으로, 인덱스를 통해 각 요소에 접근할 수 있으며, 인덱스는 0부터 시작
- 가변성: 리스트를 생성한 후에도 요소를 추가, 변경, 삭제 가능
- 다양한 데이터 타입 저장 가능: 숫자, 문자열, 불리언, 리스트, 튜플, 딕셔너리 등 여러가지의 자료형을 혼합하여 저장할 수 있음

리스트 (list)

```
a = [1, 2, 3, 4, 5]
b = ['apple', 5, 3.14, False, [1, 2], ('a', 'b')] # 다양한 자료형 저장
가능
c = [] # 빈 리스트 생성 가능
d = list() # list class를 통해 list객체 생성 가능 (빈 리스트 생성 가능)
e = list((1, 2, 3)) # iterable(반복 가능한) 객체를 list로 변환할 수 있음

print(d)
print(e)
>> []
>> [1, 2, 3]
```

리스트 (list) 인덱스 접근

✓ 인덱스

- 시퀀스 자료형은 순서를 갖기 때문에 각 요소마다 이 순서를 통해 접근 가능하며, 이 순서를 **인덱스**라고 함
- 인덱스를 통해 특정 요소에 접근하거나 일부를 추출 및 변경할 수 있음
- 요소의 순번은 앞에서 부터 0 으로 시작
- 뒤에서 부터는 -1로 시작

```
fruit = ['apple', 'banana', 'cherry']
```

0

1

2

-3

-2

-1

리스트 (list) 인덱스 접근

✓ 인덱싱 및 슬라이싱

인덱싱

```
fruit = ['apple', 'banana', 'cherry']
```

```
print(a[0])
```

```
print(a[1])
```

```
print(a[3])
```

```
>> apple
```

```
>> banana
```

```
>> IndexError: list index out of range
```


슬라이싱

✓ 슬라이싱

- 범위에 해당하는 요소들을 가져오는 방법
- 인덱스와 동일하게 대괄호 안에 범위를 표현
- 대괄호 안에 '시작 인덱스 : 끝 인덱스' 형태로 작성

```
fruit = ['apple', 'banana', 'cherry']
```

0	1	2
-3	-2	-1

```
fruit[0:2] # 0~1을 의미
```

시작 인덱스 끝 인덱스

```
>> ['apple', 'banana']
```

리스트 (list) 주요 기능

✓ 인덱싱 및 슬라이싱

슬라이싱: fruit[시작 숫자: 끝 숫자]일 경우 [시작 숫자 ~ 끝 숫자-1] 까지의 요소를 리스트로 추출

```
fruit = ['apple', 'banana', 'cherry']
```

```
print(fruit[0:3]) # 0번~2번째 요소 추출
```

```
>> ['apple', 'banana', 'cherry']
```

```
print(fruit[1:3]) # 1번~2번째 요소 추출
```

```
>> ['banana', 'cherry']
```

```
print(fruit[2:]) # 2번~마지막 요소 추출
```

```
>> ['cherry']
```

```
print(fruit[:3]) # 첫 요소~2번째 요소 추출
```

```
>> ['apple', 'banana']
```

리스트 (list) 인덱스 접근

✓ 다양한 인덱스 접근 방법

증가폭 변경

```
a = [1, 2, 3, 4, 5, 6, 7]
```

```
print(a[2:8:2]) # 0번~7번째 까지 2씩 건너뛰어 추출
```

```
>> [2, 4, 6]
```

슬라이스 요소 할당

```
a[1:4] = ['a', 'b', 'c']
```

```
print(a)
```

```
>> [1, 'a', 'b', 'c', 5, 6, 7]
```

슬라이스 삭제

```
del a[1:4]
```

```
print(a)
```

```
>> [1, 5, 6, 7]
```

리스트 (list) 다양한 기능들 1

```
a = [1, 2, 3]
```

```
# 특정 값 있는지 확인하기
```

```
print(1 in a)
```

```
>> True
```

```
print(9 in a)
```

```
>> False
```

```
# 요소 개수 구하기
```

```
print(len(a))
```

```
>> 3
```

```
# 연결하기
```

```
b = [4, 5, 6]
```

```
print(a + b)
```

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

```
# 반복하기
```

```
print(a * 3)
```

```
>> [1, 2, 3, 1, 2, 3, 1, 2, 3]
```

리스트 (list) 다양한 기능들 2

✓ 요소 추가하기 1

```
a = [1, 2, 3]
```

```
b = [1, 2, 3]
```

```
# .append(추가하고자 하는 값)
```

```
a.append(4)
```

```
print(a)
```

```
>> [1, 2, 3, 4]
```

```
a.append([5, 6])
```

```
print(a)
```

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

```
a.append((6, 7))
```

```
print(a)
```

```
>> [1, 2, 3, 4, [5, 6], (6, 7)]
```

```
# .extend(iterable 객체)
```

```
b.extend([4, 5])
```

```
>> [1, 2, 3, 4, 5]
```

```
b.extend((6, 7))
```

```
>> [1, 2, 3, 4, 5, 6, 7]
```

리스트 (list) 다양한 기능들 2

✓ 요소 추가하기 2

```
a = [1, 2, 3]
```

```
# .insert(인덱스, 추가하고자 하는 값)
```

```
a.insert(1, 100)
```

```
>> [1, 100, 2, 3]
```

```
a.insert(3, 200)
```

```
>> [1, 100, 2, 200, 3]
```

리스트 (list) 다양한 기능들 3

✓ 요소 삭제

```
a = [1, 2, 3, 4, 5]
```

```
# .pop(인덱스)
```

```
# 해당 요소 삭제 후 삭제된 요소 반환
```

```
print(a.pop(0))
```

```
>> 1
```

```
print(a)
```

```
>> [2, 3, 4, 5]
```

```
# 마지막 요소 삭제 후 삭제된 요소 반환
```

```
print(a.pop())
```

```
>> 5 # [2, 3, 4]만 남음
```

```
# .remove(값)
```

```
# 해당 값을 찾아 리스트에서 삭제
```

```
a.remove(3)
```

```
print(a)
```

```
>> [2, 4]
```

리스트 (list) 다양한 기능들 4

```
a = [1, 2, 3, 3, 4, 5]
```

```
# .index(값): 특정 값의 인덱스 구하기
```

```
print(a.index(3))
```

```
>> 2
```

```
# .count(값): 특정 값의 개수 구하기
```

```
print(a.count(3))
```

```
>> 2
```

```
# .reverse(): 특정 값의 개수 구하기
```

```
a.reverse()
```

```
print(a)
```

```
>> [5, 4, 3, 3, 2, 1]
```

```
b = [3, 5, 1, 6]
```

```
# .sort(): 정렬 (default=오름차순)
```

```
b.sort()
```

```
print(b)
```

```
>> [1, 3, 5, 6]
```

```
# .sort(reverse=True): 내림차순 정렬
```

```
b.sort(reverse=True)
```

```
print(b)
```

```
>> [6, 5, 3, 1]
```


Immutable 객체

✓ 파이썬의 Immutable(불변성) 객체

- 객체가 생성된 이후에 변경될 수 없는 성질을 의미
- 파이썬에서 Immutable객체는 값을 수정할 수 없으며, 객체를 변경하려고 하면 새로운 객체가 생성됨
- 숫자형(int, float, 등), 문자열, 튜플, 등

리스트 (list) 할당과 복사

✓ 할당

- 변수에 값을 저장하는 과정
- 파이썬에서 변수는 '객체에 대한 이름'으로 동작하며, 실제 데이터는 메모리의 특정 위치에 저장 됨
- 할당은 객체의 참조를 변수에 연결하는 것과 같음

✓ 참조

- 파이썬에서 모든 변수는 '**객체의 참조**'를 저장함 (=할당)
- 참조는 **변수 이름과 메모리의 데이터 객체를 연결**
- 하나의 객체를 여러 변수가 참조할 수 있음
- 이는 할당에서 기본적으로 발생

리스트 (list) 할당과 복사

✓ 복사

- 객체를 복제하여 별도의 메모리 공간에 새로운 객체를 생성하는 작업

✓ 얇은 복사

- 객체의 1차원 데이터만 복사
- 내부적으로 포함된 객체들은 원래 객체를 참조
- 즉, 새로운 객체가 생성되지만, 그 내부의 하위 객체는 원래 객체와 공유

✓ 깊은 복사

- 객체를 완전히 복제하여 독립적인 객체 생성
- 원본 객체와 복사본이 전혀 연결되지 않으며, 하위 객체까지 모두 복제됨

리스트 (list) 할당과 복사

✓ 할당

```
a = [1, 2, 3, 4, 5]
b = a
print(a is b)
>> True

b[0] = 10
print(a)
>> [10, 2, 3, 4, 5]
```

- a라는 리스트를 생성 후 a를 다른 변수(b)에 할당하면, 두 개의 리스트가 생기는 것이 아님.
- 실제로 메모리에 존재하는 리스트는 1개임
- 이를 **할당**이라고 함
- 두 개의 변수(a, b)는 각각 메모리를 차지하고 있는 것이 아닌, 동일한 메모리를 가리키고 있음

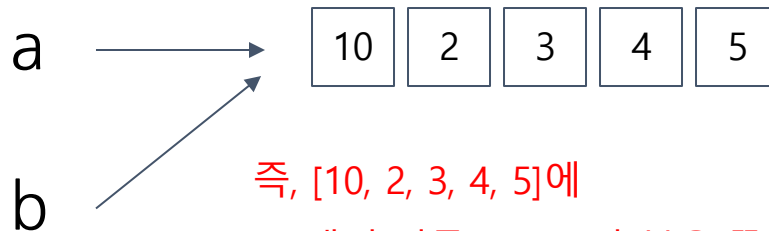
리스트 (list) 할당과 복사

✓ 할당

```
a = [1, 2, 3, 4, 5]
b = a
print(a is b)
>> True
```

```
b[0] = 10
print(a)
>> [10, 2, 3, 4, 5]
```

b를 바꾸면 a도 바뀐
즉, 복사(복제)가 되지 않음



즉, [10, 2, 3, 4, 5]에
두 개의 이름표(a, b)가 붙은 꼴이 됨
(즉, 객체의 참조를 복사)

리스트 (list) 할당과 복사

✓ 얇은 복사: `.copy()` 함수로 객체의 1차원 데이터 복사(복제)

```
a = [1, 2, 3, 4, 5]
```

```
b = a.copy()
```

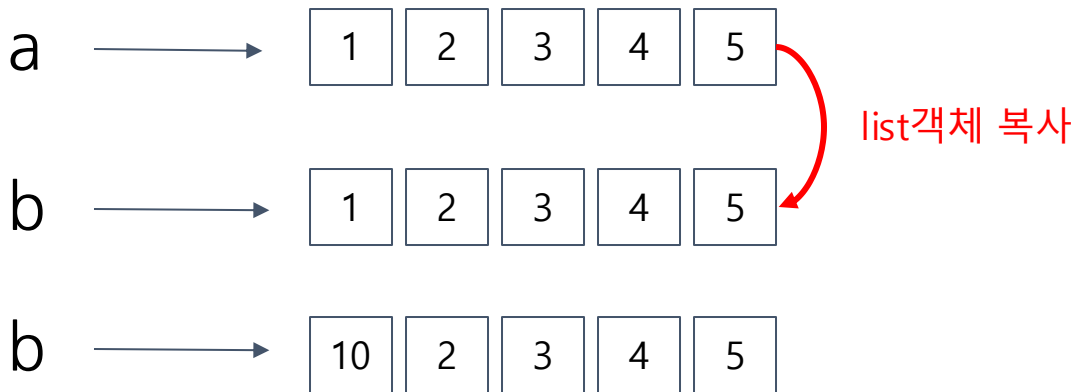
```
print(a is b)
```

```
>> False
```

```
b[0] = 10
```

```
print(a)
```

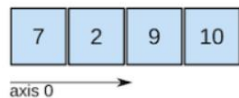
```
>> [1, 2, 3, 4, 5]
```



다차원 리스트 (list)

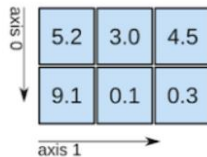
- ✓ 다차원 리스트는 리스트 안에 리스트가 있는 구조로, 고차원 데이터를 다루는데 효과적
- ✓ 2차원 리스트는 흔히 행렬로 표현이 되며 3차원, 4차원 등의 고차원 리스트 또한 구현 가능

1D array



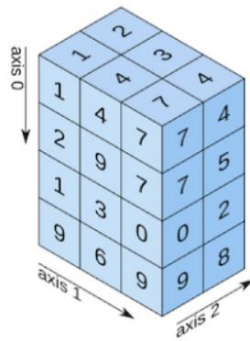
shape: (4,)

2D array



shape: (2, 3)

3D array



shape: (4, 3, 2)

다차원 리스트 (list)

✓ a = [[값, 값], [값, 값], [값, 값]]

2차원 리스트

```
a = [[1, 2], [3, 4], [4, 5]]
```

2차원 리스트의 특정 요소에 접근

```
print(a[0][1])
```

```
>> 2
```

```
print(len(a))
```

```
>> 3
```

3차원 리스트

```
a = [[[1, 2], [3, 4], [4, 5]],  
      [[6, 7], [8, 9], [10, 11]]]
```

2차원 리스트의 특정 요소에 접근

```
print(a[0][1][1])
```

```
>> 4
```

```
print(len(a))
```

```
>> 2
```


다차원 리스트 (list) 복사

✓ 2차원 이상의 리스트는 copy클래스의 .deepcopy()메소드 사용하여 복사

list클래스의 copy 메소드 사용할 경우

```
a = [[1, 2], [3, 4], [4, 5]]
```

```
b = a.copy()
```

```
b[0][0] = 10
```

```
print(a)
```

```
>> [[10, 2], [3, 4], [4, 5]]
```

copy 클래스 import

```
import copy
```

```
a = [[1, 2], [3, 4], [4, 5]]
```

```
b = copy.deepcopy(a)
```

```
b[0][0] = 10
```

```
print(a)
```

```
>> [[1, 2], [3, 4], [4, 5]]
```

```
print(b)
```

```
>> [[10, 2], [3, 4], [4, 5]]
```

튜플 (tuple)

변수 = (요소1, 요소2, 요소3)

변수 = 요소1, 요소2, 요소3

튜플 (tuple)

✓ 튜플 자료형

- 리스트와 비슷한 데이터의 목록
- 소괄호 () 로 생성하며, 콤마(,)로 구분된 여러 개의 값을 담을 수 있음
- 소괄호는 생략 가능

✓ 튜플 자료형 특징

- 순서가 있음: 순서가 있는 자료형으로, 인덱스를 통해 각 요소에 접근할 수 있으며, 인덱스는 0부터 시작
- 불변의 객체: 값 변경 및 삭제 불가능
- 다양한 데이터 타입 저장 가능: 숫자, 문자열, 불리언, 리스트, 튜플, 딕셔너리 등 여러가지의 자료형을 혼합하여 저장할 수 있음

튜플(tuple)을 쓰는 이유

✓ 범용적인 리스트 대신 튜플 자료형을 쓰는 이유

- 리스트는 크기가 바뀔 수 있기 때문에 그에 따른 초과 할당 까지 고려해서 메모리에 데이터를 저장할 수 있을지 신경 써야 함
- 튜플은 빠르게 생성할 수 있고 리스트보다 메모리 부담이 적은 대신 그 내용을 변경할 수 없음

➤ 아래의 데이터를 리스트와 튜플 중 어디에 담는 게 좋을 지 생각해보기

1. 프로그래밍 언어의 종류
2. 사람의 나이, 키, 몸무게
3. 사람의 생일과 출생지
4. 롤 게임의 결과
5. 잇단 롤게임의 결과

1. 리스트: 데이터가 계속 증가할 수 있으므로
2. 리스트: 값을 계속 갱신해야 하므로
3. 튜플: 변하지 않는 정적 데이터이므로
4. 튜플: 정적인 데이터이므로
5. 리스트: 게임을 플레이할 때 마다 새로운 결과가 추가되므로

튜플(tuple) 생성하기 1

```
a = (1, 2, 3, 4, 5)
```

```
b = 6, 7, 8, 9, 10
```

```
# 타입 확인
```

```
print(type(a))
```

```
>> <class 'tuple'>
```

```
print(type(b))
```

```
>> <class 'tuple'>
```

튜플(tuple) 생성하기 2

다른 자료형들의 모임

```
a = ('하나', 123, True)
```

```
print(a)
```

```
>> ('하나', 123, True)
```

```
print(type(a))
```

```
>> <class 'tuple'>
```

튜플이 튜플을 가지는 경우

```
b = 'id', ('kim', 'lee'), (True, False)
```

```
print(b)
```

```
>> ('id', ('kim', 'lee'), (True, False))
```

```
print(type(b))
```

```
>> <class 'tuple'>
```

튜플(tuple) 생성하기 3

튜플인것

```
a = 1,  
b = (10, )
```

```
print(type(a))  
>> <class 'tuple'>  
print(type(b))  
>> <class 'tuple'> #逗号为空时
```

튜플이 아닌것

```
c = 1  
d = (10)
```

```
print(type(c))  
>> <class 'int'>  
print(type(d))  
>> <class 'int'> #逗号为空时
```

튜플(tuple)의 다양한 기능들 1

```
a = (1, 2, 3, 4, 5)
```

```
b = 6, 7, 8, 9, 10
```

요소 개수 확인

```
print(len(a), len(b))
```

```
>> 5 5
```

인덱싱

```
print(a[0], a[1], a[2])
```

```
>> 1 2 3
```

```
print(a[-1], a[-2])
```

```
>> 5 4
```

슬라이싱

```
print(a[1:3])
```

```
>> (2, 3)
```

```
print(type(a[1:3]))
```

```
>> <class 'tuple'>
```

```
print(a[2:])
```

```
>> (3 4 5)
```

```
print(type(a[2:]))
```

```
>> <class 'tuple'>
```


튜플(tuple)의 다양한 기능들 2

```
a = (1, 2, 3)
```

```
b = 'a', 'b', 'c', 'd'
```

```
# 연결
```

```
c = a + b
```

```
print(c)
```

```
>> (1, 2, 3, 'a', 'b', 'c', 'd')
```

```
# 반복
```

```
d = b * 3
```

```
print(d)
```

```
>> ('a', 'b', 'c', 'd', 'a', 'b', 'c', 'd', 'a', 'b', 'c', 'd')
```

불변의 객체 튜플

```
a = (1, 2, 3, 4)
```

```
# 요소 변경
```

```
a[0] = 10
```

```
print(a)
```

```
>> TypeError: 'tuple' object doesn't ...
```

```
# 요소 삭제
```

```
del a[0]
```

```
>> TypeError: 'tuple' object doesn't ...
```

불변의 객체 튜플

```
a = (1, 2, 3, 4)
```

요소 변경 방법: 새로운 객체 생성해야 함

```
a = ('A',) + a[1:]
```

```
print(a)
```

```
>> ('A', 1, 2, 3, 4)
```

두 번째 값 변경 방법

```
a = (a[0],) + ('A',) + a[2:]
```

```
print(a)
```

```
>> (1, 'A', 3, 4)
```

불변의 객체 튜플

```
a = (1, 2, 3, 4)
```

```
# 두 번째 요소 삭제 방법
```

```
a = (a[0],) + a[2:]
```

```
print(a)
```

```
>> (1, 3, 4)
```

변수 여럿에 동시 할당하기

✓ 동시 할당(unpacking)

- 시퀀스 자료형의 크기 만큼 좌변의 변수 개수를 동일하게 선언하면 시퀀스 자료형의 요소 각각을 변수에 동시할당 할 수 있음
- 마치 좌측에 변수들과 값 개수를 동할하게 맞춰 선언한 것처럼 보임



$x1, x2, x3 = \text{요소1}, \text{요소2}, \text{요소3}$

소괄호가 생략된 튜플

입력 함수 input()

- ✓ 파이썬 입력함수 input() 사용해보기
 - input()은 내장함수를 사용해서 사용자로부터 값을 받아올 수 있음
 - 이 때 받아온 값은 문자열로 받음
 - input('메시지') -> return 사용자가 입력한 값 (str)

```
input_value = input('Message') # 1111을 입력해보자

print("input_value: ", input_value, "\ntype: ", type(input_value))

>> inpout_value: 1111
>> type: <class 'str'>
```

입력 함수 input()

- ✓ 파이썬 입력함수 input() 사용해보기
 - 사용자가 입력한 값을 정수로 변환하려면 int() 내장 함수를 사용하면 됨

```
input_value = int(input_value) # input_value는 1111을 입력 받았었음

print("input_value: ", input_value, "\ntype: ", type(input_value))
>> inpout_value: 1111
>> type: <class 'int'>
```

입력 함수 input()

✓ 파이썬 입력함수 input() 사용해보기

- 사용자가 입력한 값이 1.1일 때 int() 내장 함수를 활용하여 정수로 변환하려면 에러 발생

```
input_value2 = input('정수 외에 다른 값을 입력해보세요.') # 1.1을 입력해보자
```

```
>> 정수 외에 다른 값을 입력해보세요. 1.1
```

```
input_value2 = int(input_value)
```

```
>> ValueError: invalid literal for int() with base 10: '1.1'
```


입력 값을 여러 개의 변수에 저장하기: split()

✓ split() 활용 해보기

- 한 번에 두 개의 입력을 받기 위해 문자열의 split() 함수를 활용해보기
- split() 함수는 인자로 전달된 구분자를 활용하여 문자열을 쪼개는 함수
- 그 결과를 list 자료형으로 반환
- 구분자를 인자로 전달하지 않는 경우엔 공백을 기준으로 문자열을 쪼갬

```
result = '1, 2, 3'.split(',') # 구분자: 쉼표(,)
print(type(result))
print(result)

>> <class 'list'>
>> ['1', '2', '3']
```

Page 10 of 10

- ✓ map() 내장함수

- iterable 객체(반복 가능한 객체)에서 요소를 하나하나 꺼내어 특정 함수를 각각 적용하고 싶을 때 사용
- 반환은 map 클래스를 반환하는데 , 이는 iterable한 객체
- list() 함수 또는 tuple()함수를 활용하여 map 클래스를 list 혹은 tuple로 변경 가능

map(function, iterable) → <class 'map'>
iterable 객체

map() 내장함수 이용

✓ map() 내장함수

- iterable 객체(반복 가능한 객체)에서 요소를 하나하나 꺼내어 특정 함수를 각각 적용하고 싶을 때 사용
- 반환은 map 클래스를 반환하는데 , 이는 iterable한 객체
- list() 함수 또는 tuple()함수를 활용하여 map 클래스를 list 혹은 tuple로 변경 가능

```
a = [1.2, 2.5, 3.7, 4.6]
```

```
a = list(map(int, a)) # 반환된 map객체를 list객체로 변환
```

```
print(a)
```

```
>> [1, 2, 3, 4]
```

실습01

✓ [문제]

- 사용자가 숫자를 입력하면

range함수를 이용하여 사용자가 입력한 숫자까지의 2의 배수로 구성된 리스트를 만드세요.

그리고 리스트의 맨 마지막에 사용자가 입력한 숫자도 추가해주세요.

➤ 실행:

숫자를 입력해주세요: **10**

➤ 결과:

[2, 4, 6, 8, 10, 10]

➤ 실행:

숫자를 입력해주세요: **20**

➤ 결과:

[2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 20]

실습02

✓ [문제]

- 사용자의 이름을 물어보고 이어서 2개의 정수를 받아서 덧셈을 한 후 결과를 출력하는 프로그램을 작성해보세요.

➤ 실행:

이름을 입력하시오: **엔코아**

엔코아씨, 안녕하세요?

파이썬에 오신 것을 환영합니다.

첫 번째 정수를 입력하세요: **300**

두 번째 정수를 입력하세요: **400**

300과 400의 합은 700 입니다.

실습03

✓ [문제]

- 사용자로부터 "hh:mm:ss" 형태의 시간 값을 입력 받습니다.
- 예를 들면 12시 33분 23초일 때 "12:22:23"과 같이 입력 받습니다.
- 이 때 결과가 아래와 같이 나오도록 구현해보세요.

➤ 실행:

hh:mm:ss의 형태로 시간을 입력해주세요: 12:33:23

➤ 실행 결과:

시: 12

분: 33

초: 23

실습04

✓ [문제]

- 사용자로부터 값을 입력 받습니다.
- 국어, 영어, 수학, 과학 점수를 한 번에 입력 받습니다.
- 이를 kor, eng, mat, sci라는 변수에 각각 저장하고, 사칙 연산을 활용하여 평균 점수를 산출해보세요

➤ 실행:

국어, 영어, 수학, 과학 점수를 입력해주세요: 98,100,96,100

➤ 실행 결과:

평균 점수는 98.5입니다.

10. 덕셔너리

최희운 강사

딕셔너리 (dictionary)

- ✓ 딕셔너리 자료형(Dictionary type)
 - 비 시퀀스 자료형
 - 따라서 인덱스를 갖는 대신 고유한 키(key)값을 가짐
 - 이 키(key)는 1:1로 매칭되는 값(value)를 가짐 (이를 key-value pair 라고 함)
 - **중괄호 {}**를 활용하여 딕셔너리 자료형 생성
 - 중괄호 안에는 **key : value 형태**로 작성
 - key-value pair는 **쉼표(,)로 구분**지어 요소로 등록할 수 있음

변수 = { Key : Value }

변수 = dict(Key = Value)

딕셔너리 (dictionary)

✓ 딕셔너리 생성 실습

```
score1 = {'name':'Tom', 'math':80, 'english':70}  
score2 = dict(name='Tome', math=80, english=70)
```

```
print(score1['name'])
```

```
>> Tom
```

```
print(type(score1))
```

```
>> <class 'dict'>
```

딕셔너리 (dictionary)

✓ 딕셔너리 특징

- 키(key)는 중복해서 존재하지 않음
- 즉, 고유한 값을 가져야 함 (인덱스에 겹치는 숫자가 없듯이)
- 중복해서 넣을 시 가장 최근에 넣은 key-value pair만 남음
- 딕셔너리의 키(key)는 문자열(str), 정수(int), 실수(float), 불(bool), 튜플(tuple)을 사용할 수 있음
- 단, 키(key)에 리스트(list), 딕셔너리(dictionary), 집합(set)을 사용할 수 없음

```
people = {'name': 'Tom', 'name': 'Amy'}  
print(people)  
>> {'name': 'Amy'}
```

```
dic_var = {'key': 'value',  
           1: 'value2',  
           1.1: 'value3',  
           True: 'value4',  
           ('key', 1, 1.1, True): 'value5'}  
print(list(map(type, dic_var.keys())))
```

딕셔너리 (dictionary)

✓ 딕셔너리 특징

- 딕셔너리 자료형의 값(value)는 모든 자료형을 사용할 수 있음

```
dic_var = {'key': 'value',  
          2: 3,  
          1.1: 1.1,  
          True: False,  
          ('key', 1, 1.1, True): ('key', 1, 1.1, True),  
          'list': [1, '가', 1.1, True, (1, '나')],  
          'dict': {'key': 'value', 'key2': 'value2'},  
          'set': {'key', 'value'}}  
  
print(list(map(type, dic_var.values())))
```

딕셔너리 (dictionary)

- ✓ 딕셔너리 키(key) 통해 값(value) 에 접근하기

```
score = {'name':'Tom', 'math':80, 'english':70 }
```

```
# value 에 접근
```

```
print(score['name'])
```

```
>> 'Tom'
```

딕셔너리 (dictionary)

- ✓ 딕셔너리 키(key) 통해 값(value) 변경하는 방법

```
score = {'name': 'Tom', 'math': 80, 'english': 70 }
```

```
# value 변경해보기
```

```
score['name'] = 'michale'
```

```
print(score['name'])
```

```
>> michale
```

딕셔너리 기능 1

```
score = {'name': 'Tom', 'math': 80, 'english': 70 }
```

```
# key가 있는지 확인: in  
print('math' in score)  
>> True
```

```
print('age' in score)  
>> False
```

```
# key 개수 확인: len()  
print(len(score))  
>> 3
```

```
# key-value쌍 추가하기: .setdefault()  
score.setdefault('age', 20)  
print(score)  
>> {'name': 'Tom', 'math': 80, 'english': 70, 'age': 20}
```

```
# key-value 수정하기: .update({key: value})  
score.update({'math': 90})  
print(score)  
>> {'name': 'Tom', 'math': 90, 'english': 70, 'age': 20}
```

딕셔너리 기능 2

```
score = {'name':'Tom', 'math':80, 'english':70 }
```

```
# key로 딕셔너리 항목 삭제: .pop(삭제할 key)
```

```
print(score.pop('name'))
```

```
>> Tom
```

```
print(score)
```

```
>> {'math':80, 'english':70 }
```

```
# age에 해당하는 key 없으면 0 반환
```

```
print(score.pop('age', 0))
```

```
>> 0
```

```
# 모든 값 삭제: .clear()
```

```
score.clear()
```

```
print(score)
```

```
>> {}
```

```
# 모든 key, value 가져오기: .keys(), values()
```

```
print(score.keys())
```

```
>> dict_keys(['name', 'math', 'english'])
```

```
print(score)
```

```
>> dict_values(['Tom', 80, 70])
```


딕셔너리 할당과 복사

딕셔너리 복사: `.copy()`

```
a = {'a': 0, 'b': 1}
```

```
b = a.copy()
```

```
print(b)
```

```
>> {'a': 0, 'b': 1}
```

중첩 딕셔너리 복사: `copy.deepcopy()`

```
import copy
```

```
a = {'a': {'c': 0, 'd': 0}, 'b': 1}
```

```
b = copy.deepcopy(a)
```

```
print(b)
```

```
>> {'a': {'c': 0, 'd': 0}, 'b': 1}
```

zip() 내장함수

✓ zip() 내장함수

- 키와 값 한 쌍을 갖는 시퀀스 객체를 요소로 갖는 시퀀스 객체를 생성하여 딕셔너리 자료형으로 만들 수 있음
- 딕셔너리 자료형의 키와 값 한 쌍을 갖는 튜플을 요소로 갖는 객체(아래 예제에는 리스트)로 딕셔너리 자료형을 생성할 수 있음

zip(["Key1", "key2"] , ["value1", "value2"])



("Key1", "value1"), ("key2", "value2")

dict(zip(["Key1", "key2"] , ["value1", "value2"]))

실습01

- ✓ [문제] 이름, 나이, 연락처를 받아 딕셔너리를 생성하여 출력해주세요.

➤ 실행:

이름을 입력해주세요: **홍길동**

나이를 입력해주세요: **27**

연락처 입력해주세요: **010-1234-6789**

➤ 결과:

{'이름': '홍길동', '나이': '27', '연락처': '010-6768-7574'}

실습02

✓ [문제] 두 사람의 이름, 나이, 연락처를 받아 두 개의 딕셔너리를 생성 후 리스트에 담아주세요.

➤ 실행:

이름을 입력해주세요: **홍길동**

나이를 입력해주세요: **27**

연락처 입력해주세요: **010-1234-6789**

이름을 입력해주세요: **이몽룡**

나이를 입력해주세요: **30**

연락처 입력해주세요: **010-1122-3344**

➤ 결과:

```
[{'이름': '홍길동', '나이': '27', '연락처': '010-6768-7574'}, {'이름': '이몽룡', '나이': '30', '연락처': '010-1122-3344'}]
```

실습03

✓ [문제]

- input() 함수를 통해 원하는 과일 이름을 입력하면 가격이 출력되는 프로그램을 구현해봅시다.
- 과일 이름(str)을 key로 만들고 가격(int)을 value로 해서 딕셔너리 자료형을 만듭니다.
- 사과는 1000원, 바나나는 700원, 오렌지는 1500원, 파인애플은 2000원

➤ 실행:

과일 이름을 입력하세요: **사과**

사과의 가격은 1000원 입니다.

실습04

✓ [문제]

- input() 함수를 통해 이름 값 여럿을 입력 받습니다.
- 입력 받고 난 후에 이름 순서와 개수를 동일하게 몸무게 값을 입력 받습니다.
- 이름(str)을 키(key)로, 몸무게(float)를 값(value)으로 하는 딕셔너리 자료형을 만들고 그 결과를 출력해보세요

➤ 실행 결과:

이름을 기입해주세요: **뽀로로 루피 크롱 스누피**

몸무게를 입력해주세요: 20.4 16.2 22.3 5.2

11. 집합(set)

최희윤 강사

집합 (set)

변수 = {값1, 값2, 값3}

변수 = set([값1, 값2, 값3])

- ✓ 집합(set) 자료형은 중복을 허용하지 않는 자료형이다.
- ✓ 입력된 순서 또한 집합(set) 자료형에서는 중요하지 않다. (unordered)
- ✓ set() 생성자 함수를 사용하거나 {}(중괄호)를 사용하여 만들 수 있다.
- ✓ 주로 교집합, 합집합, 차집합을 구할 때 유용하게 사용할 수 있다.

집합 (set)

변수 = {값1, 값2, 값3}

변수 = set([값1, 값2, 값3])

```
animal1 = {'dog', 'cat', 'monkey', 'horse'}  
animal2 = set(['dog', 'cat', 'monkey', 'horse'])
```

```
print(type(animal1))  
>> <class 'set'>
```

```
print(type(animal2))  
>> <class 'set'>
```

집합 (set)

변수 = {값1, 값2, 값3}

변수 = set([값1, 값2, 값3])

```
animal1 = {'dog', 'cat', 'monkey', 'horse'}  
animal2 = set(['dog', 'cat', 'monkey', 'horse'])
```

```
print(animal1)  
>> {'dog', 'cat', 'monkey', 'horse'}
```

```
print(animal2)  
>> {'dog', 'cat', 'monkey', 'horse'}
```

집합 (set)

집합 자료형 접근

```
animal = {'dog', 'cat', 'monkey', 'horse'}
```

```
my_list = list(animal)
```

```
print(my_list[0])
```

집합은 순서가 보장되지 않기 때문에 list로 변환 시 순서가 임의로 정해짐

```
>> horse
```

집합 연산 1

```
s1 = set([1, 2, 3, 4, 5, 6])
```

```
s2 = set([4, 5, 6, 7, 8, 9])
```

```
# 교집합: & / .intersection(s2)
```

```
print(s1 & s2)
```

```
print(s1.intersection(s2))
```

```
>> {4, 5, 6}
```

```
>> {4, 5, 6}
```

```
# 합집합: | / .union()
```

```
print(s1 | s2)
```

```
print(s1.union(s2))
```

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

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

집합 연산 2

```
s1 = set([1, 2, 3, 4, 5, 6])
```

```
s2 = set([4, 5, 6, 7, 8, 9])
```

```
# 차집합: - / .difference()
```

```
print(s1 - s2)
```

```
print(s1.difference(s2))
```

```
>> {1, 2, 3}
```

```
>> {1, 2, 3}
```

```
print(s2 - s1)
```

```
print(s2.difference(s1))
```

```
>> {8, 9, 7}
```

```
>> {8, 9, 7}
```

집합의 다양한 기능

```
s1 = set([1, 2, 3])
```

```
# 값 추가: .add()
```

```
s1.add(4)
```

```
print(s1)
```

```
>> {1, 2, 3, 4}
```

```
# 값 수정: .update()
```

```
s1.update([4, 5, 6])
```

```
print(s1)
```

```
>> {1, 2, 3, 4, 5, 6}
```

집합의 다양한 기능

```
s1 = set([1, 2, 3])
```

```
# 값 제거: .remove()
```

```
s1.remove(2)
```

```
print(s1)
```

```
>> {1, 3}
```

```
s1 = set([1, 2, 3])
```

```
# 값 수정: .discard()
```

```
s1.discard(44)
```

```
print(s1)
```

```
>> {1, 2, 3}
```