

강사 소개 (유종훈)

- 강의 경력

- 교보생명 교보 디지털 사내 자격 인증 과정 (2022.5~2022.6)
- 스타트업 AI 기술인력 양성사업 이어드림 스쿨 (2022.4)
- 우리 FIS 디지털 프론티어 3기 (2022.4)
- 미래에셋증권 파이썬 Lv2 (2022.3)
- KB 증권 신입사원 교육 (2022.1)
- 한라인재개발원 Work Smart 프로그램 (2021.11)
- KB증권 파이썬을 활용한 웹크롤링 (2021.10)
- 매일유업 빅데이터 솔루션 심화과정 (2021.10)
- 농심 DIA GreedSeed 심화과정 (2021.10)
- 현대캐피탈 파이썬을 활용한 실무 데이터 분석 (2021.09)
- 한국투자증권 신입사원 과정 (2021.07)
- KB금융그룹 KB 데이터 분석 Academy 6기 (2021.06)
- 현대캐피탈 파이썬 실무 데이터 분석 (2021.2)
- 서울산업진흥원 SSAC 비즈니스 빅데이터 분석 과정 (2021.1)

강사 소개 (유종훈)

- 강의 경력

- 금융데이터 수집 및 리포트 자동화 과정 (삼성금융연수원, 2020 10월)
- ACE Academy (KB금융그룹, 2020 9월)
- 금융데이터 분석 입문 과정 (KB국민은행, 2020 8월)
- 빅데이터 분석 Academy (삼천리 자전거, 2020 6월)
- 빅데이터 분석 전문가 과정 (하나카드, 2020 6월)
- 파이썬을 활용한 금융 데이터 분석 및 시각화 (삼성증권, 2020)
- 청년 AI 빅데이터 아카데미 (텍스트 분석) (3기수, 포스코, 2020)
- 데이터 분석을 위한 파이썬 (현대자동차, 2019 11월)
- 파이썬을 활용한 데이터 분석 (신한카드, 2019 9월)
- 파이썬으로 배우는 퀀트 투자 (한화투자증권, 2019 9월)
- 데이터 분석 실무과정 (삼성증권 4기수, 2018~2019)
- 금융 데이터 분석 입문 과정 (미래에셋대우 6기수 진행, 2017~2019)
- 파이썬 판다스 라이브러리를 이용한 전략 백테스팅 및 분석 (한국투자증권, 2019)
- 파이썬과 딥러닝 (신한금융그룹, 2019)
- 파이썬과 로보어드바이저 (신한금융그룹, 2019)

집필 서적

- 그림으로 배우는 파이썬 기초 문법
- 할 수 있다. 비트코인 자동 매매
- 금융 데이터 분석을 위한 파이썬 판다스



Introduction

Python을 활용한 자료구조 이해하기

커리큘럼

- 데이터를 효과적으로 관리하기 위한 자료구조 심화 과정

주제	내용	시수
Overview	- 파이썬과 메모리 - 자료구조의 필요성	2H
클래스 심화	- 클래스와 메모리 - 부모 클래스와 자식 클래스 - 자료구조와 클래스의 관계	5H
문자열	- 문자열의 메서드 - 패턴 검색이 가능한 정규식	3H
리스트	- 리스트의 개념 및 특징 - 리스트 컴프리헨션 - 대용량 데이터 처리하기 - [실습] 문자열, 파일, 리스트 활용하여 데이터 뽑기	3H

주제	내용	시수
튜플	- 튜플 개념 및 특징 - 데이터 패킹과 언패킹	2H
딕셔너리	- 딕셔너리 개념 및 특징 - 딕셔너리와 메서드	3H
Set	- Set 자료구조 - Frozenset 자료구조	1H
Tree	- Tree 기초 - Tree 사용 이유	1H
Mini project	- 방대한 데이터의 전처리 실습 - 데이터 검색과 추가 기능 구현 - 자료구조로 성능 개선 확인	4H

미니콘다

- Miniconda
 - 라이브러리를 포함한 파이썬 설치 파일
 - BSD-3를 기반의 라이선스로 무료 사용 가능
 - <https://docs.conda.io/en/latest/miniconda.html>

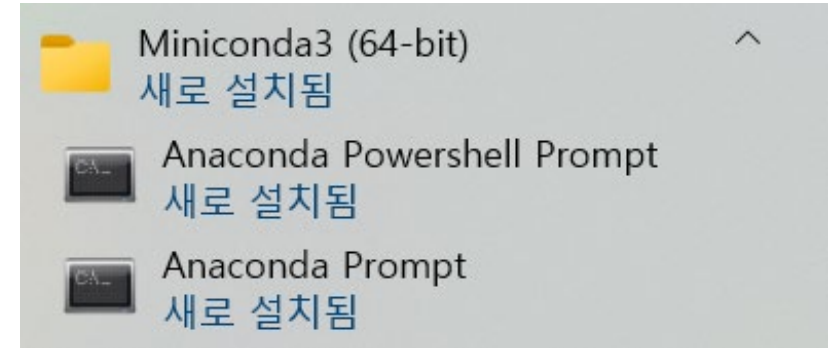
Miniconda

Miniconda is a free minimal installer for conda. It is a small, bootstrap version of Anaconda that includes only conda, Python, the packages they depend on, and a small number of other useful packages, including pip, zlib and a few others. Use the `conda install` command to install 720+ additional conda packages from the Anaconda repository.

[See if Miniconda is right for you.](#)

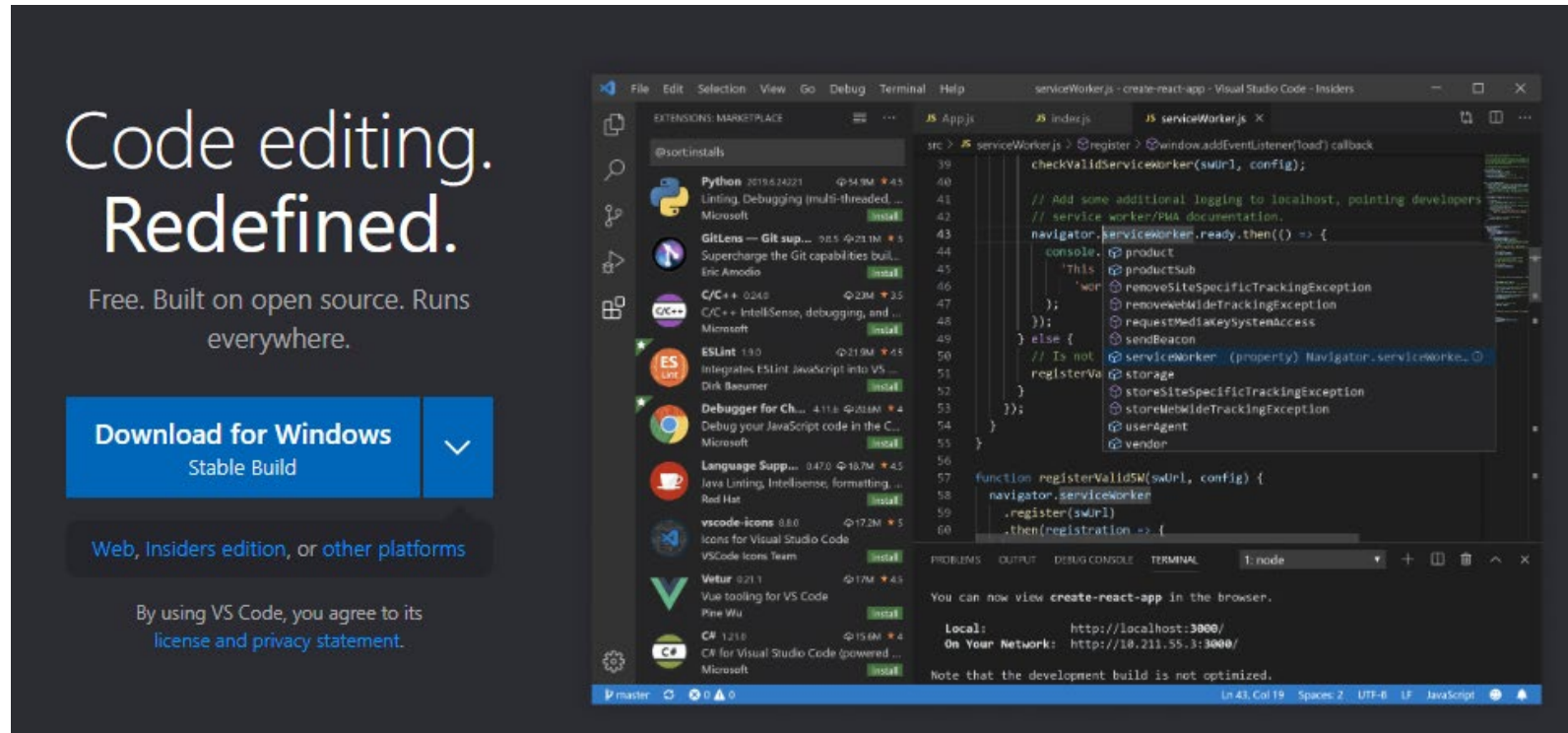
미니콘다 환경 만들기

- 미니콘다 설치 후, Anaconda prompt 에서 작업하기
 - Command line 서두에 환경이름 확인 가능
 - Defalu Env : Base
- Conda environment 생성하기
 - # **conda create -n [내 환경 이름] python=3.10**
 - # **conda activate [내 환경 이름]**



개발 환경 (IDE)

- Python In VS Code
 - 설치링크: <https://code.visualstudio.com/>
 - Python Extension 추가 설치 필요



VScode - Extension

The diagram illustrates the steps to install the Python extension in Visual Studio Code:

- VS Code Interface:** The left sidebar shows the standard VS Code icons. The **Extensions** icon (a square with four smaller squares) is highlighted with a red dashed box.
- Extensions View:** Clicking the Extensions icon opens the **EXTENSIONS: MARKETPLACE** view. The search bar contains the text **python**. The search results show the **Python** extension by **Microsoft**, which provides **IntelliSense (Pylance), Linting, Debugging, and more**.
- Extension Details:** Clicking on the Python extension opens its details page. The page shows the **Python** extension by **Microsoft**, version **v2022.20.2**. It has **74,184,958** downloads and a **5-star** rating (524 reviews). The description states: **IntelliSense (Pylance), Linting, Debugging (multi-threaded, remote), Jupyter Notebooks, code formatting, refactoring, unit tests, and more.** The extension is currently **enabled globally**. The page includes tabs for **Details**, **Feature Contributions**, **Changelog**, **Extension Pack**, and **Runtime Status**.

The **Python extension for Visual Studio Code** is described as a **Visual Studio Code extension** with rich support for the **Python language** (for all **actively supported versions** of the language: **>=3.7**), including features such as **IntelliSense (Pylance)**, **linting**, **debugging**, **code navigation**, **code formatting**, **refactoring**, **variable explorer**, **test explorer**, and **more!**

Support for vscode.dev

The Python extension does offer **some support** when running on vscode.dev (which includes github.dev). This includes partial **IntelliSense** for open files in the editor.

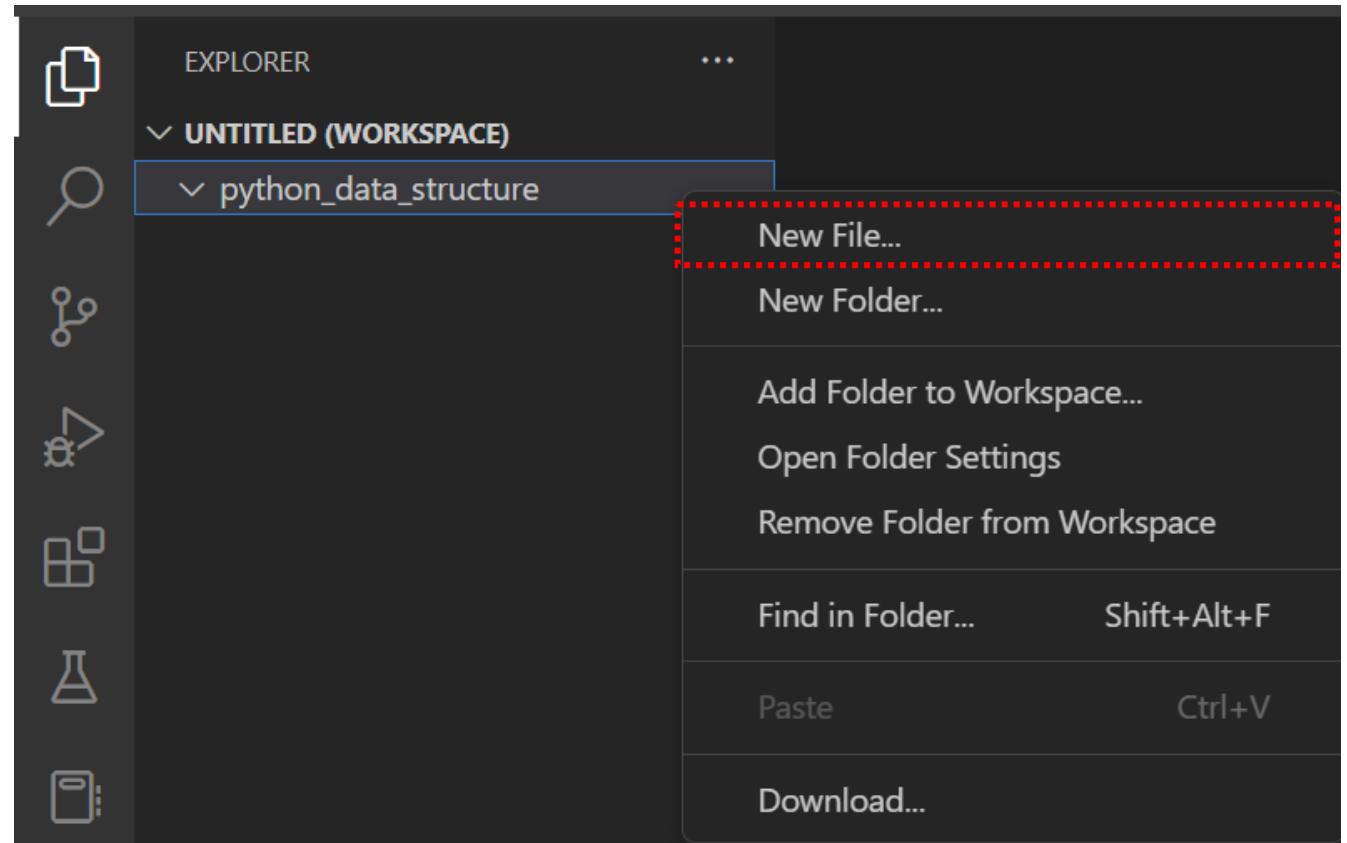
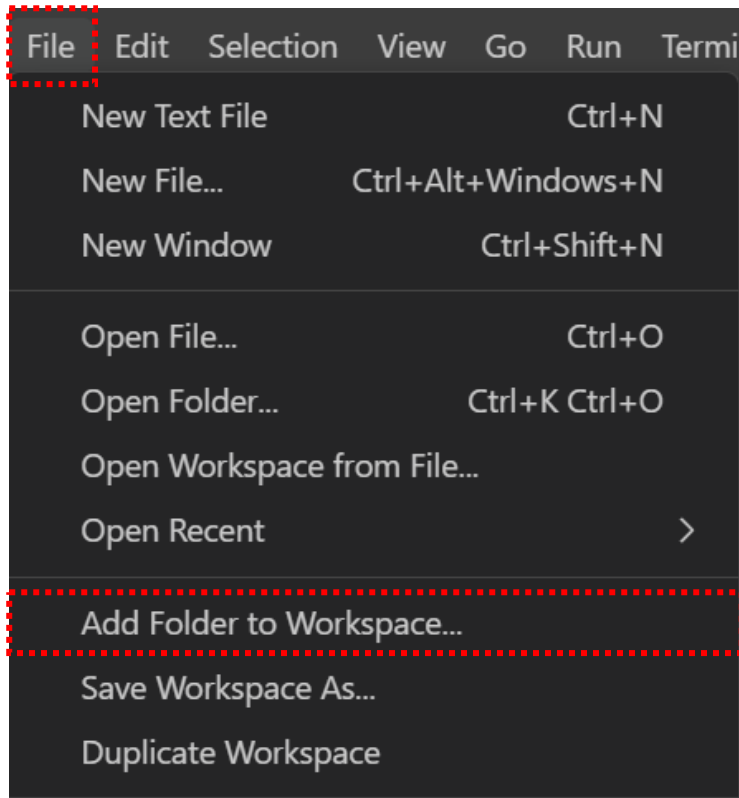
Installed extensions

Categories: Programming Languages, Debuggers, Linters, Formatters, Other, Data Science, Machine Learning, Notebooks.

Extension Resources: Marketplace, Repository, License, Microsoft.

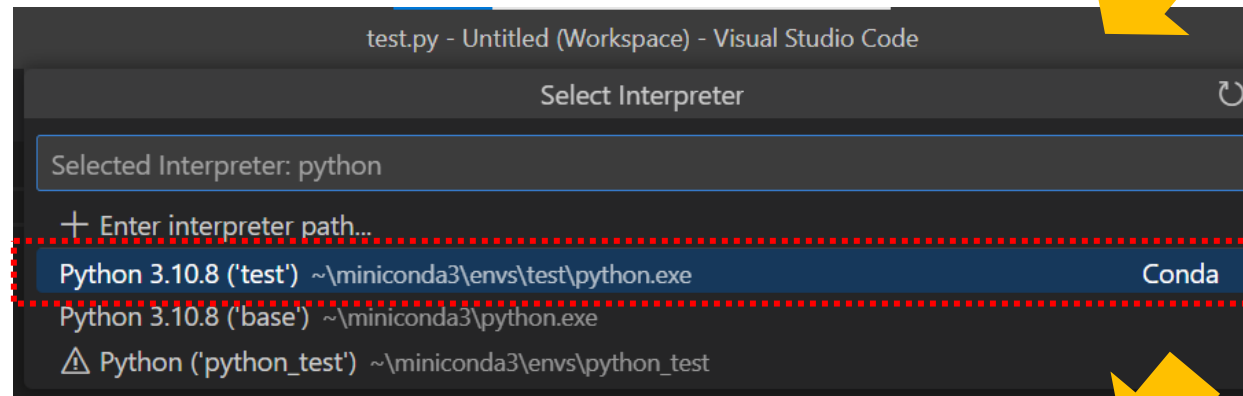
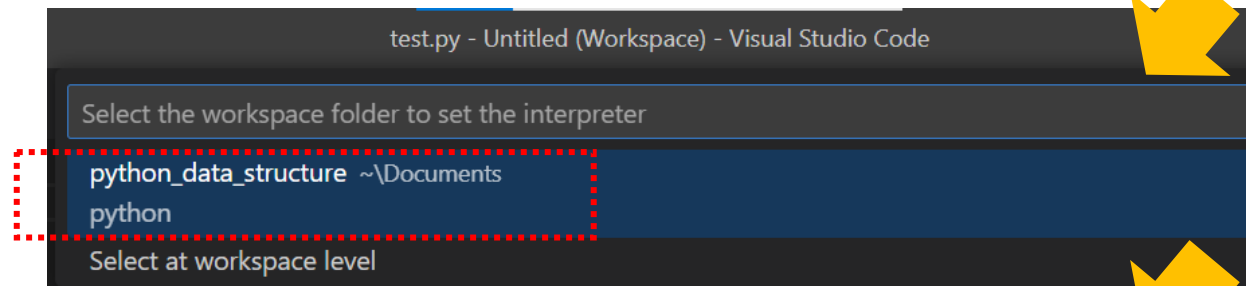
VScode – Workspace

- Workspace - 우리가 사용할 코드들이 모여있는 공간

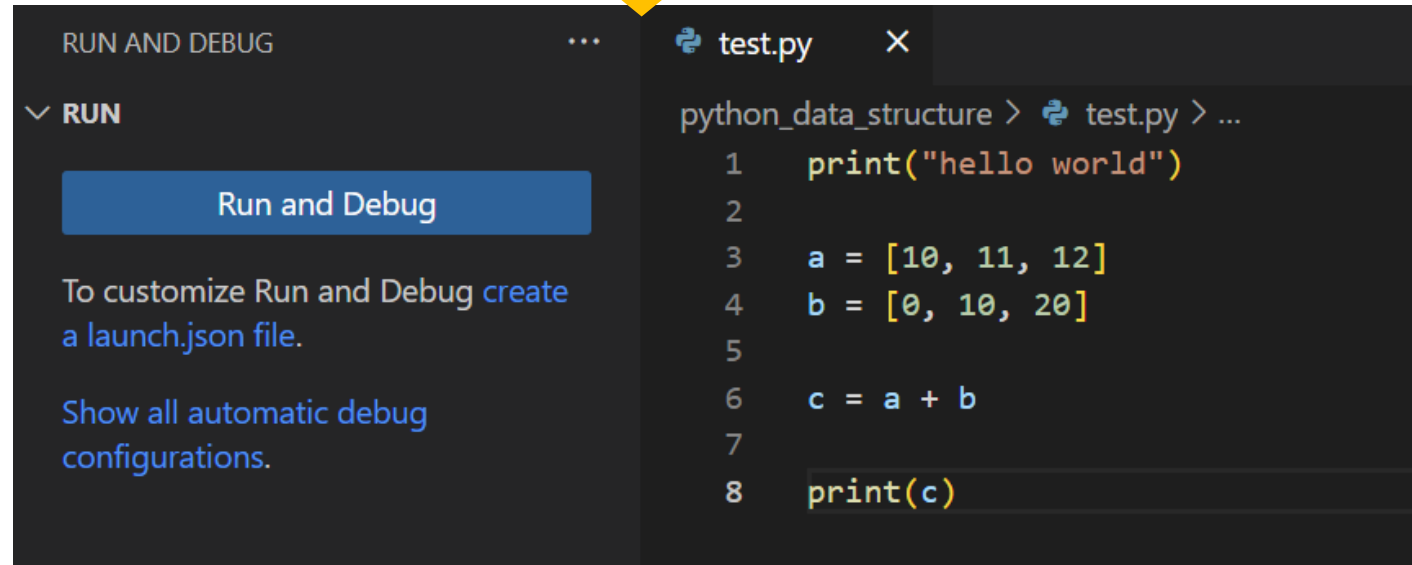
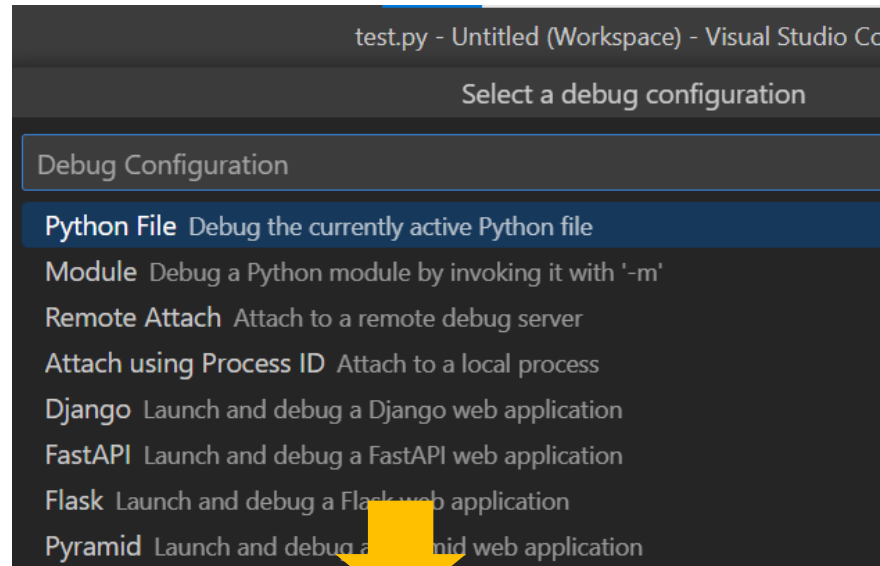


Vscode - Interpreter

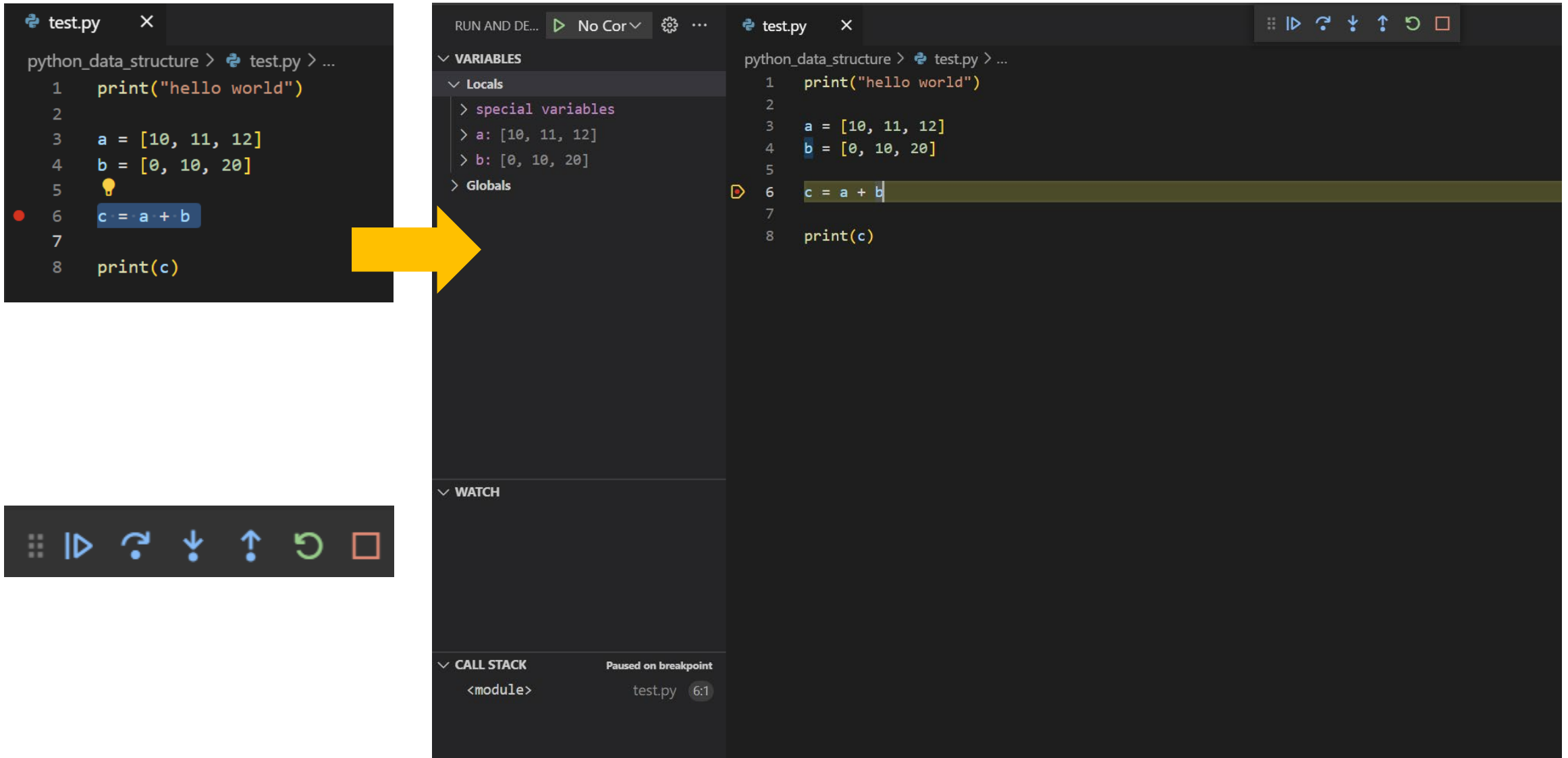
- Vscode 에서 python을 사용하기 위해 python interpreter 를 setting



VScode – Debug mode



VScode - breakpoint



The image illustrates how to set a breakpoint in VS Code. On the left, a code editor shows a Python file named `test.py` with the following code:

```
1 print("hello world")
2
3 a = [10, 11, 12]
4 b = [0, 10, 20]
5
6 c = a + b
7
8 print(c)
```

A red dot on line 6 indicates a breakpoint is set. A yellow arrow points from this breakpoint to the right-hand editor view. The right-hand view shows the same code, but with a yellow arrow pointing to line 6, which is highlighted. The left-hand view shows the code with a red dot on line 6. The right-hand view shows the code with a yellow arrow pointing to line 6, which is highlighted. The left-hand view shows the code with a red dot on line 6.

Below the code editor, a toolbar contains several icons: a grid icon, a play icon, a refresh icon, a download icon, an upload icon, a green circular arrow icon, and a red square icon.

The right-hand view also shows the **VARIABLES** panel, which is expanded to show **Locals**. It displays the following variables:

- `special variables`
- `a: [10, 11, 12]`
- `b: [0, 10, 20]`
- `Globals`

The **WATCH** panel is also visible, showing the variable `c` with the value `[10, 11, 12]`.

The **CALL STACK** panel at the bottom shows the current call stack, indicating the program is **Paused on breakpoint** at `test.py 6:1`.

파이썬과 메모리

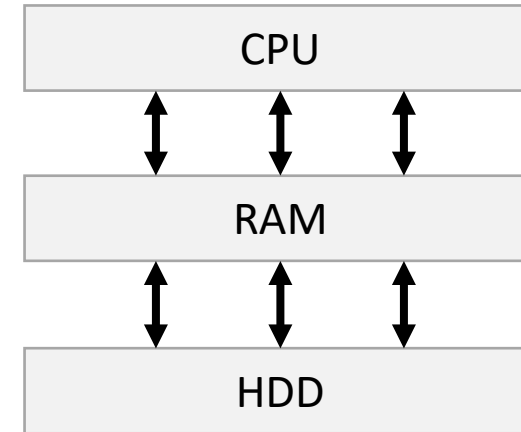
Python을 활용한 자료구조 이해하기

메모리란? (1/2)

- RAM(Random Access Memory)
 - CPU가 주로 사용하는 데이터 저장 장치
 - 메인 메모리는 전원이 차단되면 데이터는 제거



삼성전자 PC133 SDR SDRAM



메모리란? (2/2)

- 메모리의 이해를 돕기 위한 비유

- ✓ 창고

- 많은 양의 빵을 만들어 저장하는 공간

- ✓ 진열대

- 고객에게 빵을 보여주는 목적
 - 예쁘게 빵을 진열하기 위한 공간

- ✓ 하드 디스크 (HDD)

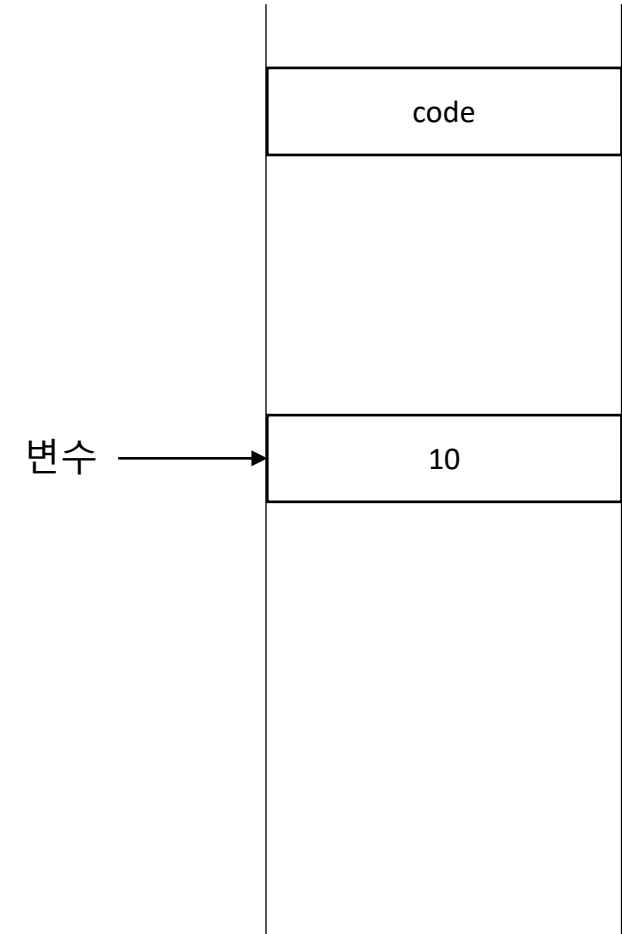
- 전원이 꺼져도 데이터를 유지
 - 방대한 양의 데이터를 저장

- ✓ 메모리

- 비교적 적은 양의 데이터를 저장
 - CPU가 빠르게 데이터를 가져갈 수 있는 장치

파이썬과 메모리

- 모든 데이터는 메모리에 로드 됨
 - 파이썬 인터프리터는 메모리에 데이터를 읽고 씀
 - 프로그램 실행 시 하드디스크에 저장된 코드도 메모리에 로드
 - 파이썬에서 사용하는 데이터는 메모리에 저장



데이터와 메모리

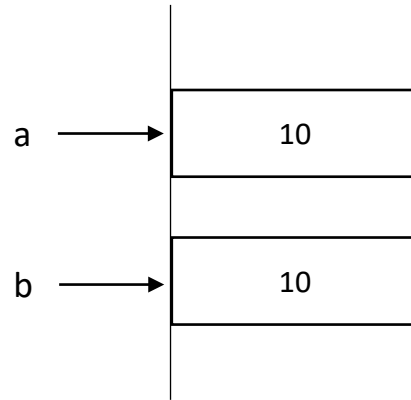
- 변수 a와 b가 저장된 메모리의 상태로 옳은 것은?

```
a = 10
```

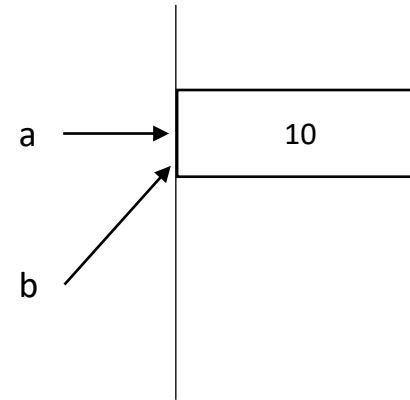
```
b = 10
```

```
print( id(a), id(b) )
```

※ id 함수는 메모리의 주소를 출력



(a)



(b)

연습 문제

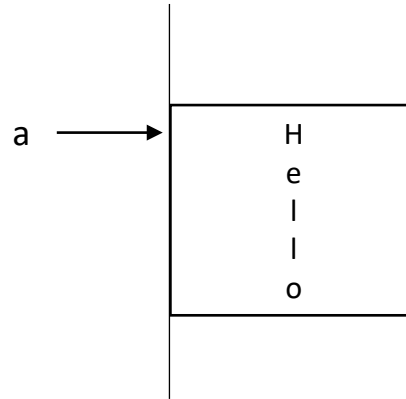
- 다음 코드를 실행했을 때의 메모리의 상태를 그리시오.

```
a = 10  
b = 10  
b = a + b + 2
```

문자열과 메모리

- 문자열이 저장된 메모리를 변수 a가 바인딩

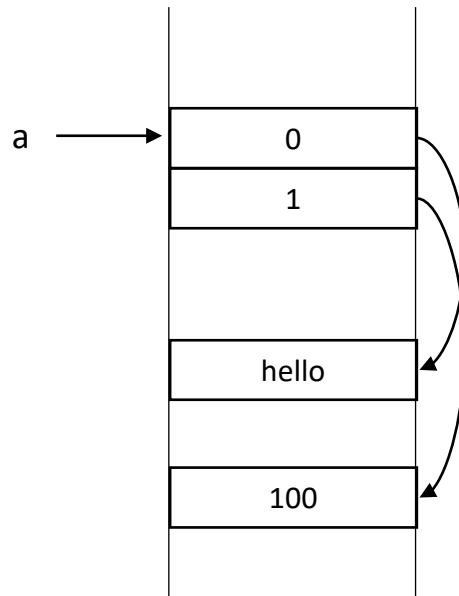
```
a = "Hello"
```



리스트와 메모리

- 다양한 데이터 타입을 저장
 - 가변 데이터를 효율적으로 저장하기 위한 구조

```
a = [ "hello", 100 ]
```

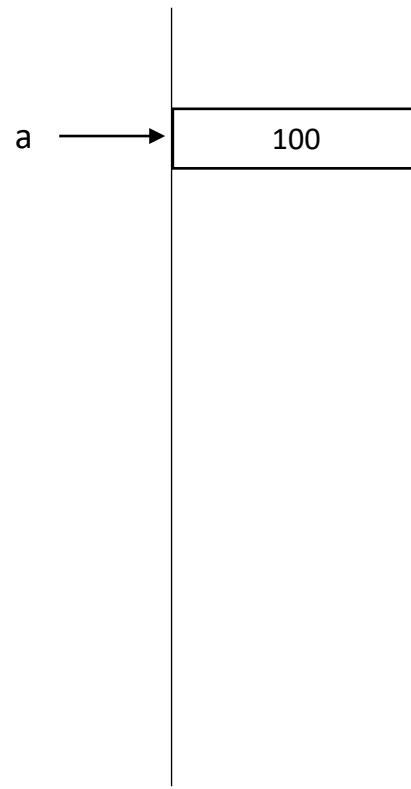


함수와 메모리 (1/3)

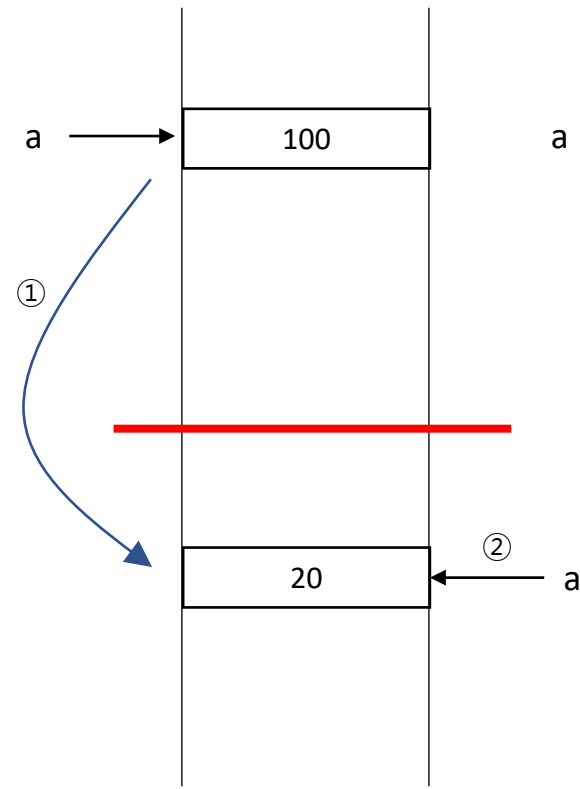
- 함수는 기본적으로 독립된 메모리 영역에서 실행
 - 실행 후 미사용 변수 제거

```
def func( ):  
    a = 20
```

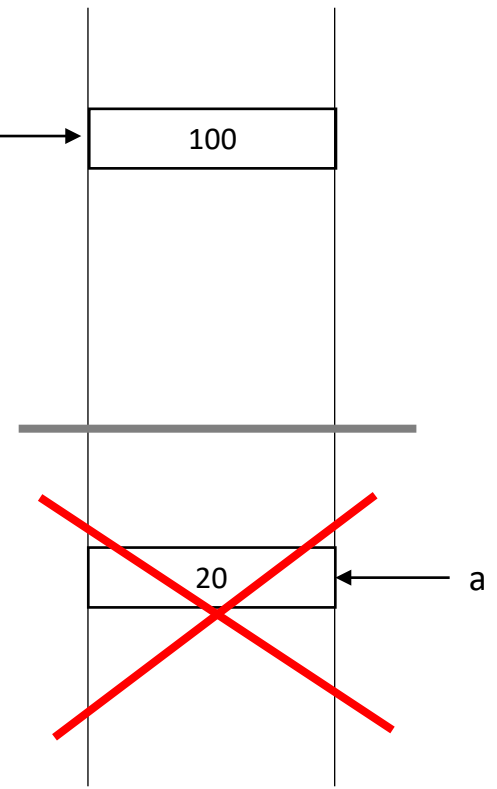
```
a = 100  
func()  
print(a)
```



함수 호출 전



함수 내부



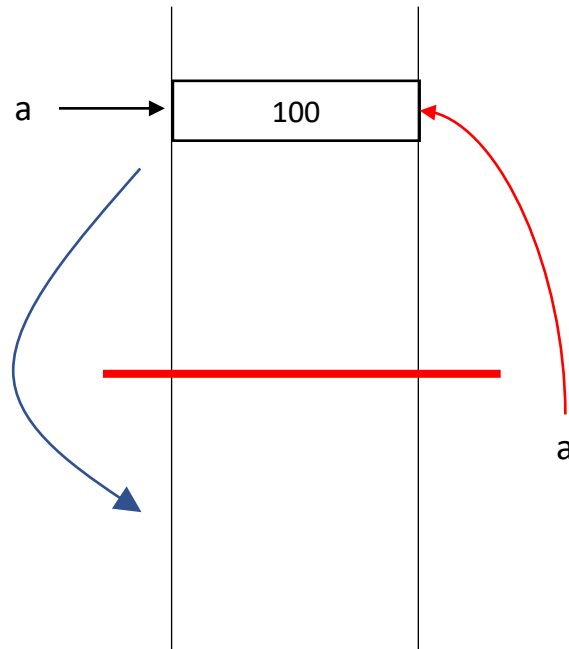
함수 호출 후

함수와 메모리 (2/3)

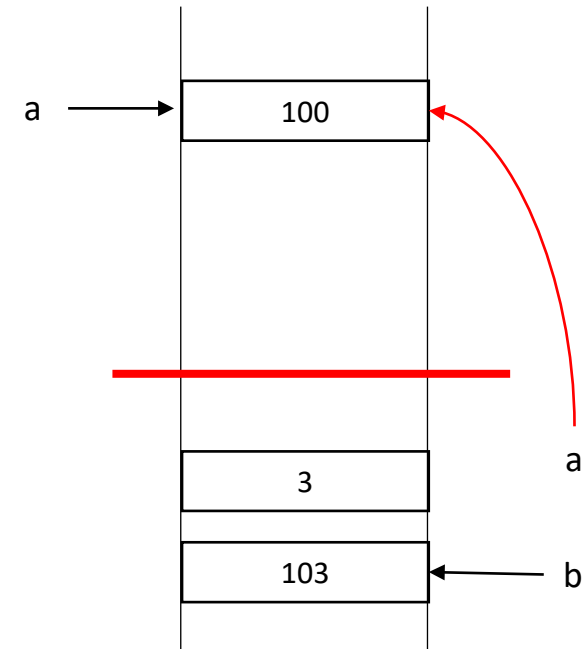
- 입력이 있는 함수와 메모리
 - 함수 호출 시 지역 변수가 전달 한 값을 바인딩

```
def func( a ) : # 1)  
    b = a + 3    # 2)  
    print( b )
```

```
a = 100  
func( a )
```



함수 내부 #1



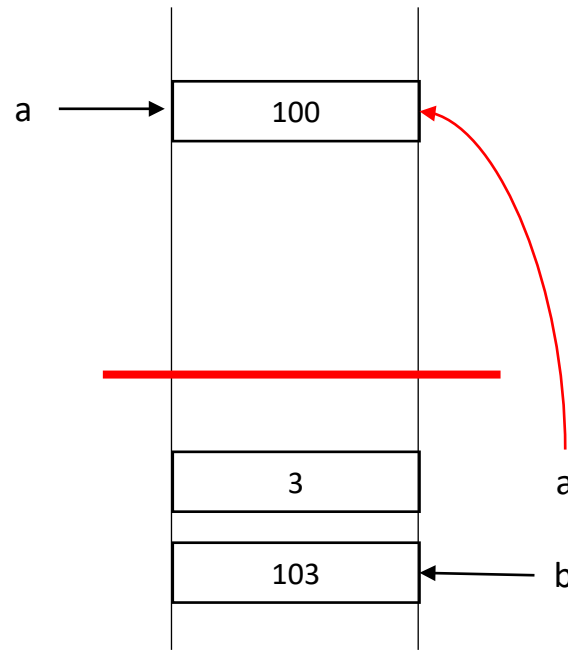
함수 내부 #2

함수와 메모리 (3/3)

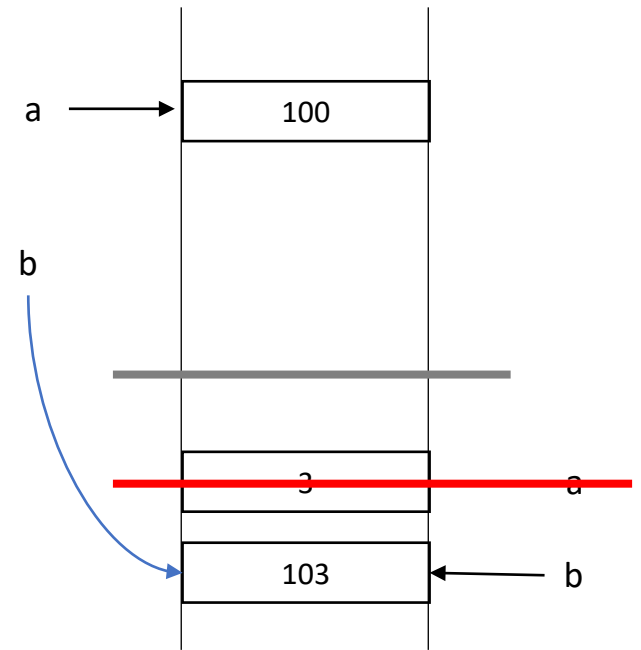
- 반환 값이 있는 함수와 메모리
 - return은 함수 외부에서 데이터를 사용한다고 알림
 - 따라서 메모리의 데이터를 제거하지 않음

```
def func( a ) : # 1)  
    b = a + 3    # 2)  
    return b
```

```
a = 100  
b = func( a )  
print( b )
```



함수 내부 #2



함수 호출 후

파이썬 클래스

Python을 활용한 자료구조 이해하기

클래스를 왜 사용하는가? (1/3)

- 도서관에 가면 책이 분야별로 분류돼 있다.
 - 같은 주제의 책을 10단위로 영역을 나누는 '한국식 십진 분류표'(KDC)를 사용
 - 000 총류, 100 철학, 200 종교, 300 사회과학, 400 순수과학 등등



300 사회과학

- 310 통 계 학
- 320 경 제 학
- 330 사회학,사회문제
- 340 정 치 학
- 350 행 정 학
- 360 법 학
- 370 교 육 학
- 380 풍속,예절,민속학
- 390 국방,군사학



400 자연과학

- 410 수 학
- 420 물 리 학
- 430 화 학
- 440 천 문 학
- 450 지 학
- 460 광 물 학
- 470 생명과학
- 480 식 물 학
- 490 동 물 학



클래스를 왜 사용하는가? (2/3)

- 클래스 안에 비슷한 기능의 함수를 정리한다.

```
def 통계학 (입력):  
    return 출력
```

```
def 경제학 (입력):  
    return 출력
```

```
def 정치학 (입력):  
    return 출력
```

class **사회과학:**

```
def 통계학 (입력):  
    return 출력
```

```
def 경제학 (입력):  
    return 출력
```

```
def 정치학 (입력):  
    return 출력
```

300 사회과학

310 통 계 학

320 경 제 학

330 사회학,사회문제

340 정 치 학

350 행 정 학

360 법 학

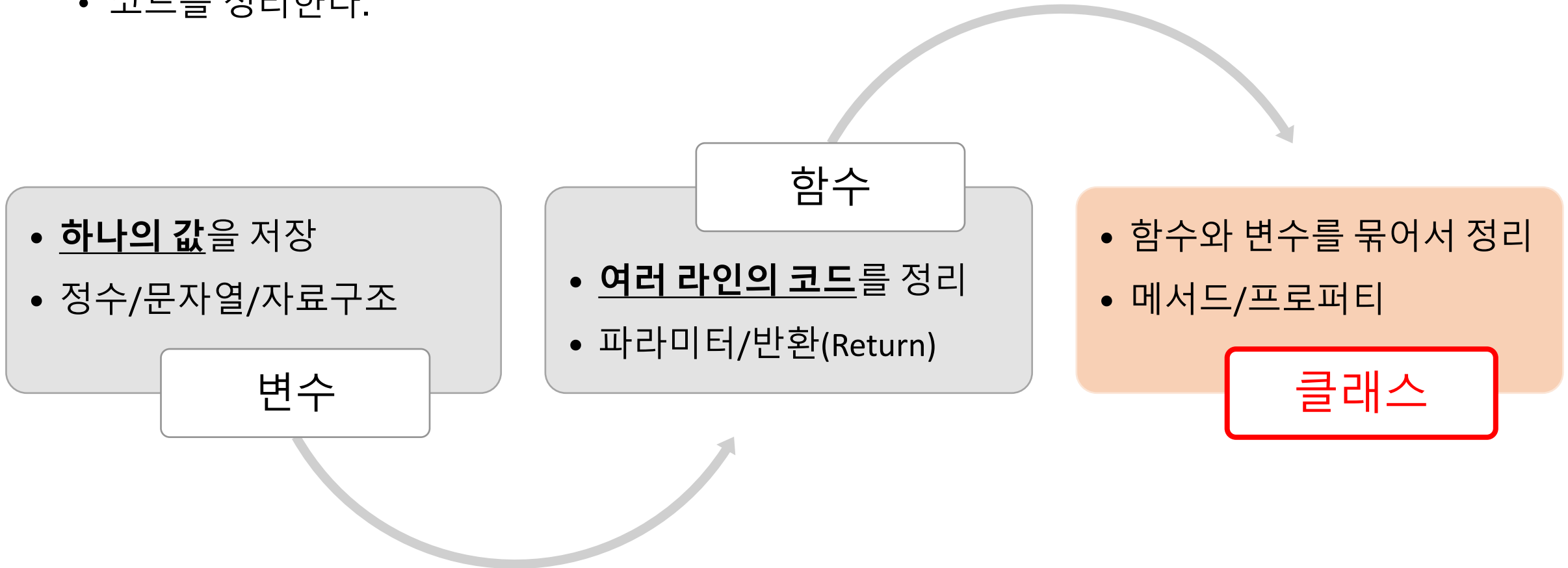
370 교 육 학

380 풍속,예절,민속학

390 국방,군사학

클래스를 왜 사용하는가? (3/3)

- 코드를 정리한다.



절차 지향 프로그래밍

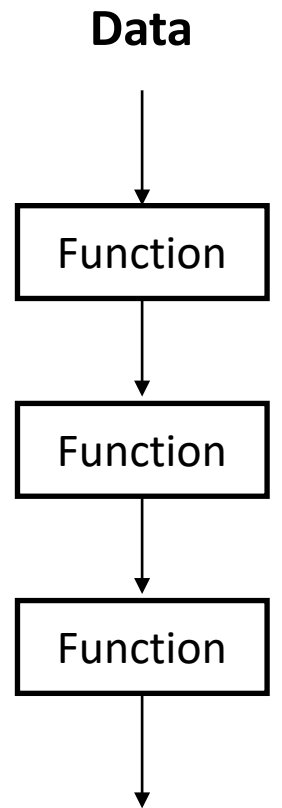
- 절차 지향 프로그래밍
 - 지금까지 해왔던 프로그래밍 방식
 - 데이터와 데이터를 처리하는 함수

↓

```
mario = [0, 0]

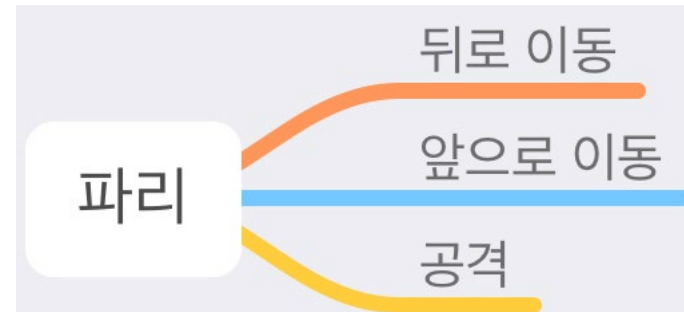
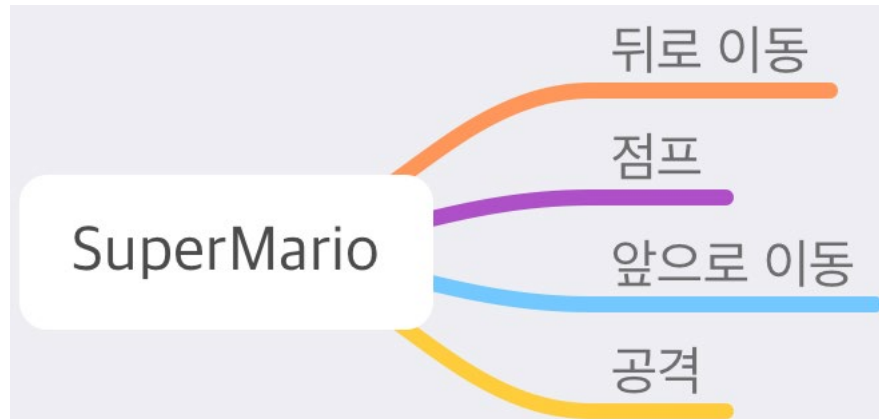
def jump(data):
    data[1] = 20

jump(mario)
print(mario)    # [0, 20]
```



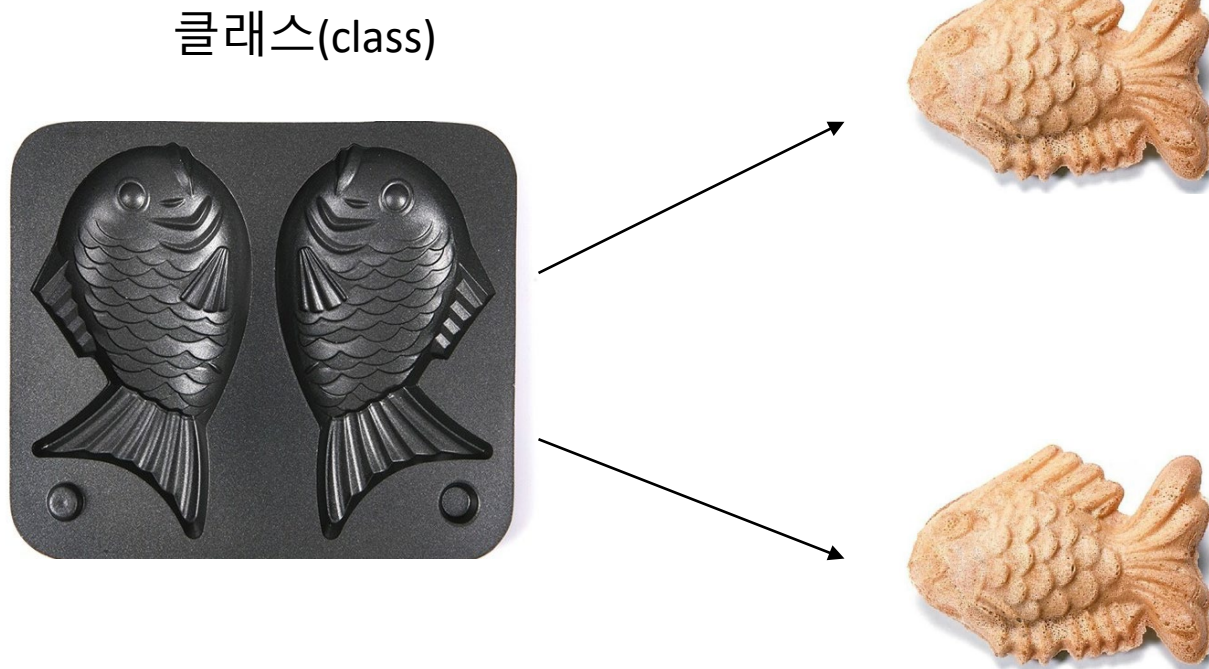
객체 지향 프로그래밍

- 객체 지향 프로그래밍 (Object Oriented Programming, OOP)
 - 1960년도에 등장한 프로그래밍 패러다임 (Simula67, smalltalk)
 - 프로그램을 **객체(object)**라는 기본 단위를 나누고 이들의 상호 작용을 서술하는 방식
 - 코드 확장 및 재사용이 용이함



용어 정리

- 정리된 클래스는 설계도
- 설계도에 의해 생성된 객체, 오브젝트, 인스턴스



객체
오브젝트(object)
인스턴스(instance)

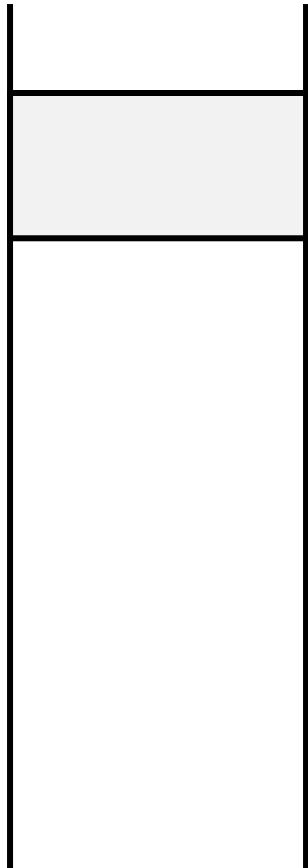
파이썬 클래스

- 파이썬 클래스 정의하기

```
class 붕어빵틀:  
    pass
```

붕어빵틀 →

클래스가 정의되면 메모리에
공간이 할당되고 이를 클래스
이름이 바인딩



클래스로부터 인스턴스 생성

- 클래스가 정의됐다면 인스턴스 생성 가능
 - 예) 붕어빵 틀이 있다면 이를 통해 붕어빵 굽기 가능
 - 예) 자동차를 만드는 설계도가 있다면 이를 통해 자동차 생산 가능
- 인스턴스 생성은 클래스 이름을 적고 '()'를 적어주면 됨
 - 생성된 인스턴스를 변수로 바인딩하기

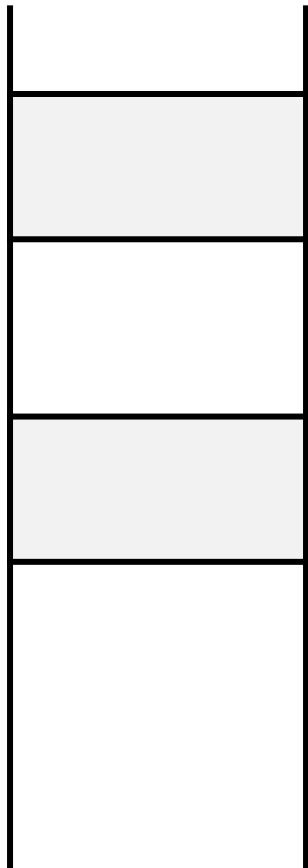
```
class 붕어빵틀:  
    pass
```

```
내빵 = 붕어빵틀()
```

붕어빵틀 →

내빵 →

인스턴스가 생성되면
공간이 할당됨

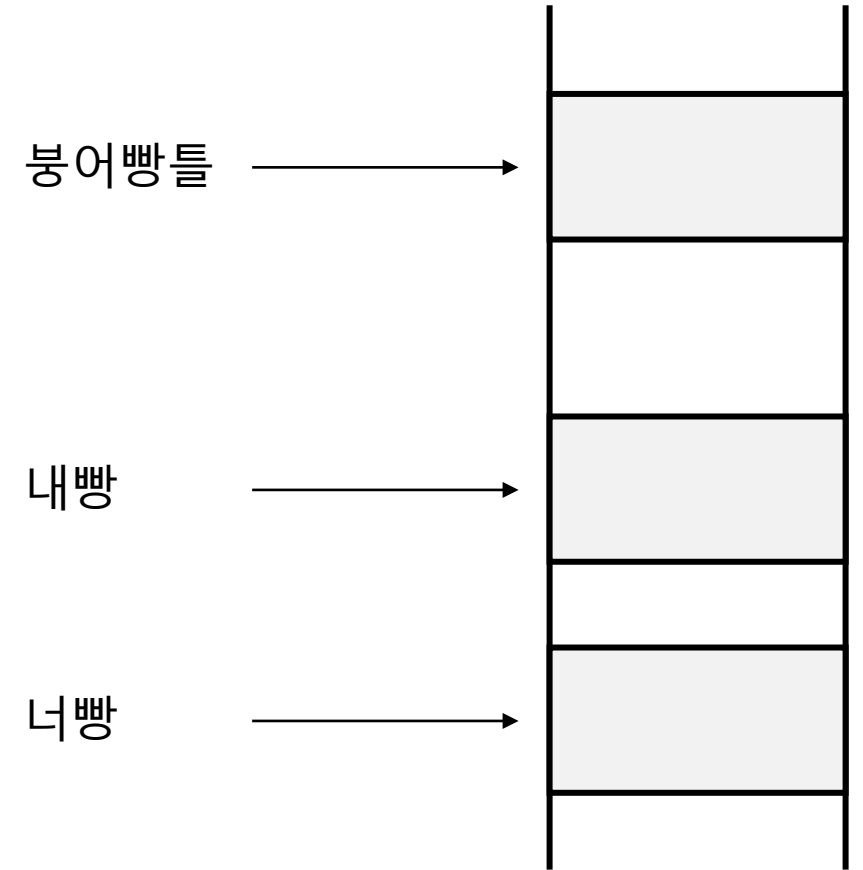


여러 인스턴스 생성

- 클래스로부터 여러 인스턴스 생성

```
class 붕어빵틀:  
    pass
```

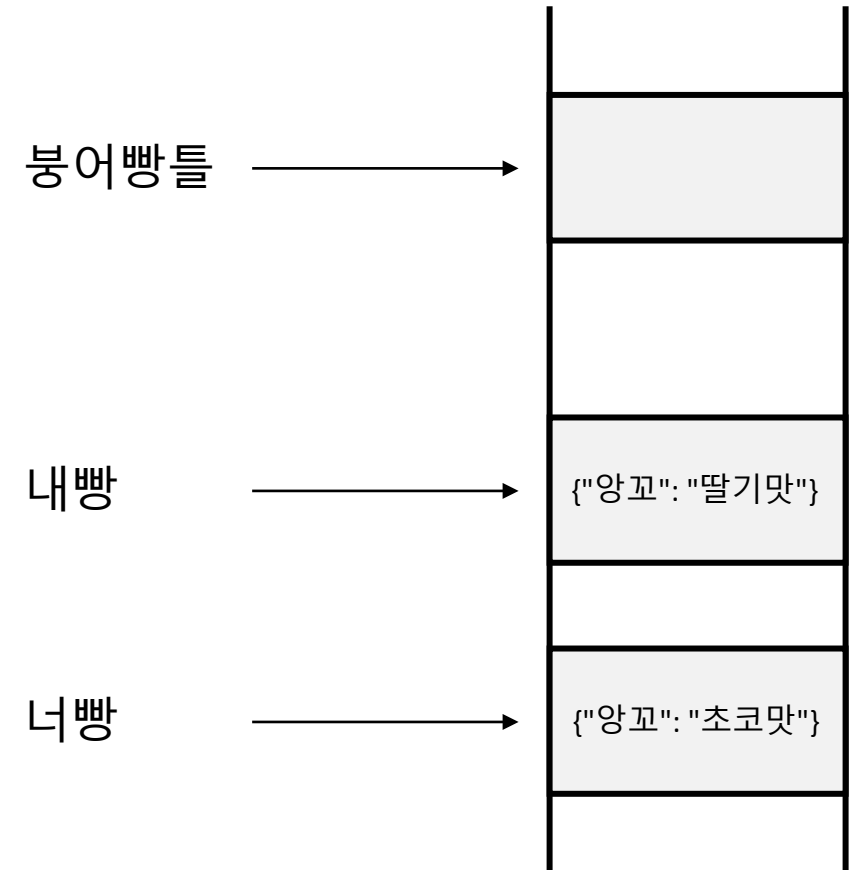
```
내빵 = 붕어빵틀()  
너빵 = 붕어빵틀()
```



인스턴스에 데이터 넣기

- 각 객체는 고유의 데이터를 저장하는 이름 공간을 가짐
 - 객체에 점(.)을 찍으면 객체 공간에 접근을 의미함
 - 객체 공간에 있는 변수를 **속성(property)**라고 부름

```
class 붕어빵틀:  
    pass  
  
내빵 = 붕어빵틀()  
너빵 = 붕어빵틀()  
  
내빵.앙꼬 = "딸기맛"  
너빵.앙꼬 = "초코맛"
```

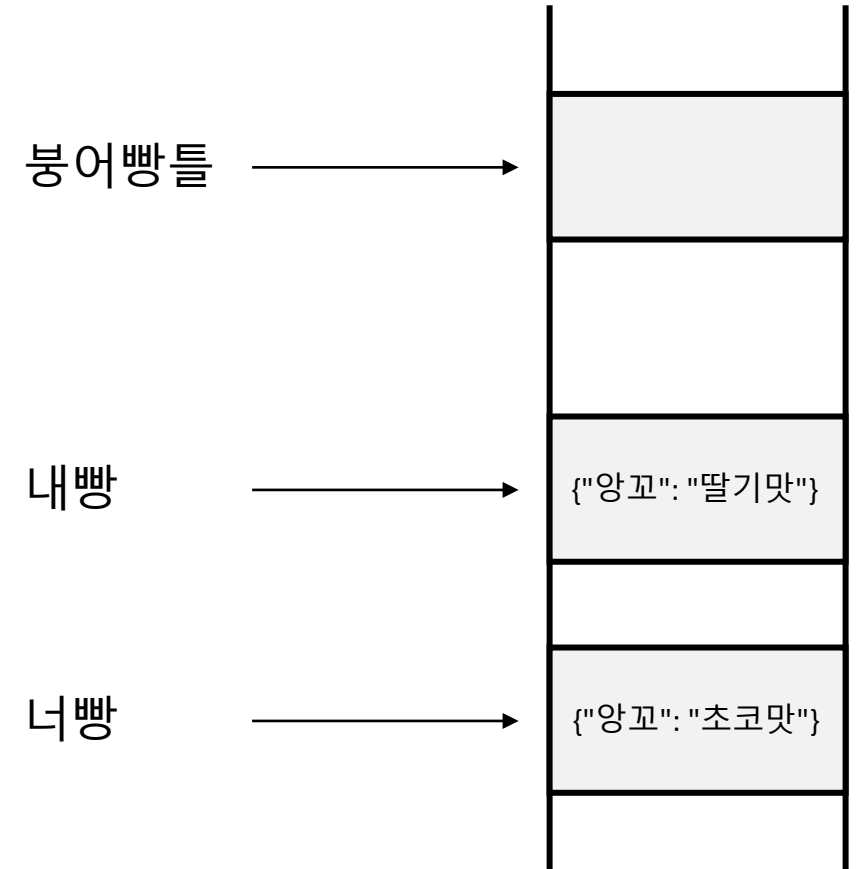


객체 공간의 값에 접근하기

- 객체 공간의 이름 공간에 저장된 값 접근하기

```
class 붕어빵틀:  
    pass  
  
내빵 = 붕어빵틀()  
너빵 = 붕어빵틀()  
  
내빵.앙꼬 = "딸기맛"  
너빵.앙꼬 = "초코맛"
```

```
print(내빵.앙꼬)  
print(너빵.앙꼬)
```



연습문제

1. 삼성차라는 이름의 클래스를 정의하고 두 개의 객체를 생성합니다. 첫 번째 객체에 차종이라는 속성에 "SM5"을 두 번째 객체의 차종이라는 속성에 "QM6"를 저장하세요. 생성한 각 객체에 “차종”이라는 속성의 값을 화면에 출력하세요.

연습문제

2. 계좌라는 이름의 클래스를 정의하세요. 계좌 클래스로부터 두 개의 객체를 생성하세요. 다음 표를 참조하여 각 객체에 속성을 추가하세요. 두 객체의 '잔고' 속성의 합을 화면에 출력하세요.

속성	값
이름	김철수
잔고	500

속성	값
이름	이영희
잔고	1000

연습 문제 더 풀어보기

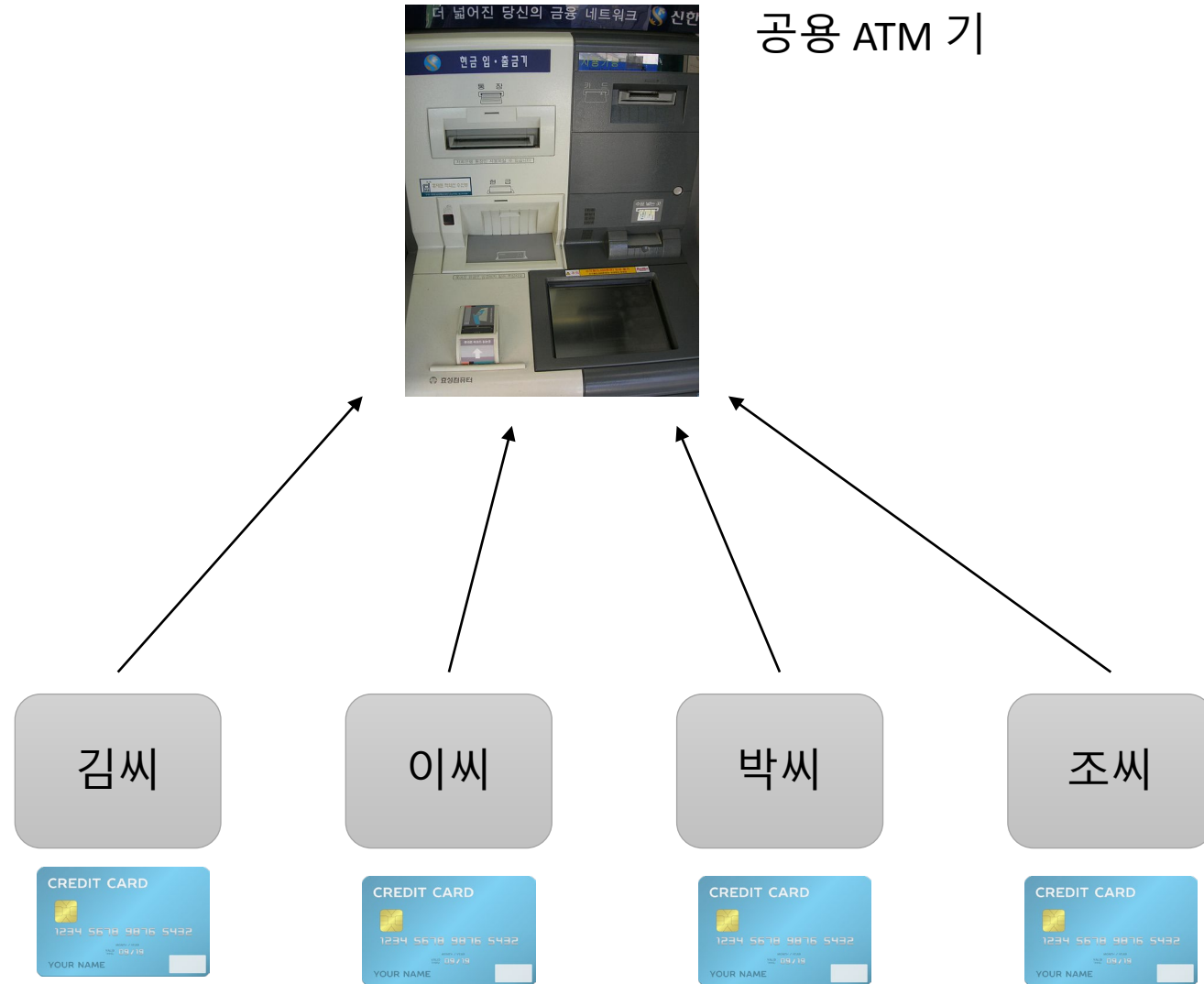
- <https://wikidocs.net/41106>

클래스와 메서드

Python을 활용한 자료구조 이해하기

클래스와 메서드

- 클래스의 정의된 함수를 메서드라고 부름
- 메서드의 클래스 공간에 저장됨
 - 메서드는 여러 객체에 의해 공유됨

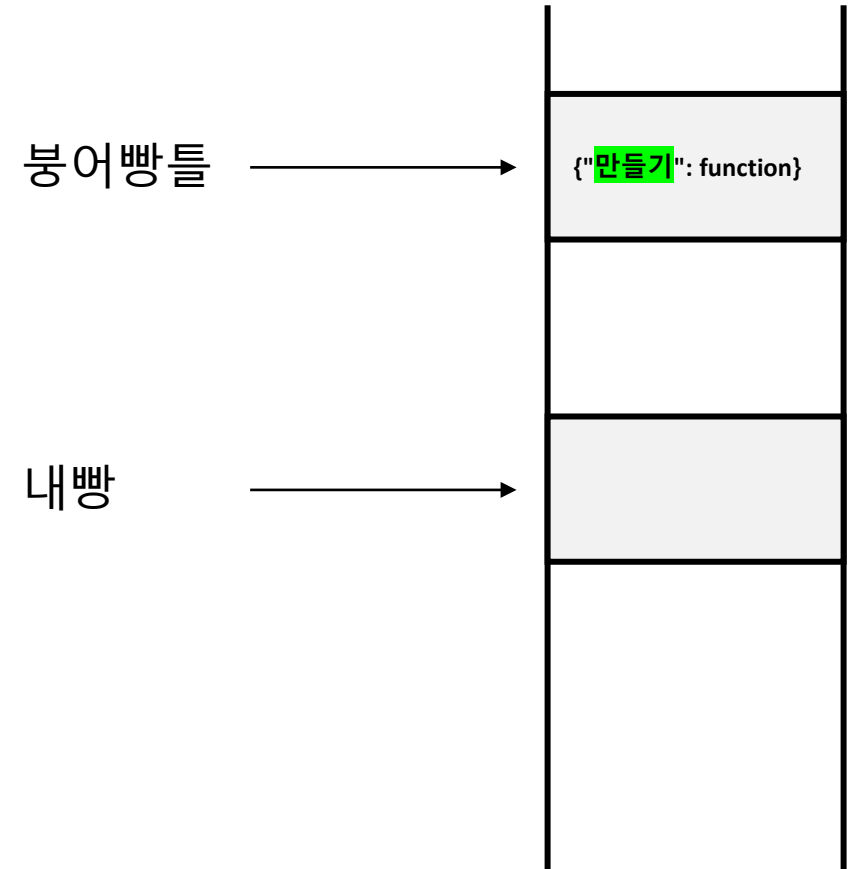


클래스에 메서드 추가 (1/2)

- 클래스의 메서드 정의
 - 클래스에 들여쓰기 후 메서드 정의
 - 첫 번째 인자는 어떤 붕어빵인지를 바인딩하는 변수
 - 메소드는 클래스 공간의 이름 공간에 저장

```
class 붕어빵틀:  
    def 만들기(어떤빵):  
        print("붕어빵 구워짐")
```

```
내빵 = 붕어빵틀()
```

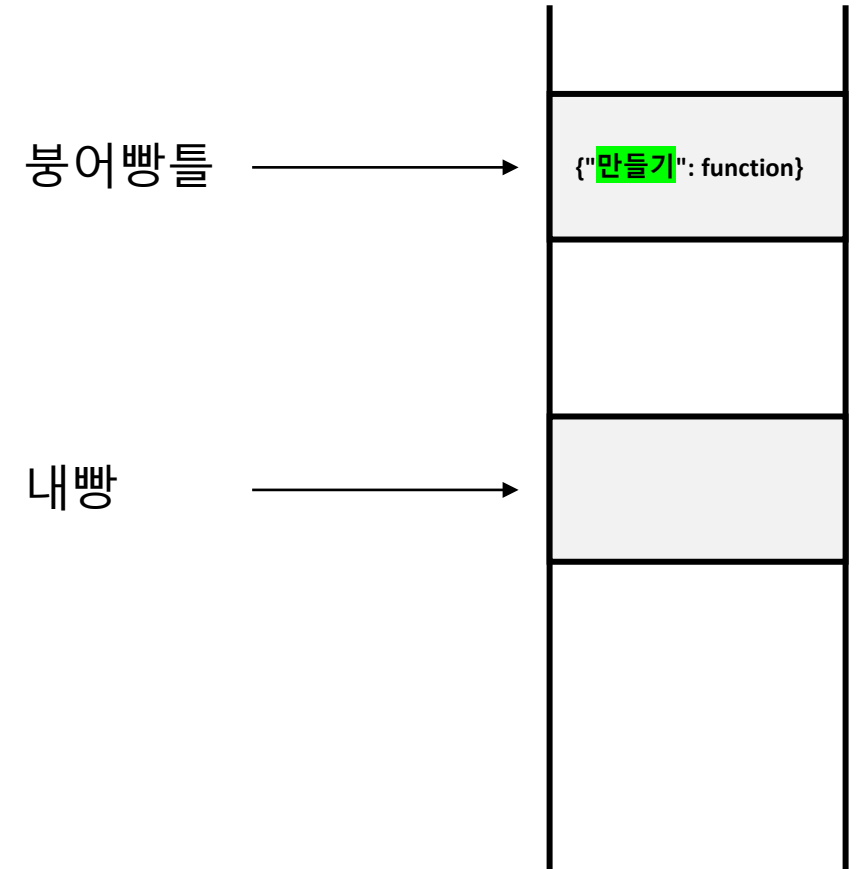


클래스에 메서드 추가 (2/2)

- 메서드 호출하기
 - 클래스 이름에 점을 찍으면 클래스 공간에 접근을 의미
 - 메서드 호출 시 생성한 객체를 전달

```
class 붕어빵틀:  
    def 만들기(어떤빵):  
        print("붕어빵 구워짐")
```

```
내빵 = 붕어빵틀()  
붕어빵틀.만들기(내빵)
```



파라미터 있는 메서드

- 앙꼬 넣기 메서드 추가
 - 첫 번째 인자는 어떤 붕어빵인지를 바인딩하는 변수
 - 두 번째 인자는 붕어빵에 넣을 앙꼬를 바인딩하는 변수

```
class 붕어빵틀:
```

```
    def 앙꼬넣기(어떤빵, 앙꼬):  
        어떤빵.앙꼬 = 앙꼬
```

```
내빵 = 붕어빵틀()
```

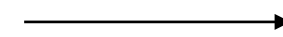
```
붕어빵틀.앙꼬넣기(내빵, "딸기맛")
```

붕어빵틀



{"앙꼬넣기": function}

내빵



{"앙꼬": "딸기맛"}

"딸기맛"

연습문제

계좌 클래스를 정의하세요. 계좌 클래스에 개설이라는 메서드를 정의하세요. 개설 메서드는 다음과 같이 총 3개의 인자를 입력 받습니다. 개설 메서드는 객체 공간에 이름과 잔고라는 속성을 생성하고 사용자로부터 입력 받은 이름과 잔고를 저장합니다.

```
def 개설(누구계좌, 이름, 잔고):  
    pass
```

연습문제

계좌 클래스에 출력이라는 메서드를 추가하세요. 출력이라는 메서드는 다음과 같이 정의되며 메서드가 호출되면 어떤 계좌 객체의 이름 공간에 저장된 “이름”과 “잔고” 속성을 출력합니다.

```
def 출력(누구계좌):  
    # 어떤 계좌의 이름과 잔고 속성 값을 출력
```

파이썬 클래스와 self

- 메서드의 첫 번째 인자의 변수 이름으로 **관례적으로 self를 사용하자.**

```
class 붕어빵틀:  
    def 앙꼬넣기(어떤빵, 앙꼬):  
        어떤빵.앙꼬 = 앙꼬
```



```
class 붕어빵틀:  
    def 앙꼬넣기(self, 앙꼬):  
        self.앙꼬 = 앙꼬
```

```
class 계좌:  
    def 개설(어떤계좌, 이름, 잔고):  
        어떤계좌.이름 = 이름  
        어떤계좌.잔고 = 잔고
```



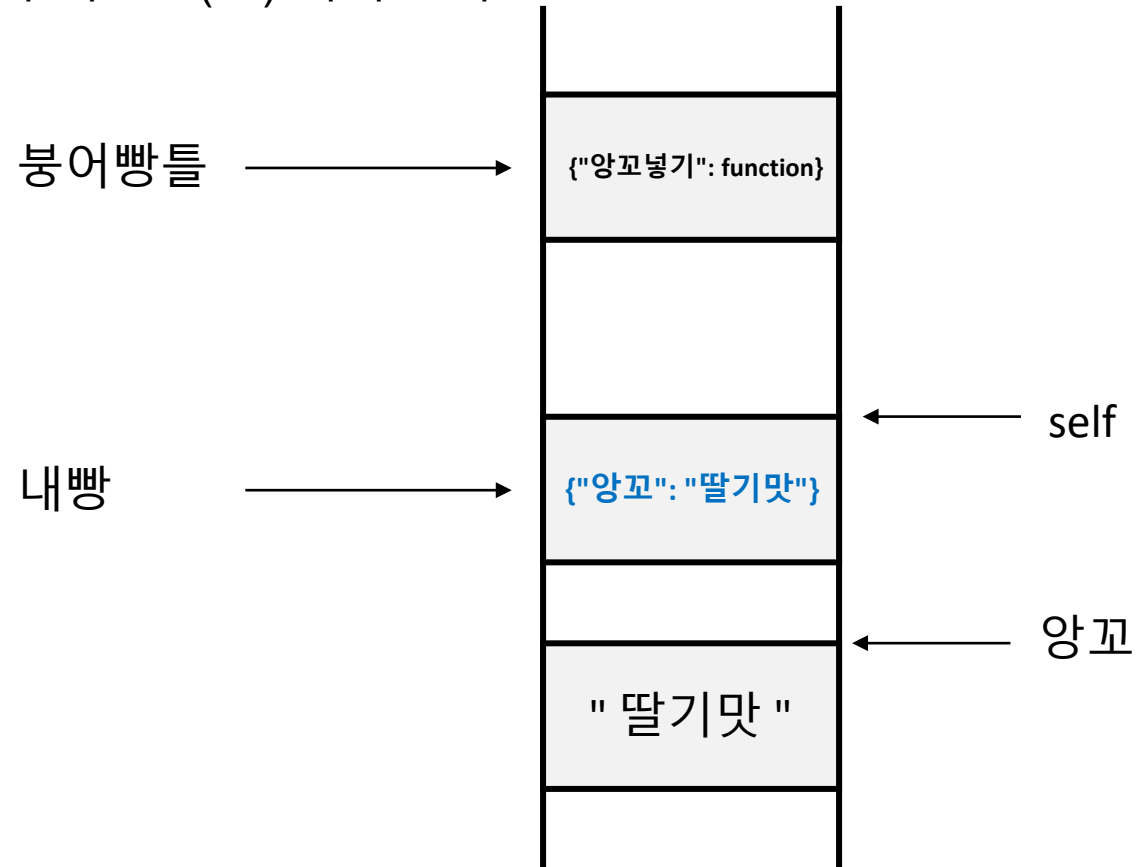
```
class 계좌:  
    def 개설(self, 이름, 잔고):  
        self.이름 = 이름  
        self.잔고 = 잔고
```

메서드 호출 방식

- 메서드는 클래스 공간에 저장되고 데이터는 각 객체 공간에 저장된다
 - 클래스 공간에 저장된 메서드는 여러 객체에 의해 호출됨
 - 메서드가 호출될 때 첫 번째 인자인 `self`는 데이터가 저장된(할) 객체를 바인딩함

```
class 붕어빵틀:  
    def 앙꼬넣기(self, 앙꼬):  
        self.앙꼬 = 앙꼬
```

```
내빵 = 붕어빵틀()  
붕어빵틀.앙꼬넣기(내빵, "딸기맛")
```



새로운 메서드 호출 방식

- 객체.메서드(인자1, 인자2, ...)
 - 객체.메서드(...) 호출 방식은 파이썬 인터프리터에 의해 자동으로 클래스.메서드(객체, ...)으로 변경됨
 - 객체.메서드(...) 호출 방식은 **self 자리에 값을 전달할 필요가 없음**
 - 파이썬 인터프리터에 의해 자동으로 전달되기 때문

```
class 붕어빵틀:  
    def 앙꼬넣기(self, 앙꼬):  
        self.앙꼬 = 앙꼬
```

```
내빵 = 붕어빵틀()
```

```
#붕어빵틀.앙꼬넣기(내빵, "딸기맛")  
내빵.앙꼬넣기("딸기맛")
```

클래스 생성자

Python을 활용한 자료구조 이해하기

붕어빵을 굽는 방식

- 붕어빵 틀로부터 먼저 붕어빵을 굽는다.
- 구어진 붕어빵에 앙꼬를 넣는다. (메서드 호출)

```
class 붕어빵틀:  
    def 앙꼬넣기(self, 앙꼬):  
        self.앙꼬 = 앙꼬
```

```
내빵 = 붕어빵틀()
```

```
내빵.앙꼬넣기("딸기맛")
```

생성자

- 클래스로부터 객체가 생성될 때 파이썬 인터프리터에 의해 자동으로 호출되는 특별한 메서드
 - 메모리에 생성된 객체 공간에 데이터를 넣거나 초기화하는 목적으로 사용됨
 - `__init__` 이라는 특별한 이름을 가짐
 - 첫 번째 인자로 `self`를 사용함

```
class 붕어빵틀:  
    def __init__(self):  # 어떤 객체를 초기화 할 것인가?  
        print("붕어빵 잘 구어짐")
```

객체 생성하기

- 객체를 생성하면 자동으로 생성자가 호출됨
 - 사용자가 호출 하는 것이 아님
 - 파이썬 인터프리터가 객체를 생성한 후 호출하는 것임

```
class 붕어빵틀:  
    def __init__(self):  
        print("붕어빵 잘 구어짐")
```

```
내빵 = 붕어빵틀()
```

붕어빵틀 클래스 업데이트 (1/2)

- 기존에 앙꼬넣기의 역할은 붕어빵이 구어질 때 앙꼬를 넣는 것임
 - 사용자가 객체를 생성한 후 명시적으로 호출했었음 → 생성자로 이름을 변경하면 알아서 호출됨
 - 객체가 생성되면 자동으로 생성자가 호출되는데 생성자는 앙꼬라는 인자가 필요함. 따라서 객체를 생성할 때 구울 붕어빵에 넣을 앙꼬를 같이 전달해야함

```
class 붕어빵틀:  
    def 앙꼬넣기(self, 앙꼬):  
        self.앙꼬 = 앙꼬
```

```
내빵 = 붕어빵틀()  
내빵.앙꼬넣기("딸기맛")
```

```
class 붕어빵틀:  
    def __init__(self, 앙꼬):  
        self.앙꼬 = 앙꼬
```

```
내빵 = 붕어빵틀()
```

TypeError: __init__() missing 1 required positional argument: '앙꼬'

붕어빵틀 클래스 업데이트 (2/2)

- 기존에 앙꼬넣기의 역할은 붕어빵이 구어질 때 앙꼬를 넣는 것임
 - 사용자가 객체를 생성한 후 명시적으로 호출했었음 → 생성자로 이름을 변경하면 알아서 호출됨
 - 객체가 생성되면 자동으로 생성자가 호출되는데 생성자는 앙꼬라는 인자가 필요함. 따라서 객체를 생성할 때 구울 붕어빵에 넣을 앙꼬를 같이 전달해야함

```
class 붕어빵틀:  
    def 앙꼬넣기(self, 앙꼬):  
        self.앙꼬 = 앙꼬
```

```
내빵 = 붕어빵틀()  
내빵.앙꼬넣기("딸기맛")
```

```
class 붕어빵틀:  
    def __init__(self, 앙꼬):  
        self.앙꼬 = 앙꼬
```

```
내빵 = 붕어빵틀("딸기맛")  
print(내빵.앙꼬)
```

연습문제

1. 사람 클래스를 정의하세요.

- 생성자를 통해 (이름, 생년월일, 성별) 입력
- 정보출력 메서드에서 정보를 출력

```
>> 첫째 = 사람("유종훈", "19860302", "남")  
>> 첫째.정보출력()  
1986년 3월 2일 출생  
(남) 유종훈
```


연습문제

- 2. 다음과 같이 사용할 수 있는 비행기 클래스를 정의해보세요.

```
>> 비행기_1 = 비행기("보잉787")  
>> 비행기_1.이륙()  
보잉787 이륙합니다
```

```
>> 비행기_2 = 비행기("에어버스A330")  
>> 비행기_2.이륙()  
에어버스A330 이륙합니다
```

연습문제

3. 다음과 같이 사용할 수 있는 계좌 클래스를 정의해보세요.

- 생성자에서 이름과 잔고를 입력
- 출력 메서드에서 이름과 잔고를 출력

기타 특수 메서드

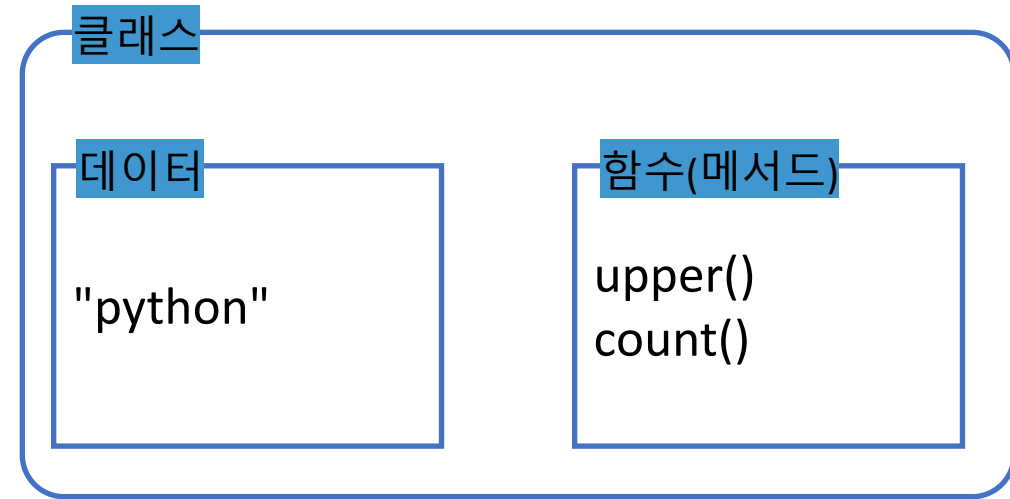
- underline ('__')은 파이썬에서 사용하는 특수기능

연산자	메서드	설명
+	__add__	덧셈
*	__mul__	곱셈
-	__sub__	뺄셈
/	__truediv__	나눗셈
%	__mod__	나머지
<	__lt__	크다
==	__eq__	같다
len	__len__	길이
str	__str__	문자열 변환

파이썬의 모든 것은 클래스

```
b = str("python")  
print(type(b))  
print(b.upper())  
print(b.count('p'))
```

```
<class 'str'>  
PYTHON  
1
```



클래스 실습

Python을 활용한 자료구조 이해하기

계좌 클래스

- 잔고를 관리하는 클래스
 - 입금하기
 - 출금하기
 - 이체하기
 - 통장정리

입출금 기능 추가

- 입출금 금액에 따라 잔고 변화

```
a = 계좌 ("철수", 1000)  
a.입금하기(100)  
print(a.잔고)
```

1100

```
a.입금하기(100)  
a.출금하기(300)  
print(a.잔고)
```

800

통장 정리

- 현재 잔고, 과거 입출금 내역, 잔고 변동 사항을 출력하라.

```
-----  
통 장 정 리  
-----  
어피치 님 환영합니다  
잔고: 1300 원  
-----  
입금: 300 원    잔고: 1300 원  
입금: 300 원    잔고: 1600 원  
출금: 300 원    잔고: 1300 원
```

※ 입금과 출금의 히스토리를 저장해야 함

이체하기

- 두 객체 간의 잔고 변경

```
a = 계좌 ("철수", 1000)  
a.입금하기(100)
```

```
b = 계좌 ("영희", 1000)  
print(b.잔고)
```

```
a.이체하기(b, 100)
```

```
print(a.잔고)  
print(b.잔고)
```

```
1000  
1100
```

클래스 상속

Python을 활용한 자료구조 이해하기

클래스 상속

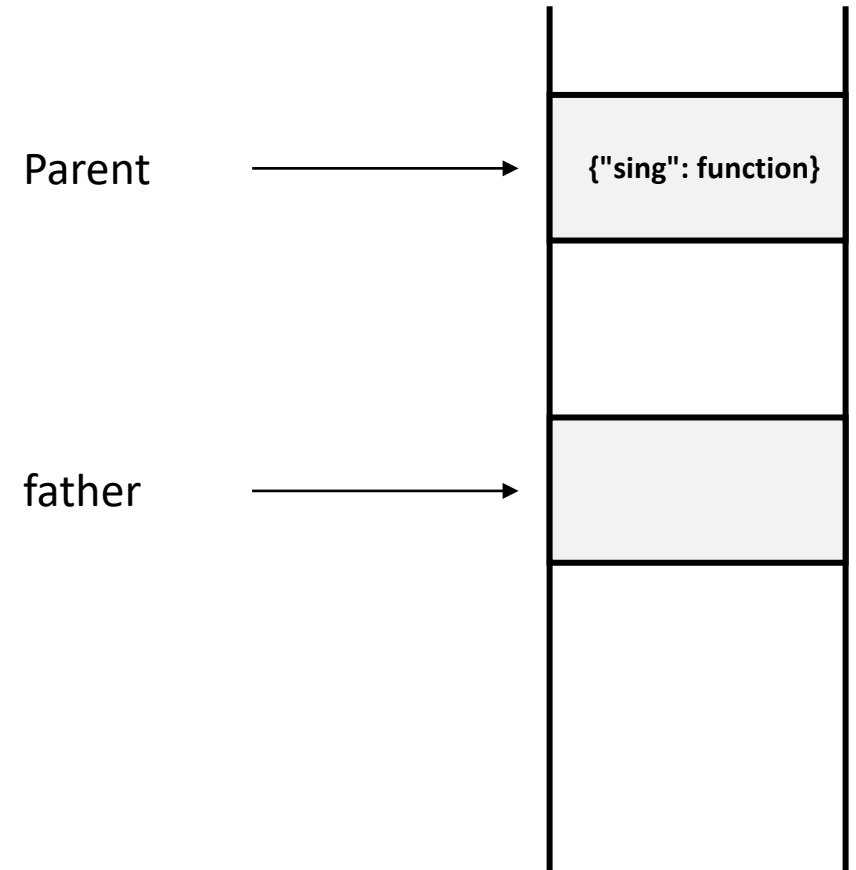
- 상속 (inheritance)는 기존 클래스의 작동 방식을 변경함으로써 새로운 클래스를 만드는 메커니즘
- 자식이 부모님으로부터 재산 등을 상속받는 것처럼 다른 클래스에 구현된 메서드나 속성값들을 상속받는 클래스에서 그대로 사용할 수 있음

노래 잘 부르는 부모 클래스

- Parent 클래스는 sing 메서드를 가짐

```
class Parent:  
    def sing(self):  
        print("sing a song")
```

```
father = Parent()  
father.sing()
```



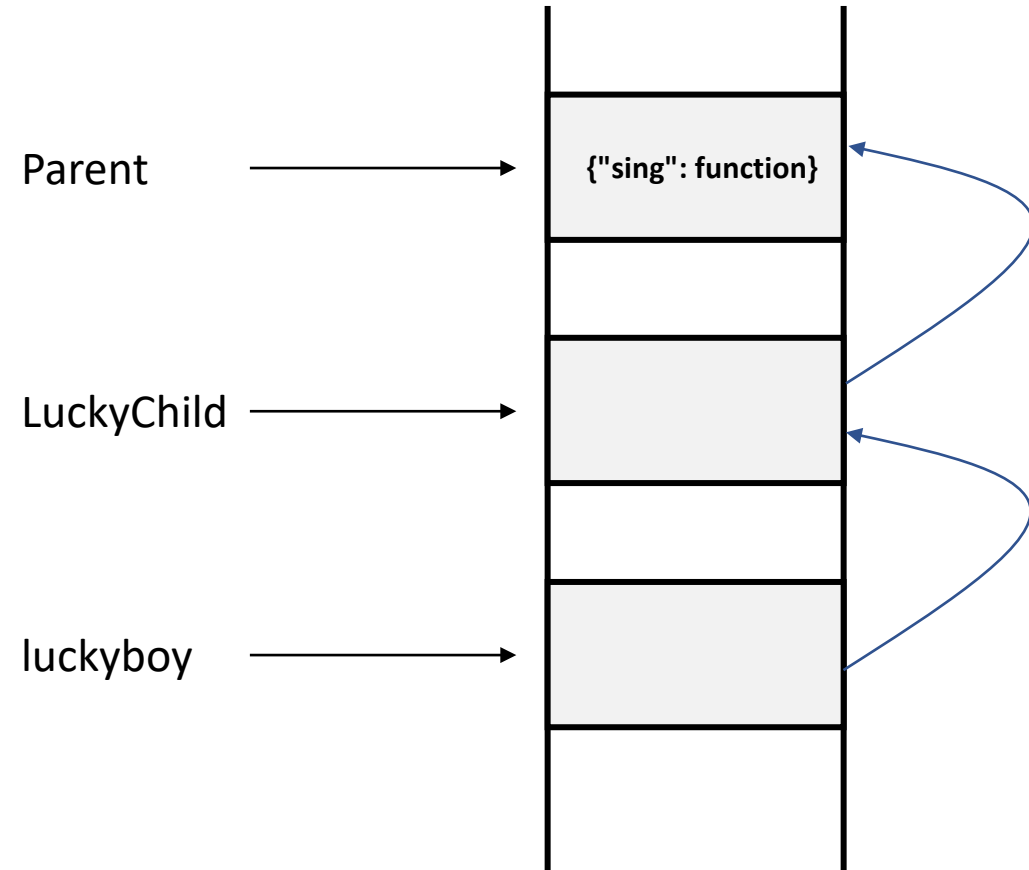
운이 좋은 자식 클래스

- 부모로부터 능력을 물려 받음

```
class Parent:
    def sing(self):
        print("sing a song")

class LuckyChild(Parent):
    pass

luckyboy = LuckyChild()
luckyboy.sing()
```



운이 없는 자식 클래스

- 부모로부터 능력을 물려 받지 못한 자식 클래스

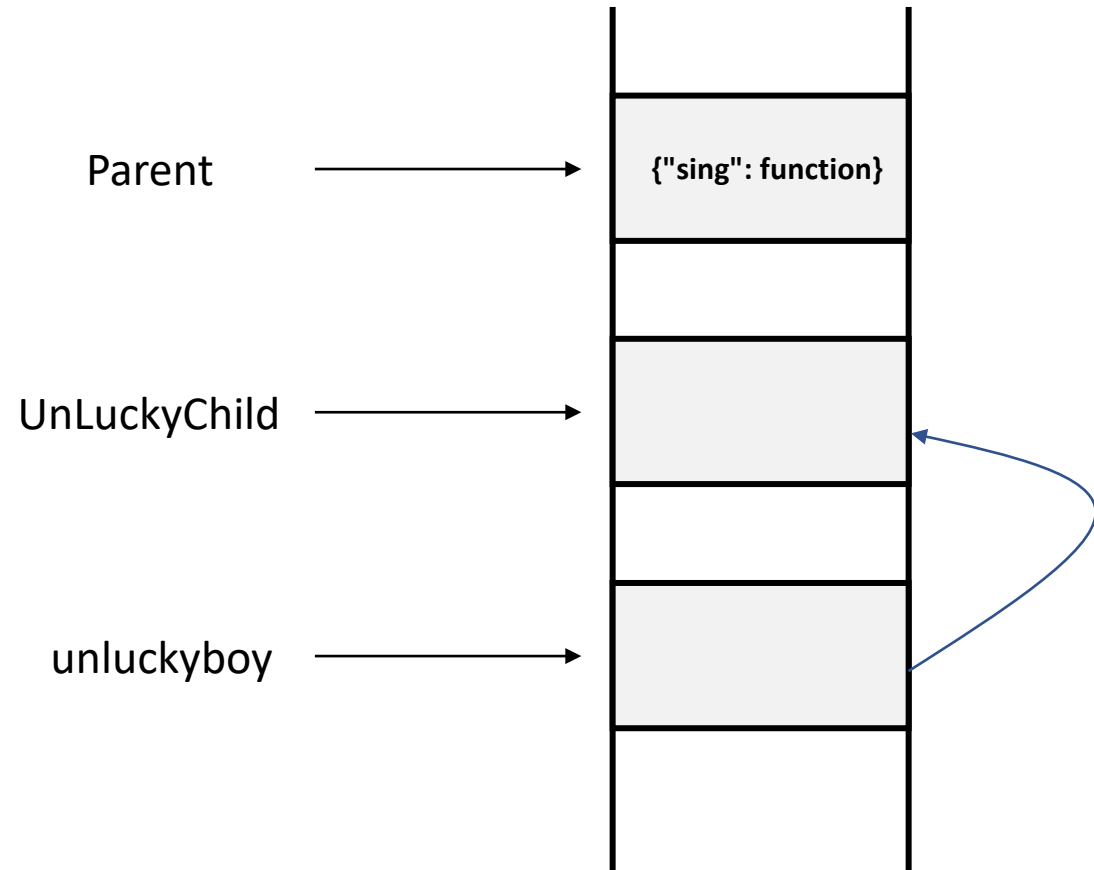
```
class Parent:
    def sing(self):
        print("sing a song")

class UnLuckyChild:
    pass

unluckyboy = UnLuckyChild()
unluckyboy.sing()
```

실행결과

```
unluckyboy.sing()
AttributeError: 'UnLuckyChild' object has no attribute 'sing'
```



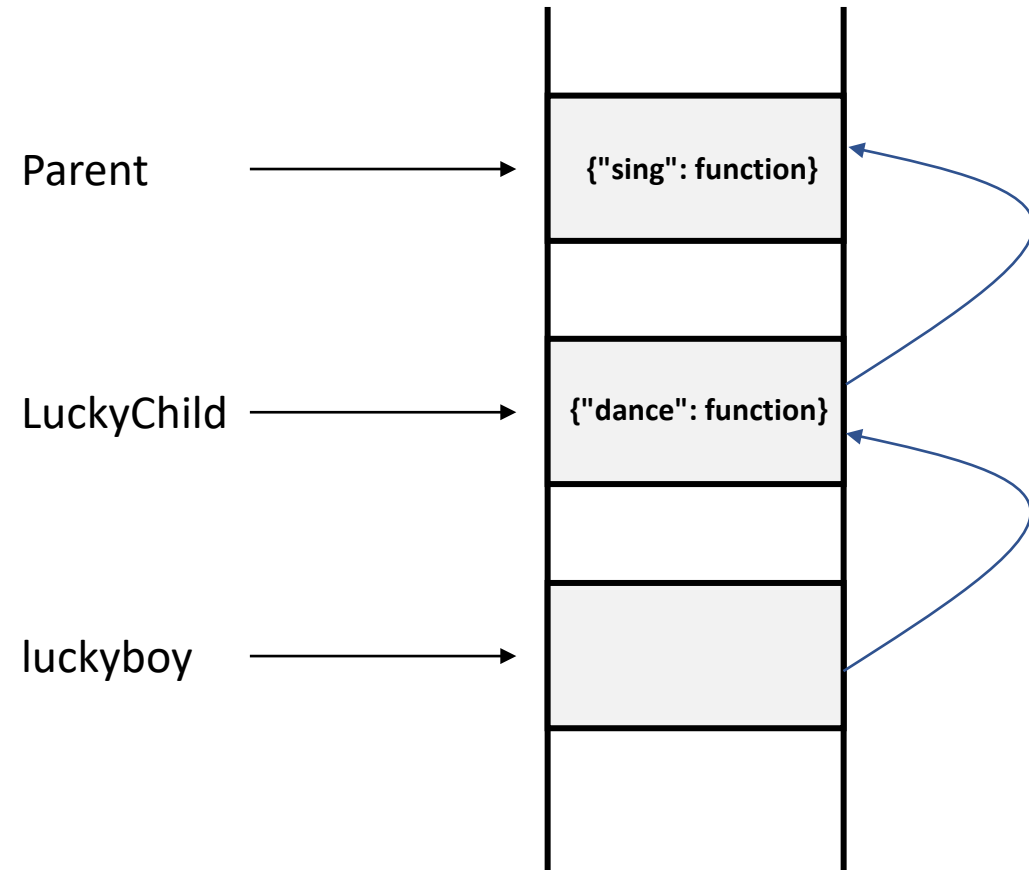
바람직한 상속 케이스

- 노래 능력은 부모로부터 물려받고 춤은 노력해서...

```
class Parent:
    def sing(self):
        print("sing a song")

class LuckyChild(Parent):
    def dance(self):
        print("shuffle dance")

luckyboy = LuckyChild()
luckyboy.sing()
luckyboy.dance()
```



인스턴스 속성 참조 (1/3)

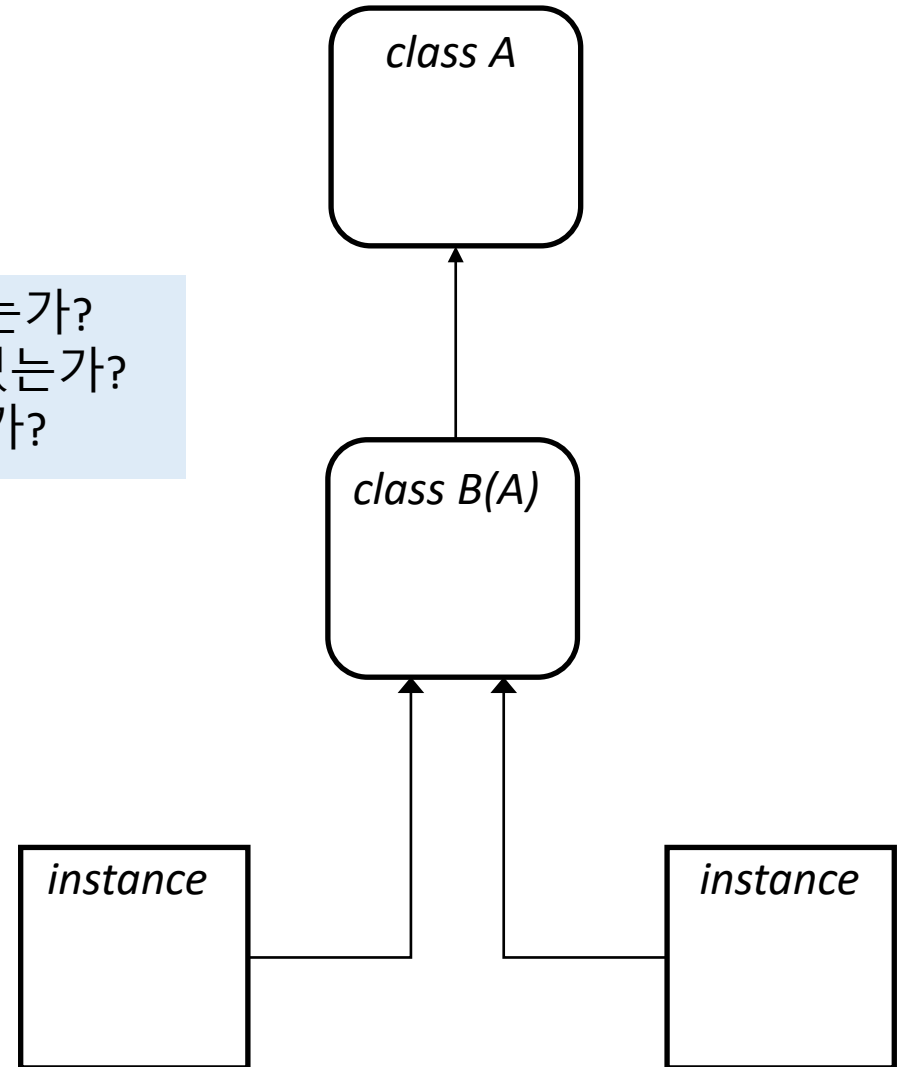
- 속성 참조 (property/method)
 - object.attribute

```
class Parent:
    def sing(self):
        print("sing a song")

class LuckyChild(Parent):
    def dance(self):
        print("shuffle dance")

luckyboy = LuckyChild()
luckyboy.sing()
luckyboy.dance()
```

- 1) instance에 sing()이 있는가?
- 2) LuckyChild에 sing()이 있는가?
- 3) Parent에 sing()이 있는가?



인스턴스 속성 참조 (2/3)

- 먼저 탐색된 속성/메서드를 우선 호출

```
class Parent:  
    def sing(self):  
        print("sing a song")
```

```
class LuckyChild(Parent):  
    def sing(self):  
        print("song?")
```

```
luckyboy = LuckyChild()  
luckyboy.sing()  
luckyboy.dance()
```

```
song?
```

인스턴스 속성 참조 (3/3)

- 명시적으로 부모 클래스를 호출하는 코드를 추가

```
class Parent:  
    def sing(self):  
        print("sing a song")
```

```
class LuckyChild(Parent):  
    def sing(self):  
        Parent.sing(self)  
        print("song?")
```

```
luckyboy = LuckyChild()  
luckyboy.sing()  
luckyboy.dance()
```

(O)

```
class Parent:  
    def sing(self):  
        print("sing a song")
```

```
class LuckyChild(Parent):  
    def sing(self):  
        self.sing(self)  
        print("song?")
```

```
luckyboy = LuckyChild()  
luckyboy.sing()  
luckyboy.dance()
```

(X)

```
class Parent:  
    def sing(self):  
        print("sing a song")
```

```
class LuckyChild(Parent):  
    def sing(self):  
        super().sing()  
        print("song?")
```

```
luckyboy = LuckyChild()  
luckyboy.sing()  
luckyboy.dance()
```

(O)

스타크래프트로 보는 상속 (1/2)

- 비슷한 보병 유닛은 비슷한 속성을 갖고 있음

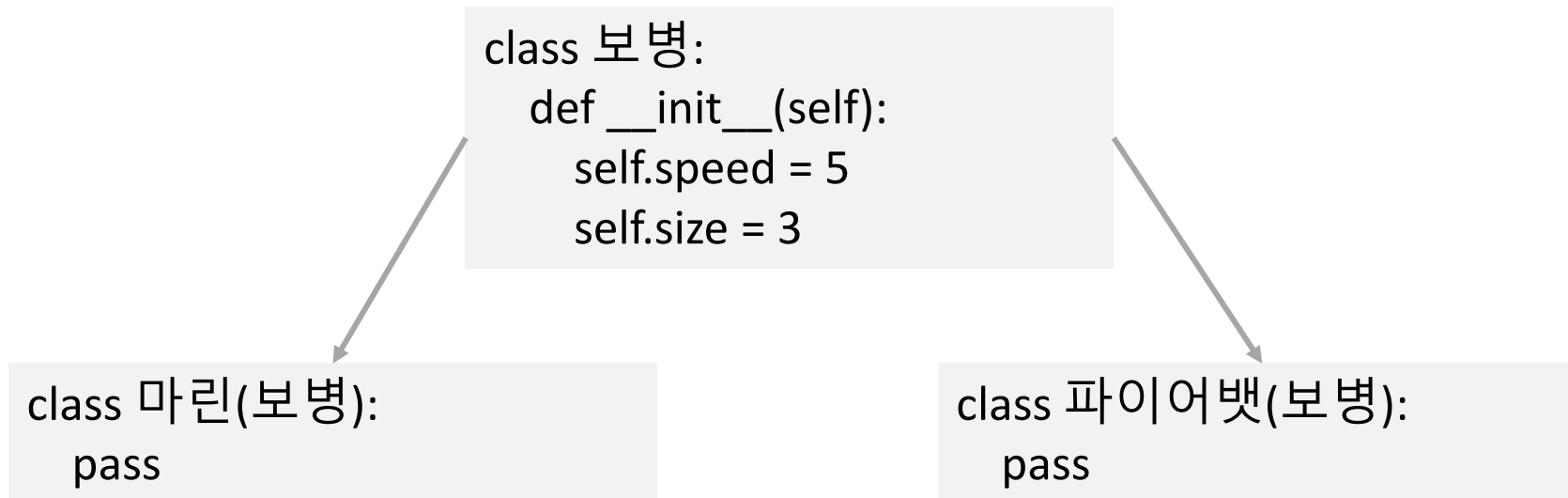
```
class 마린:  
    def __init__(self):  
        self.speed = 5  
        self.size = 3
```

```
class 파이어뱃:  
    def __init__(self):  
        self.speed = 5  
        self.size = 3
```

보병 유닛을 강화하고 싶어서 speed를 늘리고 싶다면?
-> 보병 유닛 클래스 숫자 만큼의 수정이 필요함

스타크래프트로 보는 상속 (2/2)

- 공통된 속성을 부모 클래스로 정의



스택래프로 보는 참조 순서 (1/2)

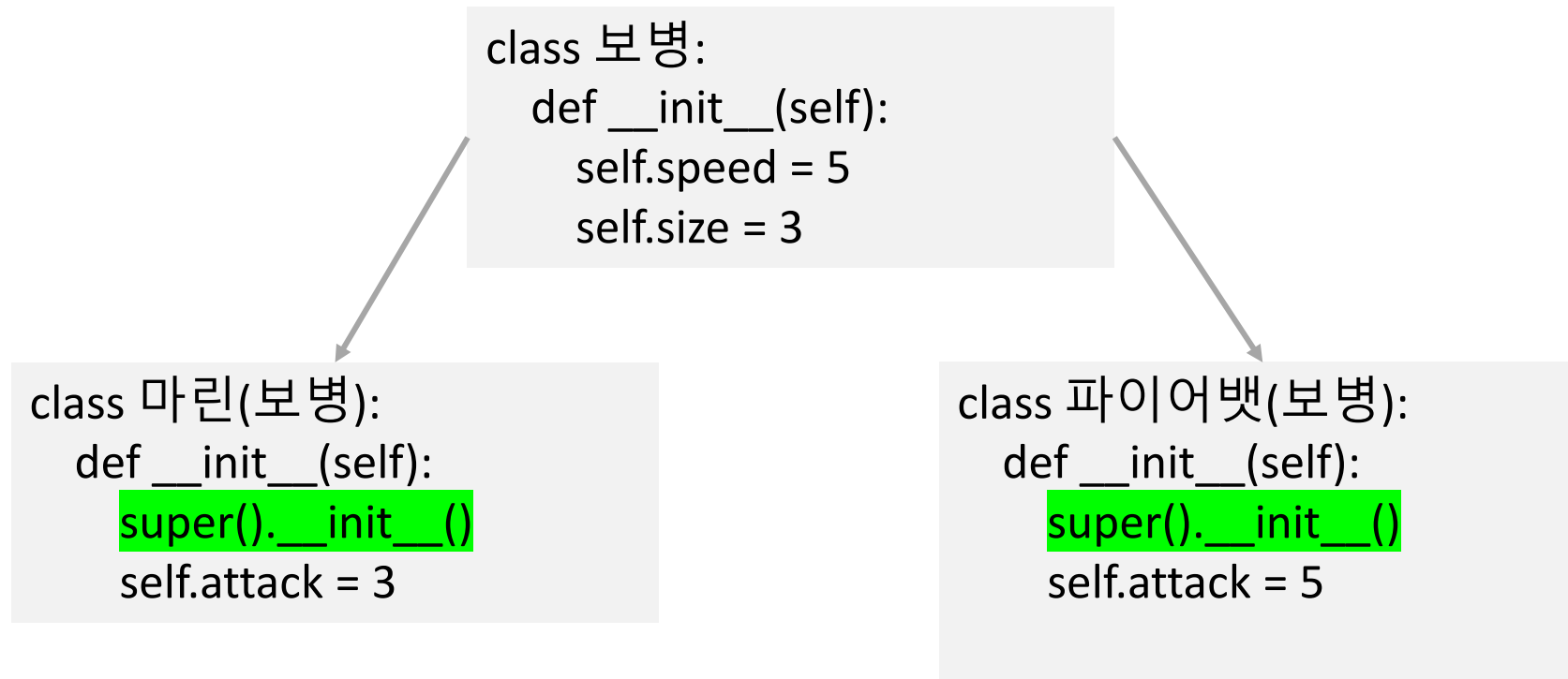
- 마린과 파이어뱃은 차이점이 존재함
 - 공통된 특성과 차이점을 어떻게 표현할 수 있을까?

```
class 마린:  
    def __init__(self):  
        self.attack = 3  
        self.speed = 5  
        self.size = 3
```

```
class 파이어뱃:  
    def __init__(self):  
        self.attack = 5  
        self.speed = 5  
        self.size = 3
```

스택크래프로 보는 참조 순서 (2/2)

- 부모 클래스를 호출하고 차이점을 추가 기술



파이썬 문자열

Python을 활용한 자료구조 이해하기

문자열의 중요성

- 많은 데이터가 문자열 형태로 제공

주요재무정보	주요재무정보							
	2011/12	2012/12	2013/12	2014/12	2015/12	2016/12(E)	2017/12(E)	2018/12(E)
	(IFRS연결)	(IFRS연결)	(IFRS연결)	(IFRS연결)	(IFRS연결)	(IFRS연결)	(IFRS연결)	(IFRS연결)
매출액	4,213	4,534	2,108	4,989	9,322	14,124	17,370	18,553
영업이익	1,168	1,018	659	1,764	886	1,303	2,268	2,840
세전계속사업이익	1,274	1,009	604	1,688	1,095	1,200	2,216	2,849
당기순이익	1,080	766	614	1,498	788	834	1,575	2,014
당기순이익(지배)	1,085	763	614	1,501	757	732	1,392	1,776
당기순이익(비지배)	-5	4	0	-3	31			
자산총계	5,587	6,119	2,172	27,680	31,885	50,601	52,591	54,630
부채총계	1,025	919	399	3,048	6,030	15,405	15,898	16,118
자본총계	4,562	5,199	1,773	24,632	25,855	34,439	36,619	38,470
자본총계(지배)	4,565	5,153	1,773	24,546	25,524	33,167	35,343	36,836
자본총계(비지배)	-3	46	0	86	331			
자본금	68	68	210	291	301	338	342	342

"18,533"

출처 : 네이버 금융

변수와 문자열

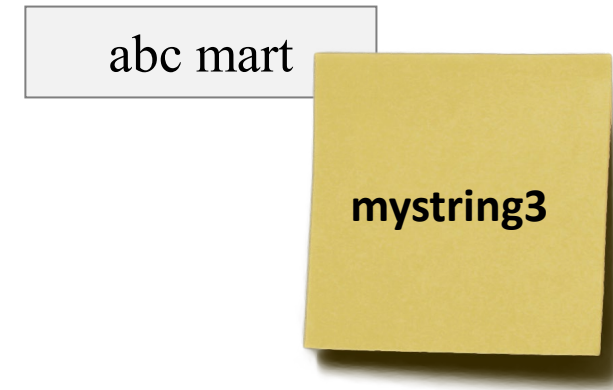
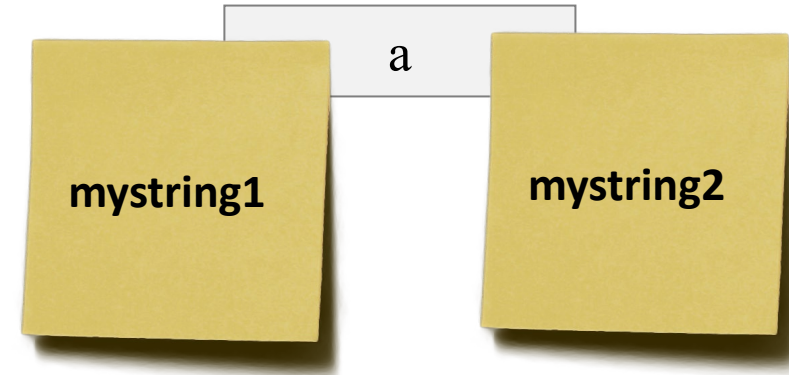
- 문자열도 변수에 바인딩할 수 있음

```
In [0]: mystring1 = 'a'
```

```
In [1]: mystring2 = "a"
```

```
In [2]: mystring3 = "abc mart"
```

같은 문자열은 같은 메모리에 저장한다.

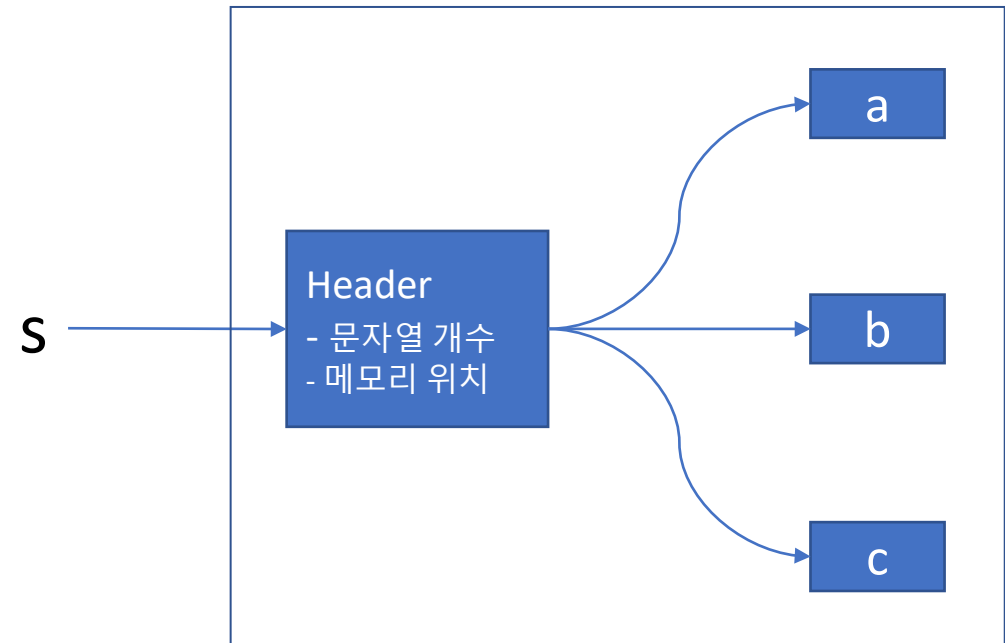


문자열 저장 구조

- 메모리 상에 저장되는 구조

```
s = "abc"  
print(id(s))  
print(id(s[0]))  
print(id(s[1]))  
print(id(s[2]))
```

```
2415747705776  
2415747890736  
2415747601520  
2415746564528
```



문자열과 메서드 (1/2)

- 문자열은 여러 개의 문자를 담아두는 데이터 타입
 - 클래스 = 데이터 타입 = 자료구조

```
변수 = "hi"  
변수 = str("hi")
```

문자열과 메서드 (2/2)

- 문자열 클래스에 정의된 다양한 메서드를 사용할 수 있음
 - `replace`는 문자열을 치환

```
변수.replace("h", "H")
```

```
class str:  
    def __init__(self, val):  
        self.data = val  
  
    def replace(self, asis, tobe):  
        pass
```

연습 문제

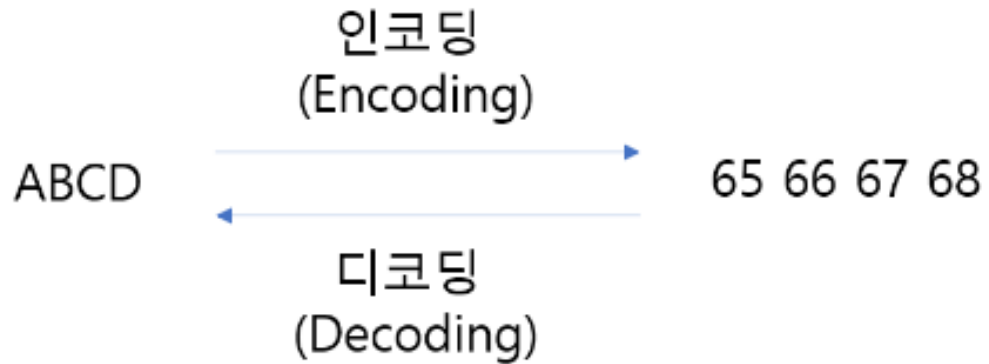
- replace 기능을 지원하는 MyStr 클래스를 정의하라.

```
변수 = MyStr("hi")  
변수.replace("h", "H")  
print(변수.data)
```

```
변수 = MyStr("hi")  
변수.replace("h", "H")  
result = 변수.replace("h", "H")  
print(result)
```

문자열 인코딩과 디코딩

- 인코딩: 컴퓨터가 내부적으로 데이터를 저장하는 방식
- 디코딩: 숫자를 사람이 읽을 수 있는 문자로 변경하는 과정



인코딩과 디코딩 예제

- 테이블을 참고하여 "가라" 라는 문자열을 숫자로 인코딩하라.
- 테이블을 참고하여 070808을 디코딩하라.

수	값
00	'가'
01	'나'
02	'다'
03	'라'
04	'마'
05	'바'
06	'사'
07	'아'
08	'자'
09	'차'
10	'카'
11	'타'
12	'파'
13	'하'

아스키 테이블

- 미국 ANSI에서 정의한 정보 교환목적의 7비트 부호체계
- 영어, 숫자, 특수문자 등 128개의 문자를 표현

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040	 	Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	\$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	((72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051))	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[[123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174	|	
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135]]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177		DEL

EUC-KR

- '가'부터 '힉'까지 한글 2,350자를 표현
- EUC-KR로는 모든 문자를 표현하지 못함
 - 현대에 사용되는 한글 11,172자

예:샤샤샤 -> ???

CP949

- 마이크로 소프트가 개발한 EUC-KR의 확장형
- 대부분의 한글을 표현할 수 있음

유니코드

- 전 세계의 모든 문자를 표현하기 위한 산업 표준
- 다국어 환경의 호환성을 높이기 위한 가변길이 방식
 - UTF-8/UTF-16/UTF-32

파이썬에서 인코딩과 디코딩

- UTF-8 예제

```
변수 = "안녕하세요"  
r = 변수.encode('utf-8')  
print(r)
```

```
b'\xec\x95\x88\xeb\x85\x95\xed\x95\x98\xec\x84\xb8\xec\x9a\x94'
```

```
r.decode('utf-8')
```

안녕하세요

정규 표현식

Python을 활용한 자료구조 이해하기

정규 표현식 (Regular Expression)

- 특정한 규칙을 가진 문자열의 집합을 표현하는데 사용하는 형식 언어
 - 텍스트 데이터를 전처리(pre-processing) 하는 용도로 사용
- 정규 표현식의 사용 예시
 - 웹 페이지에서 핸드폰 번호를 수집
 - 웹 페이지에서 email 주소를 수집
 - 사용자가 입력한 email 주소가 올바른 지 확인

정규 표현식 모듈

- 「파이썬에서 정규 표현식을 다루는 유용한 모듈: re」
 - 특정 규칙이 있는 텍스트 데이터를 빠르게 정제

임포트 방식 예시	구문사용 예시
import re	re.compile()
from re import compile	compile()
from re import *	compile
import re as regular	regular.compile()

정규 표현식의 필요성 (1/2)

- 주민등록번호의 뒷자리를 제거하는 파이썬 코드

```
# day02/02_regex/01.py
text = '''
park 800905-1049118
kim 700905-1059119
'''
```

```
result = []
for line in text.split("\n"):
    for token in line.split(" "):
        if len(token) == 14 and token[:6].isdigit() and token[7:].isdigit():
            token = token[:6] + "-" + "*****"
            result.append(token)

output = "\n".join(result)
print(output)
```

결과값:

```
park 800905-*****
kim 700905-*****
```


정규 표현식의 필요성 (2/2)

- 정규식을 사용한 예제 코드

```
import re

p = re.compile(r"(\d{6})-(\d{7})")
text1 = p.sub(r"1-*****", text)
print(text1)
```

- * 정규 표현식을 사용하면 이렇게 간단한 예제에서도 코드가 상당히 간결해 진다.
- * 만약 찾으려는 문자열 또는 바꾸어야 할 문자열의 규칙이 매우 복잡하다면 정규식의 효용은 더 커짐

re 모듈 함수

모듈 함수	설명
re.compile()	정규표현식을 컴파일하는 함수입니다. 다시 말해, 파이썬에게 전해주는 역할을 합니다. 찾고자 하는 패턴이 빈번한 사용되는 경우에 미리 컴파일해놓고 사용하면 속도와 편의성 면에서 유리합니다.
re.search()	문자열 전체에 대해서 정규표현식과 매치되는지를 검색합니다.
re.match()	문자열의 처음이 정규표현식과 매치되는지를 검색합니다.
re.split()	정규 표현식을 기준으로 문자열을 분리하여 리스트로 리턴합니다.
re.findall()	문자열에서 정규 표현식과 매치되는 모든 경우의 문자열을 찾아서 리스트로 리턴합니다. 만약, 매치되는 문자열이 없다면 빈 리스트가 리턴됩니다.
re.finditer()	문자열에서 정규 표현식과 매치되는 모든 경우의 문자열에 대한 이터레이터 객체를 리턴합니다.
re.sub()	문자열에서 정규 표현식과 일치하는 부분에 대해서 다른 문자열로 대체합니다.

findall 함수

- findall은 문자열에서 검출된 패턴 전체를 리스트로 반환

```
import re

text = "불가능을 가능하게!"
p = re.compile(r"가능")
m = re.findall(p, text)
print(m)
```

결과값: ['가능', '가능']

연습 문제

- 단어 "기린"의 출현 빈도를 출력하세요

```
text = "내가 그린 기린 그림은 잘 그린 기린 그림이고  
네가 그린 기린 그림은 잘 못 그린 기린 그림이다."  
p = re.compile(r"기린")  
m = re.findall(p, text)  
print(m)
```

결과값: 4

split 함수

- 패턴을 구분자로 문자열을 분할하여 리스트로 반환 (구분자는 제거)

```
text = "불가능을 가능하게!"  
p = re.compile(r"가능")  
m = re.split(p, text)  
print(m)
```

결과값: ['불', '을 ', '하게!']

sub 함수

- substitution는 패턴을 입력한 문자열로 치환

```
text = "010-1234-5678"  
p = re.compile(r"-")  
m = re.sub(p, "/", text)  
print(m)
```

결과값: 010/1234/5678

- 패턴 캡처와 함께 사용
 - \1은 캡처된 첫 번째 패턴을 의미

```
text = "010-1234-5678"  
p = re.compile(r"(010)-")  
m = re.sub(p, r"\1@", text)  
print(m)
```

결과값: 010@1234-5678

패턴 캡처 (Grouping and Capturing)

- 패턴에 일치하는 문자열 혹은 문자열 일부를 특정(추출) 할 때 사용
 - 괄호로 대상 지정

```
text = "21/09/06"
```

```
p = re.compile(r"(09)/(06)")  
m = re.sub(p, r"\2/\1", text)  
print(m)
```

결과값: 06/09

09/06 을 찾아서 "월"과 "일"을 치환

메타 문자 (특수 문자)

- 단순 문자가 아닌 정규 표현식에서 사용하는 특수한 명령

특수 문자	설명
.	한 개의 임의의 문자를 나타냅니다. (줄바꿈 문자인 \n는 제외)
?	앞의 문자가 존재할 수도 있고, 존재하지 않을 수도 있습니다. (문자가 0개 또는 1개)
*	앞의 문자가 무한개로 존재할 수도 있고, 존재하지 않을 수도 있습니다. (문자가 0개 이상)
+	앞의 문자가 최소 한 개 이상 존재합니다. (문자가 1개 이상)
^	뒤의 문자로 문자열이 시작됩니다.
\$	앞의 문자로 문자열이 끝납니다.
{숫자}	숫자만큼 반복합니다.
{숫자1, 숫자2}	숫자1 이상 숫자2 이하만큼 반복합니다. ?, *, +를 이것으로 대체할 수 있습니다.
{숫자,}	숫자 이상만큼 반복합니다.
[]	대괄호 안의 문자들 중 한 개의 문자와 매치합니다. [amk]라고 한다면 a 또는 m 또는 k 중 하나라도 존재하면 매치를 의미합니다. [a-z]와 같이 범위를 지정할 수도 있습니다. [a-zA-Z]는 알파벳 전체를 의미하는 범위이며, 문자열에 알파벳이 존재하면 매치를 의미합니다.
[^문자]	해당 문자를 제외한 문자를 매치합니다.
	A B와 같이 쓰이며 A 또는 B의 의미를 가집니다.

메타 문자 사용 예제 (1/2)

- 문자 중에 하나
 - [AB] : A, B 중 하나
 - [ABC] : A, B, C 중 하나
- Character Set Range
 - [A-Z] : 대문자 중 하나, [ABCDEFGHIJKLMNOPQRSTUVWXYZ]
 - [a-z] : 소문자 중 하나, [abcdefghijklmnopqrstuvwxyz]
 - [0-9] : 숫자 중 하나, [0123456789]
 - [A-z] : all characters between ASCII A to ASCII z

메타 문자 사용 예제 (2/2)

- 특정 문자 제외
 - ^ : 특정 문자는 제외하고자 할 때
 - ^[0-9] : 숫자는 제외

연습 문제

- 결과값이 출력되도록 정규 표현식을 작성하라.

```
text = '''  
a1.xls  
b1.xls  
c1.xls  
'''  
  
p = re.compile(r"[a-z]\.xls")  
m = p.findall(text)  
print(m)
```

결과값: ['a1.xls', 'b1.xls']

문자 클래스 (character class)

- 문자의 집합을 정의
 - [] 사이의 문자들과 매치

문자 클래스	설명
\\	역 슬래쉬 문자 자체를 의미합니다
\d	모든 숫자를 의미합니다. [0-9]와 의미가 동일합니다.
\D	숫자를 제외한 모든 문자를 의미합니다. [^0-9]와 의미가 동일합니다.
\s	공백을 의미합니다. [\t\n\r\f\v]와 의미가 동일합니다.
\S	공백을 제외한 문자를 의미합니다. [^\t\n\r\f\v]와 의미가 동일합니다.
\w	문자 또는 숫자를 의미합니다. [a-zA-Z0-9_]와 의미가 동일합니다.
\W	문자 또는 숫자가 아닌 문자를 의미합니다. [^a-zA-Z0-9_]와 의미가 동일합니다.

연습 문제

- 결과값이 출력되도록 정규 표현식을 작성하라.

```
text = '''
Brayden Jo 010-8212-1212
brayden.jo@outlook.com
jhyoo 010-8767-1212
jh.yoo@gmail.com
'''

# text 변수에 위와 같은 텍스트
데이터 입력

p = re.compile(r"[0-9]{3}-[0-9]{3}-[0-9]{4}")
m = p.findall(text)
# 하나 이상

print(m)
```

결과값: ['010', '8212', '1212',
'010', '8767', '1212']

연습 문제

- 결과값이 출력되도록 정규 표현식을 작성하라.

```
text = '''  
Send personal email to ben@forta.com.  
For questions  
about a book use support@forta.com.  
Feel free to send  
unsolicited email to spam@forta.com  
(wouldn't it be  
nice if it were that simple, huh?).  
'''
```

결과값: ['ben@forta.com', 'support@forta.com', 'spam@forta.com']

반복문

Python을 활용한 자료구조 이해하기

파이썬 for 문 (1/3)

- 자료구조의 데이터 수 만큼 코드를 반복 실행

for 변수 **in** 자료구조 :
code

들여쓰기!



```
리스트 = ["가", "나", "다", "라"]
```


```
for 변수 in 리스트 :  
    print(변수)
```

```
가  
나  
다  
라
```


파이썬 for 문 (2/3)

- 인덱스를 응용하는 For문

```
stocks = ["Naver", "Samsung", "SK Hynix"]  
  
for i in [0, 1, 2] :  
    print(stocks[i])
```



인덱싱!

파이썬 for 문 (3/3)

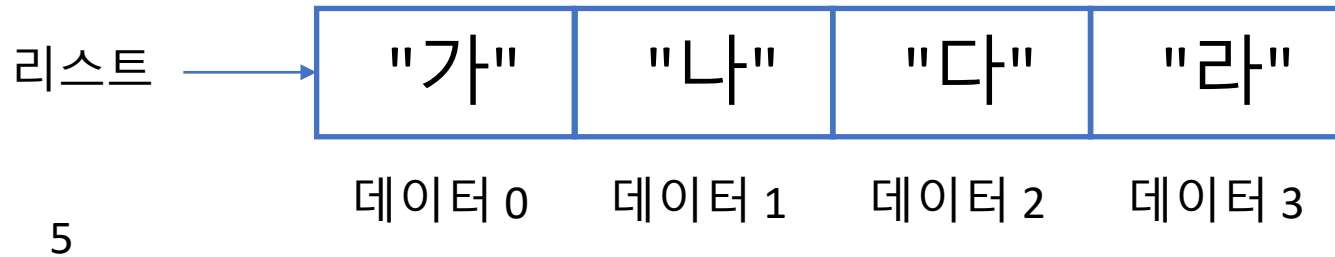
- 인덱스와 데이터를 한 번에 얻어오는 enumerate

```
stocks = ["Naver", "Samsung", "SK Hynix"]  
  
for i, val in enumerate(stocks) :  
    print(i, val)
```

1차원 for문

- For문이 데이터 (문자) 하나씩 바인딩을 함

```
리스트 = ["가", "나", "다", "라"]  
  
for 변수 in 리스트 :  
    print(변수)
```



```
변수 = "가"  
print(변수)  
  
변수 = "나"  
print(변수)  
  
변수 = "다"  
print(변수)  
  
변수 = "라"  
print(변수)
```

리스트 중첩하기

- 행과 열이 있는 2차원 데이터를 리스트로 표현

```
my_list = [ [1, 2],  
            [3, 4] ]
```

==

	A	B
1	1	2
2	3	4

연습 문제

- 아래의 표를 apart 이름의 2차원 리스트로 표현하라

```
apart = [ ?? ]
```

301호	302호	303호
201호	202호	203호
101호	102호	103호

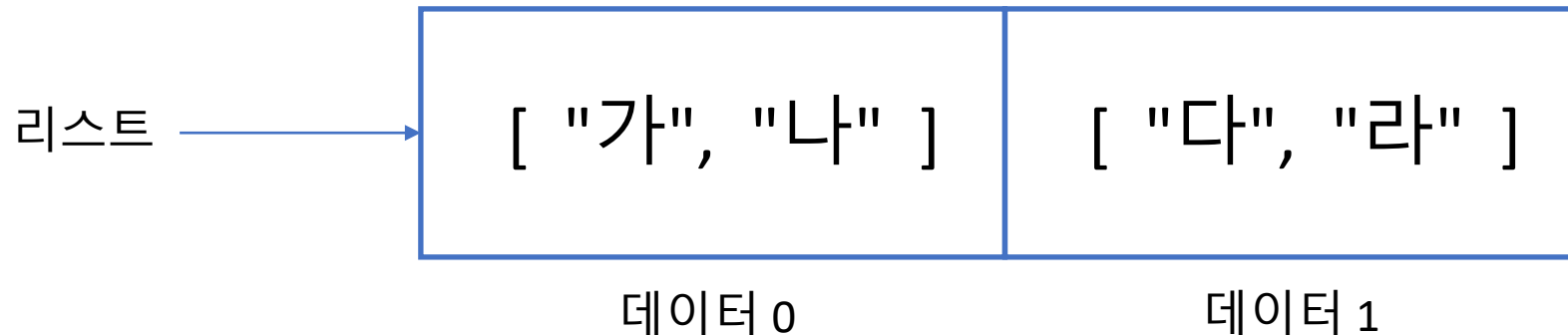
2차원 for문

- For문이 데이터 (**리스트**) 하나씩 바인딩을 함

```
이차원 = [["가", "나"], ["다", "라"]]  
for 변수 in 이차원 :  
    코드
```

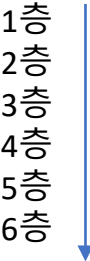
변수 = ["가", "나"]
코드

변수 = ["다", "라"]
코드



중첩된 for문 (1/2)

- 반복문을 중첩해서 사용
 - 2차원 데이터 하나 하나에 접근할 때 사용
 - 많은 데이터가 2차원 형태로 저장



	B	C	D	E	F
1층	종목명	현재가	시가	고가	저가
2층	3S	2,000	2,050	2,050	1,980
3층	AJ네트웍스	7,500	7,350	7,630	7,290
4층	AJ렌터카	15,450	15,050	15,550	14,900
5층	AK홀딩스	66,300	64,300	66,400	63,900
6층	APS홀딩스	10,700	10,000	10,800	10,000

x01호 x02호 x03호 x04호 x05호

```
price = [  
    [2000, 2050, 2050, 1980],  
    [7500, 2050, 2050, 1980],  
    [15450, 15050, 15550, 14900],  
    ...  
]
```

```
for line in price :  
    for column in line :  
        print(column)
```

```
2000  
2050  
2050  
2980  
...
```

중첩된 for문 (2/2)

- For 문 TIP
 - 1) 첫 번째 for 문은 line 단위로 움직인다
 - 2) 두 번째 for 문은 column 단위로 움직인다.

```
for line in price :  
    for column in line :  
        print(column)
```



```
price = [  
    [2000, 2050, 2050, 1980],  
    [7500, 2050, 2050, 1980],  
    [15450, 15050, 15550, 14900],  
    ...  
]
```

```
line = [2000, 2050, 2050, 1980]  
for column in line :  
    print(column)
```

```
line = [7500, 2050, 2050, 1980]  
for column in line :  
    print(column)
```

```
line = [15450, 15050, 15550, 14900]  
for column in line :  
    print(column)
```


연습 문제

- 아파트 변수를 사용해서 다음과 같이 출력하라.

```
apart = [[101, 102],  
          [201, 202]]
```

```
101호  
102호  
201호  
202호
```

연습 문제

- 리스트에 저장된 값을 아래와 같이 차례로 출력하라.

```
apart = [[101, 102, 103],  
         [201, 202, 203],  
         [301, 302, 303],  
         [401, 402, 403]]
```

출력 예시

```
101 호  
102 호  
103 호  
201 호  
202 호  
(생략)  
402 호  
403 호
```

While문

Python을 활용한 자료구조 이해하기

While 문

- for문 vs. while문
 - 일반적으로 파이썬의 'for 문'은 반복할 횟수가 적거나 리스트, 튜플, 사전과 같은 파이썬 자료 구조와 같이 사용됨
 - 반면, 'while 문'은 반복해야 할 횟수가 상대적으로 크거나 **횟수가 정해져 있지 않은 경우**에 많이 사용됨
 - 'for 문'을 사용하나 'while 문'을 사용하나 결과는 동일

While문 사용 예제

- while 문을 이용한 1~5 출력

```
num = 1

while num <= 5 :
    print (num)
    num = num + 1
```

```
1
2
3
4
5
```

num	
1 <= 5	print(num) num = num+1
2 <= 5	print(num) num = num+1
3 <= 5	print(num) num = num+1
4 <= 5	print(num) num = num+1
5 <= 5	print(num) num = num+1
6 <= 5	

While문 사용 예제

- 1초에 한 번씩 무한히 실행

```
import time

while True :
    print (num)
    num = num + 1
    time.sleep(1)
```

break

- break를 만나는 순간 가장 가까운 반복문(for/while)을 빠져 나간 후 그 다음에 위치하는 문장을 실행한다.

```
while 1 :  
    print("Find Stocks")  
    break
```

Find Stocks

continue

- While 문이나 for 문을 사용한 반복문에서 break를 사용하면 반복문 전체를 빠져나오게 됩니다. 그런데 반복문 전체를 빠져나오는 것이 아니라 **특정 조건만 건너뛰고 싶을 때**는 어떻게 하면 될까요?
- 예를 들어 화면에 1부터 10까지를 출력을 하는데 5인 경우에만 출력하지 않고 건너뛰는 것처럼 말이죠. 이럴 때 사용할 수 있는 키워드가 바로 'continue'입니다.

```
num = 0
while num < 5 :
    num += 1
    if num == 3 :
        continue;
    print (num)
```

```
0
1
2
4
5
```


연습 문제: Up & Down

- 숫자 Up & Down 게임 만들기
 - 게임이 시작되면 1~99 범위 내에서 랜덤으로 숫자를 생성
 - 사용자로부터 숫자를 입력
 - 랜덤 숫자가 사용자가 입력한 수보다 크면 Up을 출력
 - 랜덤 숫자가 사용자가 입력한 수보다 작으면 Down을 출력
 - 랜덤 숫자가 사용자가 입력한 수와 같다면 '정답'을 출력한 후 프로그램 종료
 - 시도 횟수는 총 10번

연습 문제: Up & Down 업그레이드

- 요구 사항
 - 임의의 숫자를 생성하는 RandInt 클래스
 - UpDownGame 클래스
 - print_menu
 - print_sub_menu

연습 문제: 로또 번호

- lotto 자동번호 추출 프로그램 만들기
 - 1~45까지의 숫자 6개를 생성하기
 - 숫자는 중복으로 생성될 수 없음

```
import random    # 모듈 import는 한번만
```

```
number = random.randint(1, 45) # 숫자를 만들 때 마다 호출
```

연습 문제: 로또 번호 업그레이드

- Lotto 클래스 정의
 - 객체 생성시 6개의 번호를 생성하는 클래스
 - 인스턴스 변수에 리스트로 로또 번호 저장
 - 다른 로또 번호와 일치 여부를 비교하는 compare 함수 구현

연습 문제: 명함 관리

- 사용자가 입력한 값을 정보를 저장하는 프로그램
 - 데이터를 리스트에 저장

1. 추가
2. 출력
선택: 1

> 이름: 유종훈
> 나이: 10

1. 추가
2. 출력
선택: 1

> 이름: 조아름
> 나이: 12

1. 추가
2. 출력
선택: 2

유종훈 10살
조아름 12살

연습 문제: 명함 관리 업그레이드

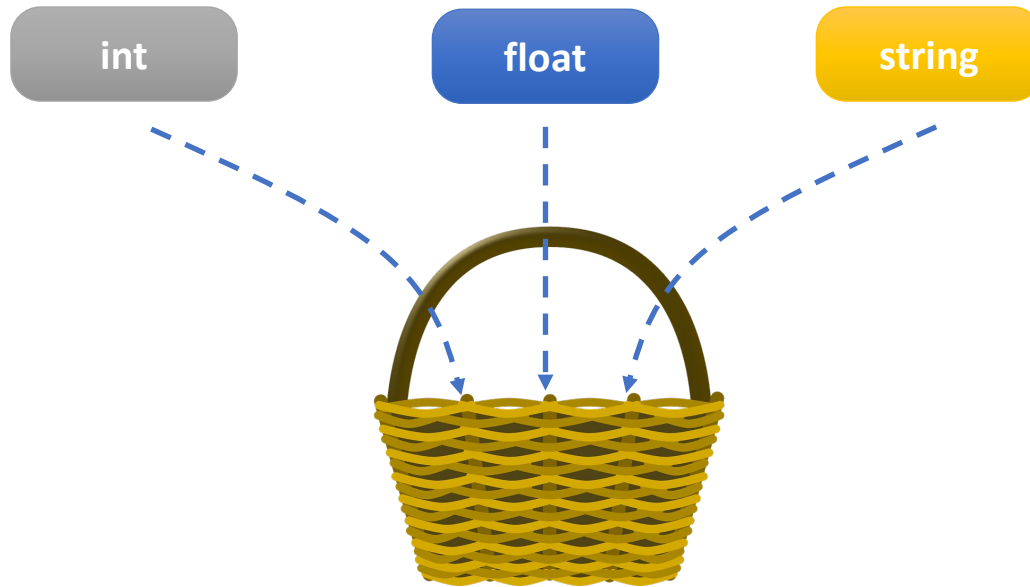
- 클래스로 코드 정리
 - 사람 인적 사항을 저장하는 Card 클래스
 - 추가 삭제하는 CardManager 클래스

파이썬 리스트-0

Python을 활용한 자료구조 이해하기

파이썬 리스트 (1/3)

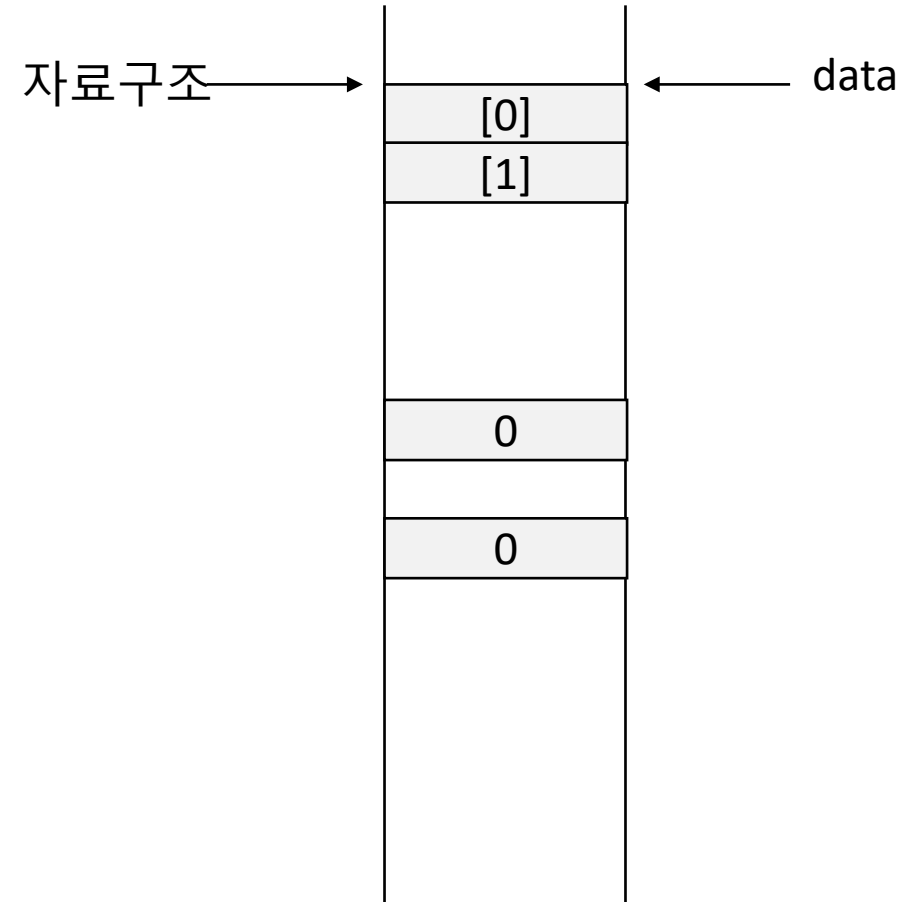
- 순서가 있는 수정 가능한 자료구조
 - 인덱싱과 슬라이싱 사용 가능



파이썬 리스트 (1/3)

- 메모리와 파이썬
 - 객체를 가리키는 형태로 보관

자료구조 = [0, 0]

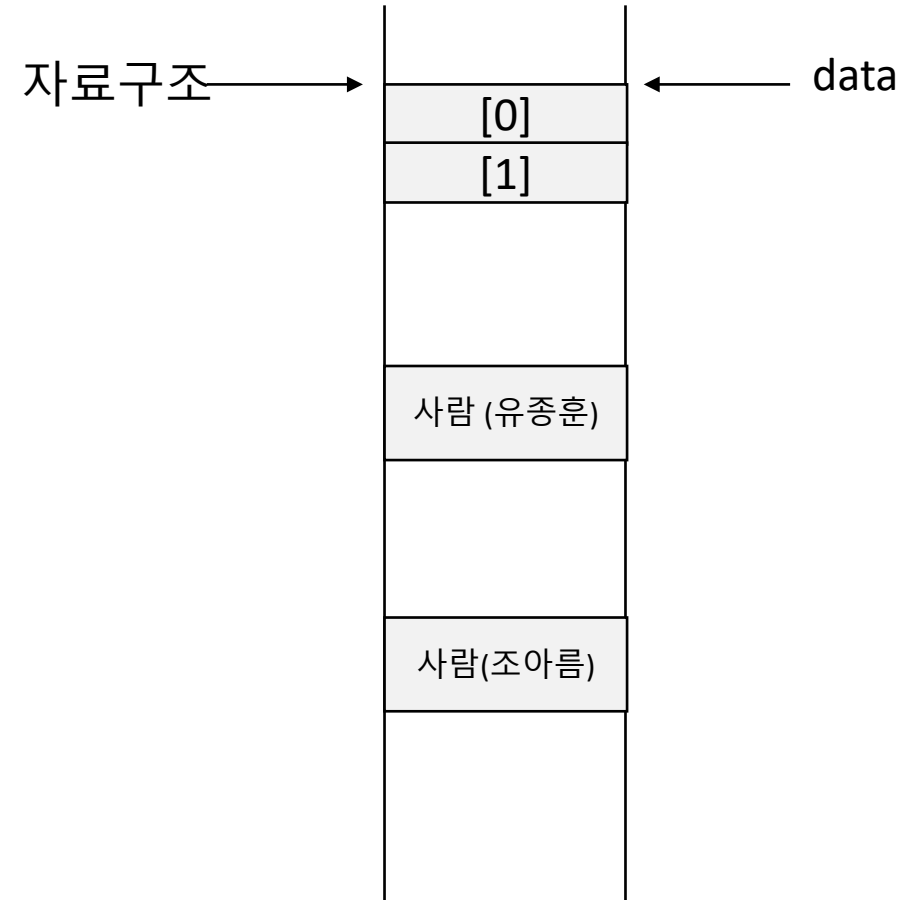


파이썬 리스트 (3/3)

- 다양한 데이터 타입을 저장

```
Class 사람:  
    pass
```

```
자료구조 = [ 사람("유종훈"), 사람("조아름") ]
```



연습 문제

- 자료구조에 저장된 사람의 모든 이름을 출력하라.

```
Class 사람:  
    def __init__(self, name):  
        self.name = name
```

```
자료구조 = [ 사람("유종훈"), 사람("조아름") ]
```

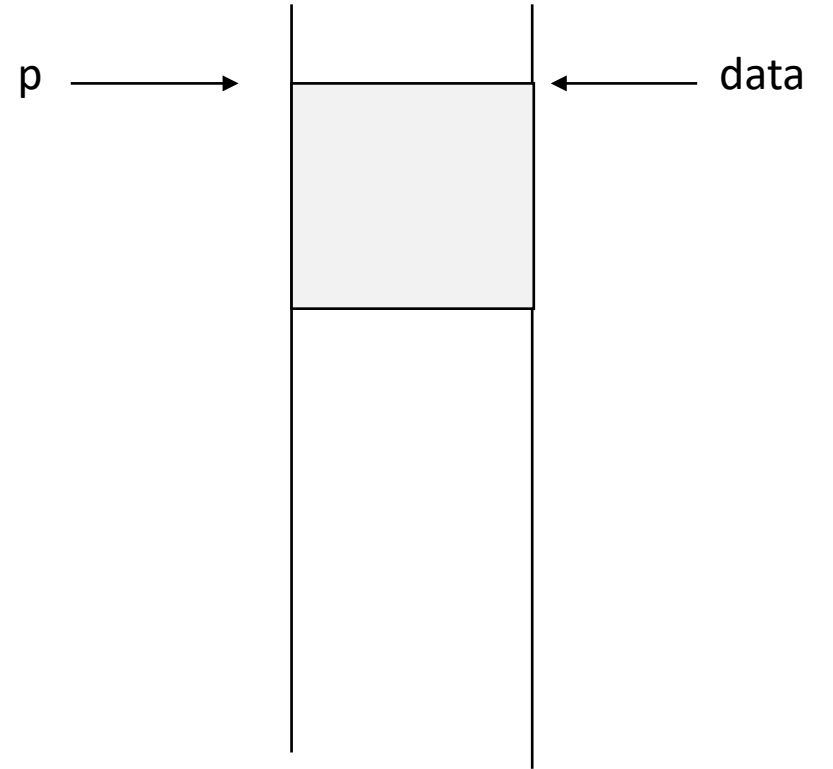
```
유종훈  
조아름
```

얕은 복사 (Shallow Copy)

- 참조에 의한 할당으로 최소한의 복사

```
data = [ 0, 1, 2]  
p = data
```

```
p[ 0 ] = 2  
print(data)
```



비교 연산자 (IS)

- 같은 객체인지 비교

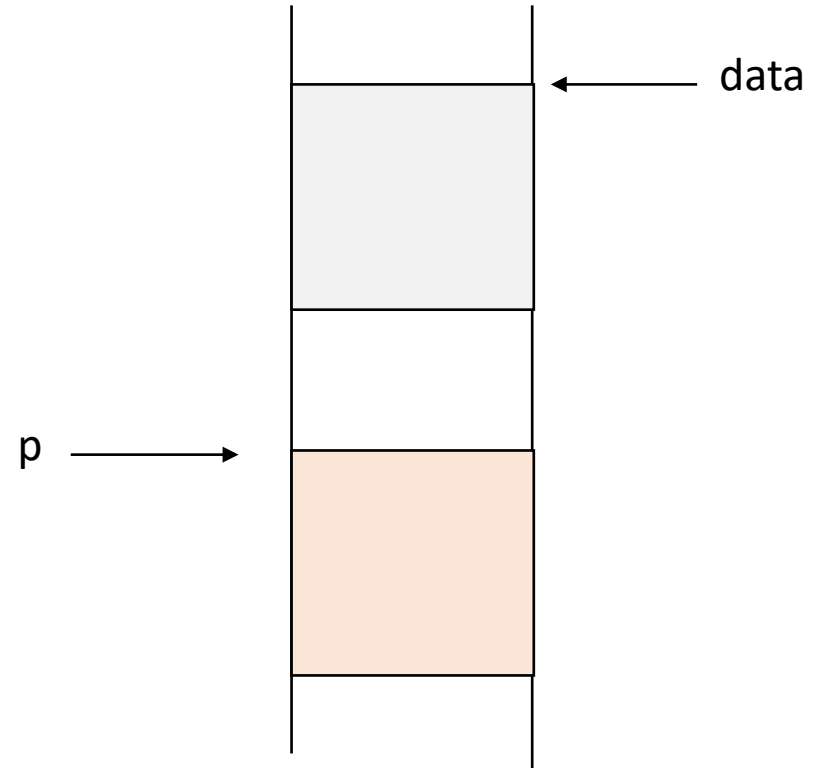
```
> cand0 = [1, 2, 3]  
> cand1 = [1, 2, 3]  
> cand0 is cand1
```

```
> cand0 = [1, 2, 3]  
> cand1 = [1, 2, 3]  
> cand0 == cand1
```

깊은 복사 (Deep Copy)

- 메모리에 객체를 복사해서 추가 생성

```
import copy  
p = copy.deepcopy(data)
```



연습 문제 – 파이썬의 착한 거짓말

- 다음 코드는 깊은 복사인가? 얇은 복사인가?

```
a = [1, 2, 3]  
b = list( a )
```

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

```
c = "Hi"  
d = str( c )
```

리스트 컴프리헨션 (Comprehension)

- 반복문, 계산식, 조건식을 축약해서 리스트를 생성하는 방법

```
data = [ 0, 1, 2]
```

```
result = [ ]
```

```
for x in data:  
    result.append(x)
```



```
result = [ x for x in data]
```


리스트 컴프리헨션과 연산

```
data = [ 0, 1, 2]  
result = [ ]
```

```
for x in data:  
    if x % 2 == 0:  
        result.append(x)
```

→ `result = [x for x in data if x % 2 == 0]`

```
data = [ 0, 1, 2]  
result = [ ]
```

```
for x in data:  
    result.append(x + 10)
```

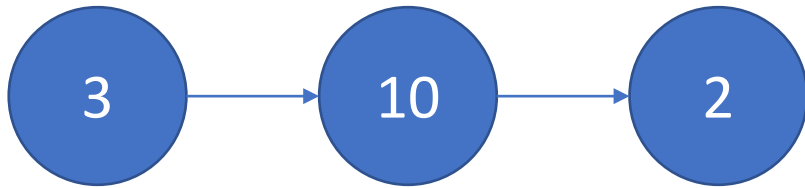
→ `result = [x + 10 for x in data]`

파이썬 리스트 - 1

Python을 활용한 자료구조 이해하기

어플리케이션에 따른 성능

- 읽기 연산이 빈번한 경우 정렬되지 않은 데이터는 성능 저하
 - 어플리케이션에 적합한 형태로 데이터 관리 필요
 - 정렬 연산은 비교적 높은 계산 복잡도를 가짐



```
data = [ 3, 10, 2 ]  
data.sort()  
print(data[-2])
```

예) 두 번째 큰 값을 찾는 경우

연습 문제

- 정렬 기능 구현하기
 - 반복문을 사용해서 데이터를 오름 차순으로 정렬하기

```
data = [ 3, 10, 1 ]
```

연습 문제

- 오름 차순으로 데이터를 정렬하는 클래스
 - 데이터를 추가할 때 정렬해서 저장

```
class MySortedList(list) :  
    def __init__(self, data):  
        super().__init__(data)  
  
    // your code
```

```
data = MySortedList([ 1, 10, 2] )  
print(data)
```

연습 문제

- append 메서드를 추가하라.
 - 추가한 값을 오름 차순으로 정렬해서 저장

연습 문제

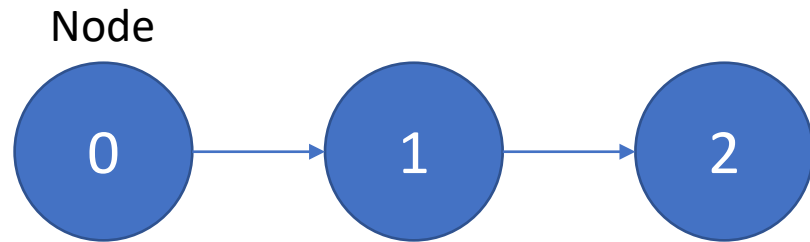
- 내림 차순으로 데이터를 정렬하는 클래스를 설계하라.

연결 리스트

Python을 활용한 자료구조 이해하기

연결 리스트 (Linked List)

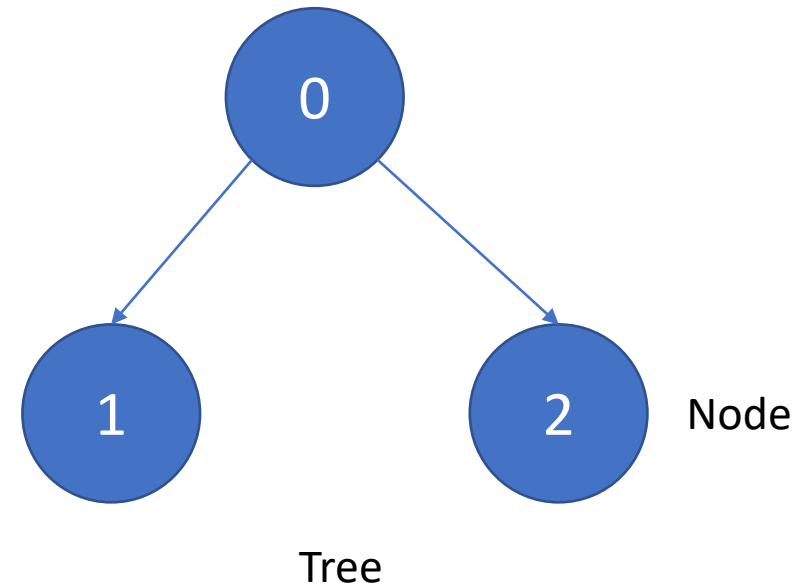
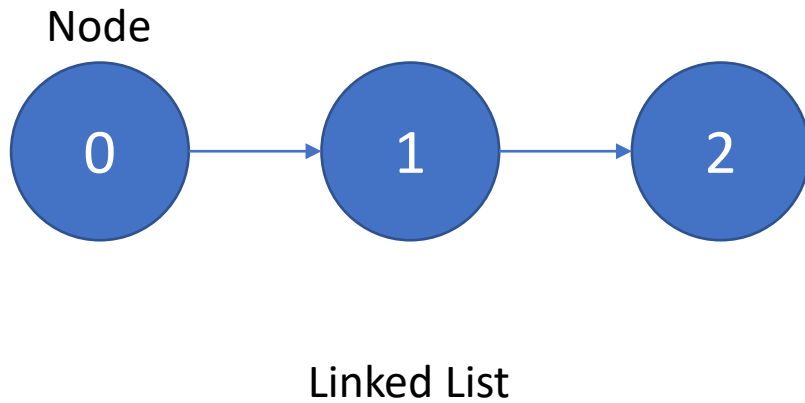
- 데이터를 연결해 놓은 자료구조
 - 순서가 있는 자료구조
 - 파이썬 List도 일종의 연결 리스트



Linked List

노드 (Node)

- 자료구조에서 데이터를 저장하는 단위
- 연결 방식에 따라 자료구조의 특성이 결정 됨



노드 클래스

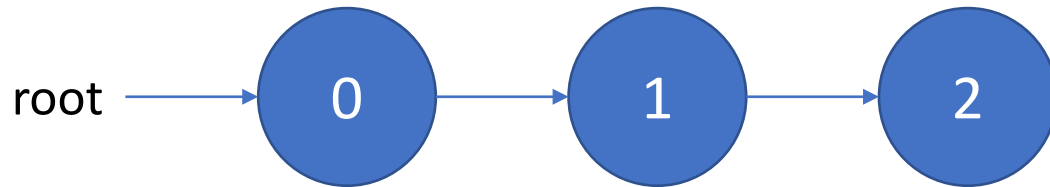
- 클래스로 표현

```
class Node:  
    def __init__(self, val, next=None):  
        self.data = val  
        self.next = next
```

연습 문제: 단순 연결

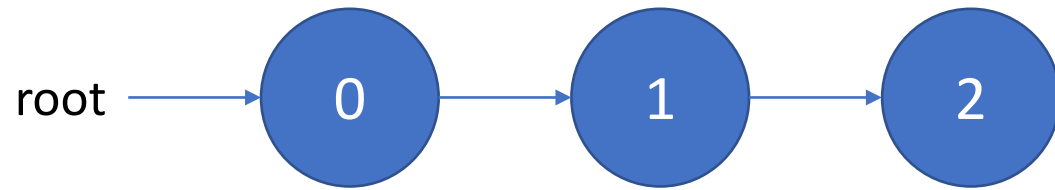
- 다음 구조로 노드를 연결하라.
 - Node는 0, 1, 2 순서대로 생성
 - 순서대로 값을 연결

```
a = Node(0)  
b = Node(1)  
c = Node(2)
```



연습 문제

- 모든 값을 화면에 출력하라.
 - root에서 시작해서 노드를 이동하며 값을 출력



클래스로 코드 정리

- 노드 연결 관계를 정의하는 클래스 정의
 - 가독성 및 재사용성 향상

```
l = LinkedList( )
```

```
n = Node(1)  
l.push(n)
```

```
n = Node(2)  
l.push(n)
```

```
n = Node(2)  
l.push(n)
```

```
l.show()
```

클래스로 코드 정리

- 클래스를 정의하는 사람의 의도에 맞게 코드 정리

```
l = LinkedList( )
```

```
l.push(1)
```

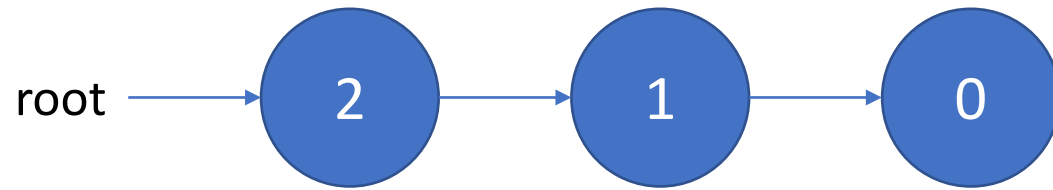
```
l.push(2)
```

```
l.push(3)
```

```
l.show()
```

연습 문제

- 최신 추가된 데이터를 root에 연결하는 Linked List 클래스를 정의하라.



연습 문제

- 최대값을 찾는 max 메서드를 구현하라.

```
l = LinkedList( )
```

```
l.push(1)
```

```
l.push(2)
```

```
l.push(3)
```

```
l.max()
```

3

연습 문제

- 데이터를 정렬해서 저장하는 SortedList 클래스를 정의해라
 - 데이터를 내림 차순으로 정렬해서 데이터를 추가한다.

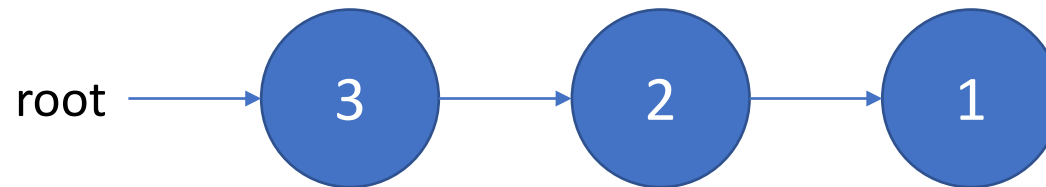
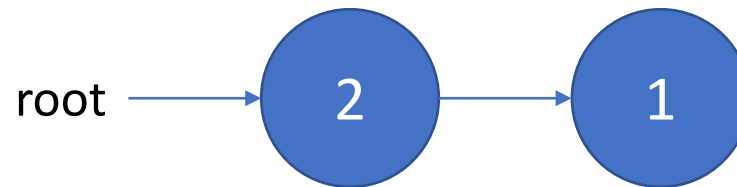
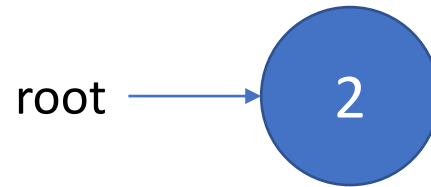
```
l = LinkedList( )
```

```
l.push(2)
```

```
l.push(1)
```

```
l.push(3)
```

```
l.max()
```



파이썬 Set 자료구조

Python을 활용한 자료구조 이해하기

집합 자료형 SET

- 중복을 허용하지 않는 자료구조
 - 데이터의 저장 순서 없음에 주의

```
s = set( "Hello World" )
```

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

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

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

연습 문제

- 다음 코드의 출력 결과를 예상하라.

```
s = set( ["Hello", "World", "Hi"])  
print(s)
```

교집합

- 공통으로 포함하고 있는 데이터를 SET 자료구조로 반환

```
s0 = set( [1, 2, 3] )  
s1 = set( [2, 3, 4] )
```

```
s0 & s1
```

```
s0.intersection(s1)
```

합집합

- 포함된 모든 데이터를 중복없이 SET 자료구조로 반환

```
s0 = set( [1, 2, 3] )  
s1 = set( [2, 3, 4] )
```

```
s0 | s1
```

```
s0.union(s1)
```

차집합

- s0에 속하지만 s1에는 속하지 않는 데이터를 선택

```
s0 = set( [1, 2, 3] )  
s1 = set( [2, 3, 4] )
```

```
s0 - s1
```

```
s0.difference(s1)
```


연습 문제

- 단어에 사용된 문자는 몇 개인가?
 - 공백 제거

"Hello World"

연습 문제 – 중복 탐색

- 다음 두 개의 데이터에서 중복된 단어만을 선택하라.

```
data0 = [ 'apple', 'banana', 'orange' ]  
data1 = [ 'apple', 'melon', 'banana' ]
```

연습 문제 – 데이터 전처리

- 다음 두 개의 데이터에서 중복된 단어만을 선택하라.
 - 대소 문자의 구분은 하지 않는다.
 - Apple과 apple은 같은 단어

```
data0 = [ 'apple', 'banana', 'orange' ]  
data1 = [ 'Apple', 'melon', 'Banana' ]
```

튜플 (tuple)

Python을 활용한 자료구조 이해하기

튜플이란?

- 데이터의 순서가 있는 자료구조
 - 인덱싱과 슬라이싱 사용 가능
- 저장된 요소의 삽입, 수정, 삭제 불가 (immutable)

튜플의 패킹 (packing)

- 하나 이상의 값을 튜플로 묶는 것

```
resolution = ( 3, 4 )
```

```
resolution = 3, 4
```

괄호 생략 가능

튜플의 언패킹 (unpacking)

- 튜플에 묶여 있는 값을 꺼내오는 행위

```
w, h, c = ( 3, 4, 5 )
```

```
w, *others = ( 3, 4, 5 )
```



나머지를 튜플로 저장

```
(w, h), c = ( (3, 4), 5 )
```

이차원 데이터의 언패킹

연습 문제

- data에서 이름만을 선택해서 출력하라.

```
data = ( ('유종훈', '37', 184), ('유서하', '2', 80) )  
_____ = data
```

유종훈 유서하

함수와 튜플 (1/3)

- 함수의 반환 값으로 사용
 - 함수내에서 패킹
 - 함수 호출부에서 언패킹

```
def get_nums():  
    return 1, 2, 3
```

← 나머지를 튜플로 저장

```
a, *dont_care = get_nums()
```

함수와 튜플 (2/3)

- 가변 인자
 - 인자의 수를 자유롭게 입력

```
def print_args( a, b, *others ):  
    print( a, b, others)
```

← 나머지를 튜플로 전달

```
print_args( 1, 2, 3, 4 ):
```

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

함수와 튜플 (3/3)

- Unpacking 후 데이터 전달

```
def print_sum( a, b ):
    print( a + b )
```

```
data = ( 10, 20 )
print_sum( *data ):
```

네임드 튜플

- 딕셔너리와 유사하게 이름이 부여된 튜플

```
from collections import namedtuple
```

```
Res = namedtuple('resolution', ['width', 'height'])
```

```
r = Res(3, 4)
```

```
print(r)
```

← 나머지를 튜플로 전달

네임드 튜플 인덱싱

- 숫자 인덱싱

```
Res = namedtuple('resolution', ['width', 'height'])  
r = Res(3, 4)  
print(r[0])
```

- 이름 인덱싱

```
Res = namedtuple('resolution', ['width', 'height'])  
r = Res(3, 4)  
print(r.width)
```

네임드 튜플 단점

- 기본 인수값을 설정할 수 없다.
 - 데이터에 선택적 속성이 많은 경우 불편함

딕셔너리

Python을 활용한 자료구조 이해하기

파이썬 딕셔너리

- 연관된 키-값의 쌍을 저장하는데 효율적인 자료구조
 - 데이터에 레이블을 붙여서 저장하고 할 때 사용

딕셔너리 = { 데이터0, 데이터1 }



중괄호로 딕셔너리를 정의

딕셔너리 = dict(key=value)

딕셔너리와 For문

- 딕셔너리의 메서드와 함께 사용
 - keys / values / items

```
for k in 딕셔너리:  
    pass
```

```
for v in 딕셔너리.values():  
    pass
```

```
for k, v in 딕셔너리.items():  
    pass
```

딕셔너리 컴프리헨션

- 리스트 컴프리헨션과 원리는 동일함

```
d = dict( a = 1, b = 1)
r = { }
for k, v in dict.items() :
    r[k] = v
```

```
d = dict( a = 1, b = 1)
r = { k:v in k, v in d.items() }
```

연습 문제

- 가격이 400원 이상인 데이터만을 필터링해서 딕셔너리에 저장하라.
 - 딕셔너리 컴프리헨션을 사용하라.

```
d = dict( 양파링=1000, 새우깡=300, 고래밥=600)
```

연습 문제

- 파일 (fruit.txt)에는 과일이 이름이 저장돼 있다. 단어의 빈도수를 세서 딕셔너리로 결과를 저장하라.



setdefault

- 데이터가 존재하지 않을 경우 처리방법

```
d = dict( )  
d[k] = d.setdefault( k, 0)
```

Defaultdict

- 데이터가 존재하지 않을 경우 자동으로 키추가
- default 값의 지정 가능

```
import collections
```

```
d = collections.defaultdict(int)  
print(d["a"])
```