

Python 중간고사

20191764 진영웅



# Python 중간고사 과제



학교	동양미래대학교
이름	진영웅
학번	20191764
과목	파이썬 프로그래밍

2020 학년도 1학기	전공	컴퓨터정보공학과	학부	컴퓨터공학부
과 목 명	파이썬프로그래밍(2019009-PD)			
강의실 과 강의시간	수:8(9-217),X(9-217),8(9-217)		학점	3
교과분류	이론/실습		시수	3
담당 교수	강환수 + 연구실 : 2호관-706 + 전 화 : 02-2610-1944 + E-MAIL : hskang@dongyang.ac.kr + 면담가능기간 : 화요일 13~16			
학과 교육목표				
과목 개요	2010년 이후 파이썬의 폭발적인 인기는 제4차 산업혁명 시대의 도래와도 밀접한 연관성이 있다. 컴퓨팅 사고력은 누구나가 가져야할 역량이며, 인공지능, 빅데이터, 사물인터넷 등의 첨단 정보기술이 제4차 산업혁명 시대의 기술을 이끌고 있다. 제4차 산업혁명 시대를 주도하는 핵심 기술은 데이터과학과 머신러닝, 딥러닝이며, 이러한 분야에 적합한 언어인 파이썬은 매우 중요한 언어가 되었다. 본 교과목은 파이썬 프로그래밍의 기초적이고 체계적인 학습을 수행한다. 본 교과목을 통하여 데이터 처리 방법에 대한 효율적인 파이썬 프로그래밍 방법을 학습한다.			
학습목표 및 성취수준	1. 컴퓨팅 사고력의 중요성을 인지하고 4차 산업혁명에서 파이썬 언어의 필요성을 이해할 수 있다. 2. 기본적인 파이썬 문법을 이해하고 데이터 처리를 위한 자료구조를 이해하여 적용할 수 있다. 3. 문제 해결 방법을 위한 알고리즘을 이해하고 데이터 처리에 적용 할 수 있다. 4. 파이썬 프로그램을 이용하여 실무적인 코딩 작업을 할 수 있다.			
	도서명	저자	출판사	비고
주교재	파이썬으로 배우는 누구나 코딩	강환수, 신웅현	홍릉과학출판사	
수업시 사용도구	파이썬 기본 도구, 파이썬, 아나콘다와 주피터 노트북			
평가방법	중간고사 30%, 기말고사 40%, 과제물 및 퀴즈 10% 출석 20%(학교 규정, 학업성적 처리 지침에 따름)			
수강안내	1. 파이썬의 개발 환경을 설치하고 활용할 수 있다. 2. 파이썬의 기본 자료형을 이해하고 조건과 반복 구문을 활용할 수 있다. 3. 파이썬의 주요 자료인 리스트, 튜플, 딕셔너리, 집합을 활용할 수 있다. 4. 파이썬의 표준 라이브러리와 외부 라이브러리를 이해하고 활용할 수 있다. 5. 파이썬으로 객체지향 프로그래밍을 수행할 수 있다.			

# 목차

Chapter 01 파이썬 언어의 개요와 첫 프로그래밍

Chapter 02 파이썬 프로그래밍을 위한 기초 다지기

Chapter 03 일상에서 활용되는 문자열과 논리 연산

Chapter 04 일상생활과 비유되는 조건과 반복

Chapter 05 행복의 나열인 리스트와 튜플

Chapter 06 일상에서 활용되는 문자열과 논리연산

## Chap 01 파이썬 언어의 개요와 첫 프로그래밍



# Chap 01 파이썬 언어의 개요와 첫 프로그래밍

## Section 1.1 파이썬 언어란?

- 파이썬은 배우기 쉬운 프로그래밍 언어

- ① 파이썬은 배우기 쉽고 누구나 무료로 사용할 수 있는 오픈소스 프로그래밍 언어이다.
- ② 파이썬은 네덜란드의 귀도 반 로섬이 개발했다.
- ③ 현재는 비영리 단체인 파이썬 소프트웨어 재단이 관리하고 있다.

- 현재 파이썬의 인기는 최고이다.

- ① 파이썬은 현재 미국과 우리나라의 대학 등 전 세계적으로 가장 많이 가르치는 프로그래밍 언어 중 하나이다.
- ② 비전공자의 컴퓨팅 사고력을 키우기 위한 프로그래밍 언어로도 활용
- ③ 파이썬은 배우기 쉽고 간결하며, 개발 속도가 빠르고 강력하다.
- ④ 파이썬은 라이브러리가 풍부하고 다양한 개발환경을 제공하고 있어 개발자가 쉽고 빠르게 소프트웨어 개발하는데 도움
- ⑤ 스택오버플로에서 '가장 빠르게 성장하는 프로그래밍 언어'로 선택

# Chap 01 파이썬 언어의 개요와 첫 프로그래밍

## Section 1.2 컴퓨팅 사고력과 파이썬

- 지금은 지능 · 정보화 혁명 시대인 제 4차 산업혁명 시대
  - 4차 산업혁명이란 모든 사물이 연결된 초연결 사회에서 생성되는 빅데이터를 기존 산업과 융합해 인공지능, 클라우드 등의 첨단 기술로 처리하는 정보 · 지능화 혁명 시대
- 제 4차 산업혁명 시대 인재의 핵심역량은 문제 해결 능력과 창의 · 융합 사고 능력
  - 제 4차 산업혁명을 이끌 인재가 갖춰야 할 덕목으로는 문제 해결 능력, 창의 · 융합 사고 능력, 의사소통 능력과 협업 능력, 자기 주도 학습 능력이다.
- 문제 해결 능력과 창의 · 융합 사고 능력에 필요한 컴퓨팅 사고력
  - 컴퓨터 사고력이란 컴퓨터 과학 원리와 개념을 활용해 자신의 영역과 융합할 수 있는 역량을 갖추는 것
    - > 컴퓨터 과학의 기본 개념과 원리 및 컴퓨터 시스템을 활용해 실생활 및 다양한 학문 분야의 문제를 이해하고 창의적 해법을 구현해 적용할 수 있는 능력
  - 컴퓨터 사고력 교육은 컴퓨터 분야의 문제 해결은 물론, 일상생활에서의 문제 해결에 효율적으로 사용할 수 있는 방법을 제공할 뿐 아니라 창의성을 높이기 때문에 제4차 산업혁명 시대의 인재 교육에서 가장 중요하다.

## Section 1.2 컴퓨팅 사고력과 파이썬

- 컴퓨팅 사고력 구성 요소

1. 분해 : 데이터, 프로세스 또는 문제를 작고 관리 가능한 부분으로 나눔
2. 패턴 인식 : 데이터의 패턴, 프로세스 또는 문제를 작고 관리 가능한 부분으로 나눔
3. 추상화 : 패턴을 생성하는 일반 원칙을 규정
4. 알고리즘 설계 : 이 문제와 유사한 문제 해결을 위한 단계별 지침을 개발

- 컴퓨팅 사고력 향상에 필요한 코딩 교육

- 컴퓨팅 사고력을 향상시키는 방법은 직접 프로그래밍하는 코딩 교육이다.
  - > 컴퓨터 비전공자에게 실시하는 코딩 교육은 전문적인 컴퓨터 프로그래머로 육성하기 위한 것이 아니라 문제 해결 능력과 창의·융합 사고 능력, 의사소통 및 협업 능력을 향상시키기 위한 것



# Chap 01 파이썬 언어의 개요와 첫 프로그래밍

## Section 1.2 컴퓨팅 사고력과 파이썬

- 프로그래밍의 절차 : 이해, 설계, 구현, 공유
  - 이해
    1. 주어진 문제를 이해하고 파악
    2. 분할 : 문제를 좀 더 쉬운 작은 문제들로 분해
  - 설계
    1. 패턴 인식 : 분해된 문제들 사이의 유사성 또는 패턴을 탐색
    2. 추상화 : 문제에서 불필요한 부분은 제거하고 주요 핵심 요소를 추려 내는 과정
    3. 알고리즘 : 프로그래밍 언어인 파이썬에 맞는 입력과 출력, 변수 저장 그리고 절차에 따라 구성
  - 구현
    1. 문제 해결을 위해 파이썬으로 코드 개발
    2. 작성된 코드의 실행과 테스트, 디버깅 과정을 거치면서 코드를 수정하고 필요하면 다시 설계
  - 공유
    1. 자신이 구현한 프로그램을 발표하고 다른 학습자의 프로그램과 비교
    2. 현재의 기능을 향상시키는 방향으로 프로그램 개선 연구
    3. 교수자의 피드백 및 평가

## Chap 01 파이썬 언어의 개요와 첫 프로그래밍

- 중간 점검

1. 네덜란드의 귀도 반 로섬이 개발한 오픈 소스 프로그래밍 언어는 무엇인가?

답 : 파이썬

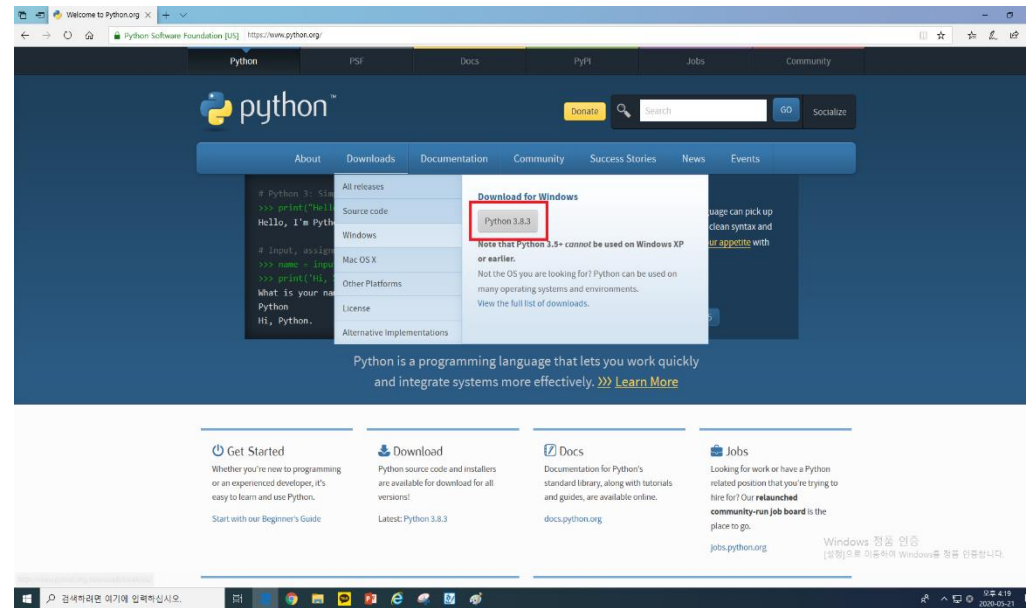
2. 컴퓨팅의 기본 개념과 원리를 기반으로 문제를 효율적으로 해결하는 사고 능력을 무엇이라고 하는가?

답 : 컴퓨팅 사고력

# Chap 01 파이썬 언어의 개요와 첫 프로그래밍

## Section 2.1 파이썬 개발 도구 설치와 파이썬 셸의 실행

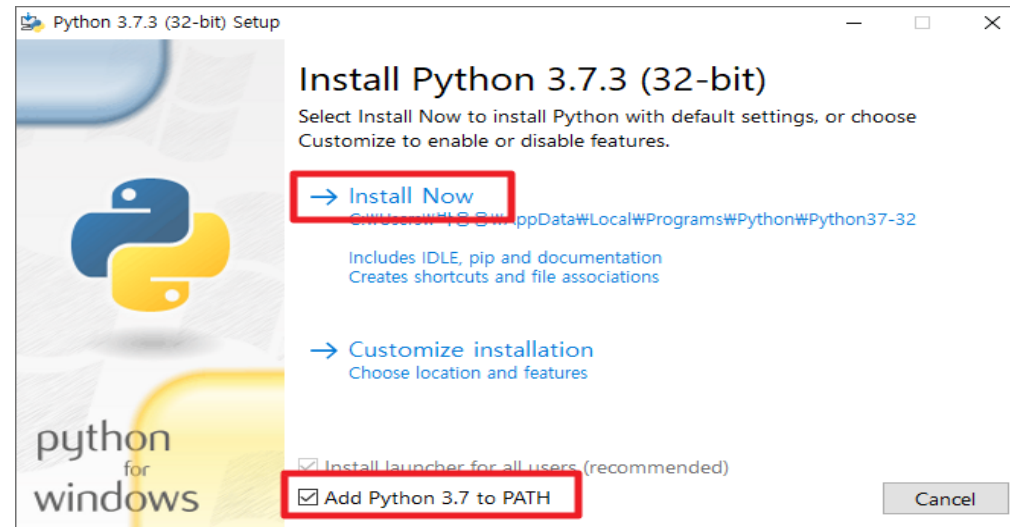
- 자신의 운영체제에 적합한 파이썬 설치
  - [www.python.org](https://www.python.org)로 들어가 Downloads에서 Python을 다운 받는다.



# Chap 01 파이썬 언어의 개요와 첫 프로그래밍

## Section 2.1 파이썬 개발 도구 설치와 파이썬 셸의 실행

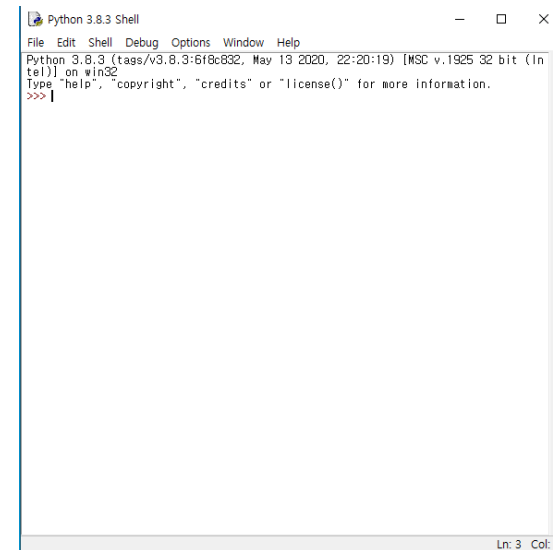
- 자신의 운영체제에 적합한 파이썬 설치
  - 쉬운 사용과 자동 path를 설정하기 위해 하단에 위치한 2개의 체크박스를 모두 선택
  - Install Now를 클릭하여 설치



# Chap 01 파이썬 언어의 개요와 첫 프로그래밍

## Section 2.1 파이썬 개발 도구 설치와 파이썬 셸의 실행

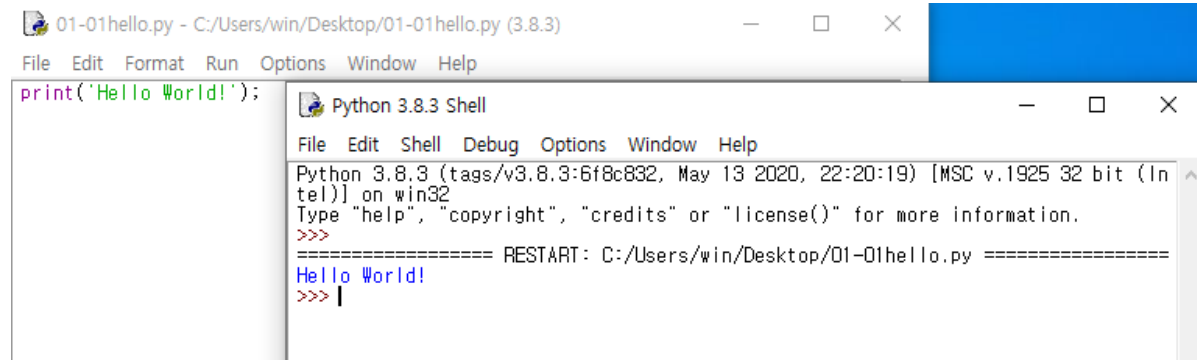
- 설치한 파이썬 셸의 실행과 종료
  - IDLE을 눌러 '파이썬 셸 IDLE'를 실행한다.



# Chap 01 파이썬 언어의 개요와 첫 프로그래밍

## Section 2.2 파이썬 쉘에서 첫 대화형 프로그래밍

- 글자 Hello World!를 출력하는 첫 대화형 코딩
  - 'Hello World'와 같이 출력하고자 하는 글자의 앞뒤에 작은따옴표(또는 홑따옴표)를 붙인다.
  - 작은따옴표를 붙인 글자를 문자열 또는 스트링(string)이라 한다.
  - New File을 눌러 새 창에 파일을 코딩
  - print('Hello World')를 입력 후에 저장하고 F5를 눌러 결과를 알아보자.



The screenshot shows a Python IDE window titled '01-01hello.py - C:/Users/win/Desktop/01-01hello.py (3.8.3)'. The menu bar includes File, Edit, Format, Run, Options, Window, and Help. The code editor contains the line `print('Hello World!');`. Overlaid on the IDE is a 'Python 3.8.3 Shell' window. Its menu bar includes File, Edit, Shell, Debug, Options, Window, and Help. The shell displays the following text: 'Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (Intel)] on win32', 'Type "help", "copyright", "credits" or "license()" for more information.', and the prompt '>>>'. Below this, a separator line reads '===== RESTART: C:/Users/win/Desktop/01-01hello.py ====='. The prompt '>>>' is followed by the output 'Hello World!' and a new prompt '>>>' with a cursor.

# Chap 01 파이썬 언어의 개요와 첫 프로그래밍

## Section 2.2 파이썬 쉘에서 첫 대화형 프로그래밍

- 두 줄의 출력이 있는 파이썬 두 번째 프로그램 작성
  - 'Hello World'를 첫 번째 줄에 출력
  - 'Hi, python'을 두 번째 줄에 출력



```
1-2.py - C:/Users/win/De Python 3.8.3 Shell
File Edit Format Run C File Edit Shell Debug Options Window Help
print('Hello World')
print('Hi, python')
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (In
tel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Users/win/Desktop/1-2.py =====
Hello World
Hi, python
>>> |
```

## Chap 01 파이썬 언어의 개요와 첫 프로그래밍

- 중간 점검

3. 글자 '파이썬'을 출력하는 프로그램을 작성하시오.

답 : `print('파이썬')`

4. 두 줄에 글자 '파이썬 개발 도구 ' 와 '파이썬 쉘 IDLE'을 출력하시오.

답 : `print('파이썬 개발 도구')`  
`print('파이썬 쉘 IDLE')`



# Chap 01 파이썬 언어의 개요와 첫 프로그래밍

## Section 3.1 쉽고 강력한 언어

- 누구나 배우기 쉬운 언어

- 파이썬의 특징

1. 간결하고, 인간의 사고 체계와 닮아 사용하기가 쉽다.  
-> 파이썬은 C나 자바 언어에 비해 문법이 쉽고, 표현 구조가 인간의 사고 체계와 닮아 있어 사용하기가 쉬우며, 간결해 초보자도 쉽게 배울 수 있다.
2. 무료이며, 다양한 자료 구조의 제공으로 생산성이 높다.  
-> 파이썬은 무료이며 자료형의 제공으로 인해 코드의 양이 줄고, 소스의 개발과 테스트도 빨라 실제 프로젝트 수행 시 생산성이 높다.
3. 라이브러리가 독보적이고 강력하며, 데이터과학과 머신 러닝 등과 같은 다양한 분야에 적용할 수 있다.  
-> 다양한 라이브러리를 쉽게 사용할 수 있도록 표준 라이브러리를 제공
4. 다른 모듈을 연결해 사용할 수 있는 풀 언어로도 자주 활용된다.  
-> 순수한 자체 언어로 프로그램을 개발하는 분야뿐 아니라 다른 언어로 개발하거나 이미 개발돼 쓰이고 있는 모듈들을 연결하는 풀 언어로도 자주 활용

# Chap 01 파이썬 언어의 개요와 첫 프로그래밍

## Section 3.2 빅데이터 처리와 머신 러닝 등 다양한 분야에 적합한 언어

- 교육과 학술, 실무 등 다양한 분야에 사용
  - 파이썬 외부에 풍부한 라이브러리가 있어 대학 프로그래밍 교양 수업부터, 스타트업, 대형 글로벌 기업에 이르기까지 다양한 분야에 활용되고 있다.
- 인공지능의 구현과 빅데이터 분석 처리 분야에 사용
  - 넘파이(NumPy), 싸이파이(SciPy), 심파이(SimPy) 등을 이용하면 수식을 빠르게 연산할 수 있기 때문
  - 판다스(Pandas), 맷플롯리브(Matplotlib), 씨본(Seaborn), 보케(Bokeh) 등은 데이터 처리와 고급 통계 차트를 그리기 위한 통계용 시각화 기능을 제공
  - 싸이킷런(Scikit-Learn), 텐서플로(TensorFlow), 케라스(Keras)는 머신 러닝에서 신경망 모형 등과 같은 딥러닝 모형을 위한 파이썬 패키지다.
- 파이썬은 좀 속도가 느리다는 단점

## Section 3.3 다양한 종류의 파이썬과 개발 환경

- 다양한 종류의 파이썬

- 인터프리터 방식의 언어인 파이썬은 다양한 버전으로 발전했으며, 파이썬 재단이 발표하는 C언어로 구현된 C 파이썬이 사실상의 표준이다.
- 여러 종류의 파이썬
  - ✓ 아이파이썬(Ipython) : 파이썬 기능에서 대화형인 REPL의 기능을 확장한 것
  - ✓ 자이썬(jython) : 과거에는 제이파이썬(J.Python)이라 불렸으며, 자바로 구현돼 자바 가상 머신에서 실행된다.
  - ✓ 아이언파이썬(IronPython) : C#언어로 작성된 닷넷(.NET) 플랫폼을 위한 파이썬
  - ✓ 파이파이(PyPy) : 파이썬으로 작성된 파이썬으로, JIT 컴파일러 기술을 도입해 Cpython보다 빠르다.

# Chap 01 파이썬 언어의 개요와 첫 프로그래밍

## Section 3.3 다양한 종류의 파이썬과 개발 환경

- 다양한 종류의 개발 환경
  - 기본 IDE에 추가 : 비주얼 스튜디오, 파이썬 도구, PyDev 설치 이클립스
  - 파이썬 전용 IDE : PyCharm, Spyder, Jupyter Notebook, Jupyter Lab 등
  - 편집기 전문 개발 환경 : Sublime Text, Visual Studio Code 등

## Section 3.4 인터프리터 방식의 언어, 파이썬

- 인터프리터와 컴파일러가 뭔가요?
  - 인터프리터 방식 : 동시 통역처럼 파이썬의 문장을 한 줄 한 줄마다 즉시 번역해 실행하는 방식
  - 컴파일 방식 : 외국 책을 번역할 때 처럼 여러 문장의 소스 단위로 번역해 기계어 파일의 실행 파일을 만든 후 실행
  - ✓ 인터프리터는 한 줄 한 줄의 해석을 담당하고 컴파일러는 컴파일을 담당하는 개발 도구 소프트웨어.

## Chap 01 파이썬 언어의 개요와 첫 프로그래밍

- 중간 점검

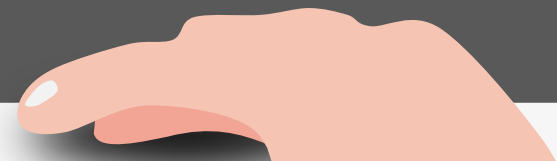
5. 파이썬의 특징 중 하나로, 다른 언어로 개발하거나 이미 개발돼 쓰이고 있는 모듈들을 연결하는 언어는 무엇인가?

답 : 풀언어

6. 문장을 한 줄 한 줄마다 즉시 번역해 실행하는 방식의 프로그램인 번역기를 무엇이라 하는가?

답 : 인터프리터

## Chap 02 파이썬 프로그래밍을 위한 기초 다지기



# Chap 02 파이썬 프로그래밍을 위한 기초 다지기

## Section 1.1 자료의 종류와 문자열 표현

- 글자의 모임인 문자열
  - 파이썬에서는 문자 하나 또는 문자가 모인 단어나 문장 또는 단락 등을 문자열이라 한다.
    - > 문자열은 '일련의 문자 모임'이라 할 수 있다.
  - 파이썬은 문자 하나도 문자열로 취급하며, 따옴표로 둘러싼 숫자(예 : '34', '3.14')도 문자열로 취급한다.
    - > 따옴표로 둘러싸면 모두 문자열이다.
- 문자열의 따옴표는 앞뒤를 동일하게 사용

예) `print("Hello string!")` -> Error

# Chap 02 파이썬 프로그래밍을 위한 기초 다지기

## Section 1.2 문자열 연산자 +, \* 와 주석

- 문자열의 연결 연산자 +와 반복 연산자 \*
  - 더하기 기호인 +는 문자열에서 문자열을 연결하는 역할
  - 문자열과 문자열 사이에 아무것도 없거나 공백이 있어도 상관없음
  - 별표 \*는 문자열에서 문자열을 지정된 수만큼 반복하는 연산을 수행한다.
  - 반복 횟수인 정수가 하나 있어야 하므로 '파이썬 \* 언어' 처럼 두 문자열로만 반복 연산자 \*를 사용하면 오류가 발생

```
print("문자열" '연결')
print("python " + "program")
print('방가 ' * 3)
print(4 * '쿨룩 ')
```

Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (Intel)] on win32  
Type "help", "copyright", "credits" or "license()" for more information.  
>>>  
===== RESTART: C:/Users/win/Desktop/1-2.py =====  
문자열연결  
python program  
방가 방가 방가  
쿨룩 쿨룩 쿨룩 쿨룩  
>>> |



## Chap 02 파이썬 프로그래밍을 위한 기초 다지기

### Section 1.2 문자열 연산자 +, \* 와 주석

- 여러 줄의 문자열의 처리에 사용되는 삼중 따옴표
  - 문자열이 길거나 필요에 의해 여러 줄에 걸쳐 문자열을 처리하기 위해서는 삼중 따옴표를 사용할 수도 있다.
  - 삼중 따옴표란, 작은 따옴표나 큰따옴표를 연속적으로 3개씩 문자열 앞뒤에 둘러싸는 것을 말한다.
- 문법과 상관 없는 주석
  - 파이썬 주석은 #으로 시작하고 그 줄의 끝까지 유효하다.

```
print(''''비록 그대가 우둔해 그 방법이 처음에는 명확해 보이지 않더라도  
지금 하는 게 아예 안 하는 것보다 낫다.'''') #이거는 주석
```

```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (In  
tel)] on win32  
Type "help", "copyright", "credits" or "license()" for more information.  
>>>  
===== RESTART: C:/Users/win/Desktop/1-2.py =====  
비록 그대가 우둔해 그 방법이 처음에는 명확해 보이지 않더라도  
지금 하는 게 아예 안 하는 것보다 낫다.  
>>> |
```

## Chap 02 파이썬 프로그래밍을 위한 기초 다지기

### Section 1.3 정수와 실수의 이해

- 수학에서 사용하던 정수와 실수
  - 숫자는 간단히 정수(integer)와 실수(real 또는 float)로 나눈다.
  - 15, 20, 7 등은 정수, 소수점이 있는 3.14, 2.781 등은 실수다.

### Section 1.4 정수와 실수의 다양한 연산

- 수의 더하기, 빼기, 곱하기, 나누기 연산자  $+ - * /$ 
  - 파이썬 셸은 간단한 계산기로 사용이 가능하며 수의 연산인 더하기, 빼기, 곱하기, 나누기가 가능하다.

```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (In
tel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> 5 + 3
8
>>> 5 - 3
2
>>> 5 * 3
15
>>> 5 / 3
1.6666666666666667
>>> |
```

## Chap 02 파이썬 프로그래밍을 위한 기초 다지기

### Section 1.4 정수와 실수의 다양한 연산

- 수의 몫, 나머지, 지수승 연산자 `//`, `%`, `**`
  - 정수 나눗셈 연산자 `//`는 나눈 몫이 결과다.
    - > 나누기 양수에서는 나누기 `/` 연산에서 소수부 없이 정수만 남는다.(실수와 정수 모두 가능)
  - 나머지 연산자 `%`는 나눈 나머지가 결과다.
    - > 5를 3으로 나누면 나머지는 2다.
  - 연속한 별표 2개인 연산자 `**`는 지수승 연산자이다.

```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (In
tel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> 8//5
1
>>> -1.5//0.2
-8.0
>>> 5%3
2
>>> 5**3
125
>>> |
```

## Chap 02 파이썬 프로그래밍을 위한 기초 다지기

### Section 1.4 정수와 실수의 다양한 연산

- 산술 연산자의 계산 우선순위

- 우선 순위는 괄호 계산, 지수승인 \*\*, 부호의 +, - 다음으로 \*, /, //, % 그리고 더하기와 빼기 연산자 +, - 이다.

연산자	명칭	의미	우선순위	예
+	더하기	두 피연산자를 더하거나 수의 부호	4, 2	4+5, +3
-	빼기	두 피연산자를 빼거나 수의 부호	4, 2	9 - 5, -7
*	곱하기	두 피연산자 곱하기	3	3 * 4
/	나누기	왼쪽을 오른쪽 피연산자로 나누기	3	10/4
%	나머지	왼쪽을 오른쪽 피연산자로 나눈 나머지	3	21 % 4
//	몫 나누기	왼쪽을 오른쪽 피연산자로 나눈 결과에서 작거나 같은 정수	3	10 // 3
**	거듭제곱	왼쪽을 오른쪽 피연산자로 거듭제곱	1	2 ** 3

## Chap 02 파이썬 프로그래밍을 위한 기초 다지기

### Section 1.4 정수와 실수의 다양한 연산

- 최근 결과의 특별한 저장 장소, 언더스코어(`_`)
  - 대화형 모드에서 마지막에 실행된 결과 값은 특별한 저장 공간인 `_`에 대입된다.
- 표현식 문자열 실행 함수 `eval()`
  - 함수 `eval('expression')`은 실행 가능한 연산식 문자열인 `expression`을 실행한 결과를 반환한다.

```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> 60
60
>>> -
60
>>> 3 + _
180
>>> -
180
>>>
===== RESTART: Shell =====
>>> eval('3 + 15 / 2')
10.5
```

## Chap 02 파이썬 프로그래밍을 위한 기초 다지기

- 중간 점검

1. 파이썬에서 다음 연산 결과는 무엇인가?

①  $10 * 3 / 2$       #15.0

②  $3 - 2 * 3 // 5$       #2

③  $81 / -3 ** 2$       #-9.0

④  $10 + 3 - 23 \% 4$       #10

⑤  $20 // 2 ** 3$       #2

2. 문자열의 반복 연산자인 \*을 사용해 다음 모양을 출력하는 프로그램을 작성하시오.

```
*      print('* * 1)
**     print('* * 2)
***    print('* * 3)
****   print('* * 4)
***** print('* * 5)
```

# Chap 02 파이썬 프로그래밍을 위한 기초 다지기

## Section 2.1 자료형

- 자료형과 type() 함수

- 지금까지 알아본 정수와 실수, 문자열 등을 자료형이라 한다.
- 파이썬에서 자료형을 살펴보면 정수는 int, 실수는 float, 문자열은 str로 사용한다.
- 자료형을 직접 알아보려면 type() 함수를 사용해야 한다.

```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> type(3)
<class 'int'>
>>> pi = 3.14
>>> type(pi)
<class 'float'>
>>> type('python')
```

# Chap 02 파이썬 프로그래밍을 위한 기초 다지기

## Section 2.2 변수와 대입연산자

- 변수의 이해와 대입 연산자 = 를 이용한 값의 저장
  - 변수란 말 그대로 '변하는 자료를 저장하는 메모리 공간'이다.
  - 변수 이름은 저장 값에 의미 있는 이름을 붙이자.
- 프로그래밍 언어가 이미 정해놓은 단어, 키워드
  - 프로그래밍 언어 문법에서 사용하는 이미 예약된 단어를 키워드라고 한다.
  - 키워드를 대화형 모드에서 확인할 수 있는데 import를 사용하면 된다.

```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> import keyword
>>> print(keyword.kwlist)
['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await', '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']
>>> |
```



# Chap 02 파이썬 프로그래밍을 위한 기초 다지기

## Section 2.2 변수와 대입연산자

- 변수 이름을 붙일 때의 규칙
  - 문자는 대소문자의 영문자, 숫자 그리고 \_로 구성되며, 대소문자는 구별된다.
  - 숫자는 맨 앞에 올 수 없다. 그러므로 영문자로 시작해야 한다.
  - Import, True, False 등과 같은 키워드는 사용할 수 없다.
  - 처음 변수에 이름을 붙여 값을 대입하는 것을 변수 선언이라고 한다.
- 동일한 변수에 값을 수정하는 다양한 대입 연산자
  - 수학에서 =는 동등 연산자이지만 파이썬에선 =이 대입 연산자이다.
  - 정의되지 않은 변수를 사용하면 오류가 발생한다.

```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> value = 1
>>> value = value + 1
>>> print(value)
2
>>> var = 5
>>> var += 10
>>> print(var)
50
>>> undefined += 2
Traceback (most recent call last):
  File "<pyshe11#6>", line 1, in <module>
    undefined += 2
NameError: name 'undefined' is not defined
>>> |
```

## Chap 02 파이썬 프로그래밍을 위한 기초 다지기

### Section 2.2 변수와 대입연산자

- 한 번에 여러 자료 대입
  - 파이썬에서는 콤마로 구분된 여러 변수에 순서대로 값을 대입할 수 있다.
  - 함수 `divmod(a, b)`는 나누기 몫 연산 `//`와 나머지 연산 `%`를 함께 수행해 2개의 결과를 반환한다.

```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (In
tel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> a, b = 5, 9
>>> print(a, b)
5 9
>>> divmod(a, b)
(0, 5)
>>> |
```

### • Section 3.1 표준 입력과 다양한 변환 함수

- 표준 입력이란?
  - 프로그램 과정에서 셸이나 콘솔에서 사용자의 입력을 받아 처리하는 방식을 표준 입력이라고 한다.

## Chap 02 파이썬 프로그래밍을 위한 기초 다지기

- 중간 점검

3. 다음 변수의 이름은 적합한가?

① 30day            #x

② ln                #x

③ subtraction    #o

④ \$dallar        #x

4. 변수가  $a = 10$ ,  $b = 5$ 인 경우 각각의 연산 결과 값은?

①  $b / a / 2$         #0.25

②  $b ** (a - 8)$     #25

③  $a // 3 - b / a$     #2.5

④  $b * (a \% 7)$      #15

## Chap 02 파이썬 프로그래밍을 위한 기초 다지기

- 중간 점검

5. 257일은 몇 월 며칠인지 출력하는 코딩을 작성하시오.  
(한 달은 30일로 계산)

```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> days = 257
>>> months = days
>>> days %= 30
>>> print(months, '달', days, '일')
257 달 17 일
```

## Chap 02 파이썬 프로그래밍을 위한 기초 다지기

### Section 3.1 변수와 대입연산자

- 함수 input()으로 문자열 표준 입력
  - 함수 input()은 입력되는 표준 입력을 문자열로 읽어 반환하는 함수이다.
  - Input()에서 반환되는 입력 문자열을 변수에 저장하려면 대입 연산자 = 를 사용해 변수에 저장하여야 한다.
- 문자열과 정수, 실수 간의 자료 변환 함수 str(), int(), float()
  - 함수 str()은 주로 정수와 실수를 문자열로 변환하는 데 사용한다.
  - 함수 int()는 정수 형태의 문자열인 '6500', '265' 등을 정수, float()는 소수점이 있는 실수 형태의 문자열인 '2.784', '3.141592' 등을 실수로 반환한다.
  - 또한 함수 int()는 실수를 정수, float()는 정수를 실수로 변환한다.

```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (In
tel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> input()
java
'java'
>>>
===== RESTART: Shell =====
>>> str(234)
'234'
>>> int(6400)
6400
>>> float('3.141592')
3.141592
>>> int(2.71828)
2
>>> float(3)
3.0
>>> |
```

## Chap 02 파이썬 프로그래밍을 위한 기초 다지기

- Section 3.1 변수와 대입연산자
  - 숫자 형태의 문자열을 정수나 실수로 변환
    - 함수 input()으로 받은 입력은 형태가 수라도 문자열이므로 필요하면 실수나 정수로 변환해야 한다.
- Section 3.2 16진수, 10진수, 8진수, 2진수와 활용
  - 16진수는 맨 앞에 0x로 시작한다.
  - 8진수와 2진수는 각각 0o, 0b로 시작한다.
  - 각각의 진수를 표시하는 알파벳은 대소문자 모두 가능하다.
- 10진수의 변환 함수 bin(), oct(), hex()

```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (In
tel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> age = input('나이는? ')
나이는? 20
>>> print('실제 나이는', int(age) + 1)
실제 나이는 21
>>>
===== RESTART: Shell =====
>>> bin(5), bin(15), bin(11)      #2진수
('0b101', '0b1111', '0b1011')
>>> oct(5), oct(10), oct(12)      #8진수
('0o5', '0o12', '0o14')
>>> hex(15), hex(12), hex(13)     #16진수
('0xf', '0xc', '0xd')
>>> |
```

## Chap 02 파이썬 프로그래밍을 위한 기초 다지기

### Section 3.2 16진수, 10진수, 8진수, 2진수와 활용

- int(정수, 진법기수)를 활용한 여러 진수 상수 형태 문자열의 10진수 변환
  - 함수 int('strnum') 또는 int('strnum', 10)은 10진수 형태의 문자열을 10진수 정수로 변환한다.
  - 여러 진수 형태 문자열을 10진수로 바꾸려면 함수 int('strnum', n)을 사용해야 한다.

```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> int('17'), int('25', 10)
(17, 25)
>>> int('11', 2)          #2진수 문자열을 10진수로 변환
3
>>> int('10', 8)          #2진수 문자열을 8진수로 변환
8
>>> int('1A', 16)         #16진수 문자열을 10진수로 변환
26
>>> |
```

## Chap 02 파이썬 프로그래밍을 위한 기초 다지기

- 중간 점검

6. 다음 문자열을 10진수로 변환하는 문장을 작성하시오.

- 6. '156'            `#int('156')`
- 7. '0b101'        `#int('0b101', 2)`
- 8. '0o11'          `#int('0o11', 8)`
- 9. '0xf'           `#int('0xf', 16)`

7. 다음과 같이 표준 입력으로 이름을 입력받아 출력하는 코드를 작성하시오.

당신의 이름은? 홍길동  
만나서 반가워요, 홍길동씨!

```
답 : name = input("당신의 이름은? ")  
print("만나서 반가워요, " + name + "씨!")
```



## Chap 03 일상에서 활용되는 문자열과 논리 연산



## Chap 03 일상에서 활용되는 문자열과 논리 연산

### Section 1.1 문자열 str 클래스와 부분 문자열 참조 슬라이싱

- 문자열은 클래스 str의 객체
  - 파이썬에서 문자열은 '문자의 나열'로 텍스트 시퀀스라고도 한다.
  - 문자열의 자료형은 class str이다.
  - 작은따옴표 : "큰" 따옴표 처럼 문자열 내부에 큰따옴표 사용 가능
  - 큰따옴표 : '작은' 따옴표 처럼 문자열 내부에 작은따옴표 사용 가능
  - 삼중따옴표 : '''문자열''' 또는 """문자열"""로 여러 줄의 문자열 표현
- 함수 len()으로 문자열의 길이 참조
  - 함수 len(문자열)으로 문자열의 길이를 알 수 있다.

```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> a = 'python'
>>> len(a)
6
>>> |
```

## Chap 03 일상에서 활용되는 문자열과 논리 연산

### Section 1.1 문자열 str 클래스와 부분 문자열 참조 슬라이싱

- 문자열의 문자 참조

- 문자열을 구성하는 문자는 0부터 시작되는 첨자를 대괄호 안에 기술해 참조가 가능하다.
- 1부터 시작돼 -2, -3으로 작아지는 첨자도 역순으로 참조한다.
- 첨자가 유효 범위를 벗어나면 IndexError가 발생한다.

```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (In
tel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> 'python'[0]
'p'
>>> 'python'[1]
'y'
>>> 'python'[-4]
't'
>>> 'python'[6]
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    'python'[6]
IndexError: string index out of range
>>> |
```

## Chap 03 일상에서 활용되는 문자열과 논리 연산

### Section 1.2 문자열의 부분 문자열 참조 방식

- 0과 양수를 이용한 문자열 슬라이싱
  - 문자열에서 일부분을 참조하는 방법을 슬라이싱이라고 한다.
  - 콜론을 사용한 [start : end]로 부분 문자열을 반환하는데 start 첨자에서 end -1 첨자까지의 문자열을 반환한다.
- 음수를 이용한 문자열 슬라이싱
  - 슬라이스 [start : end]에서는 음수도 사용할 수 있다.
  - start 첨자에서 end -1 첨자까지의 문자열을 반환한다.

```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> 'python'[1:5]
'ytho'
>>> 'python'[2:4]
'th'
>>> 'python'[-5:-1]
'ytho'
>>> 'python'[-4 : -1]
'tho'
>>> |
```

## Chap 03 일상에서 활용되는 문자열과 논리 연산

### Section 1.2 문자열의 부분 문자열 참조 방식

- start와 end를 비우면 '처음부터'와 '끝까지'를 의미
  - 슬라이싱에서 start와 end를 비울 수 있으며, 각각 '처음부터'와 '끝까지'를 의미한다.
- str[start:end:step]으로 문자 사이의 간격을 step으로 조정 가능
  - 문자 사이의 간격을 step으로 조정하려면 str[start:end:step]을 사용해야 한다.
  - Step을 생략하면 1이다.
  - 간격인 step은 음수도 가능하며, 이 경우 start는 end보다 작아야 한다.

```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> 'python'[:3]
'pyt'
>>> 'python'[1:]
'ython'
>>> 'python'[::-1]
'nohtyp'
>>> 'python'[1:5:2]
'yh'
>>> |
```

## Chap 03 일상에서 활용되는 문자열과 논리 연산

### Section 1.3 문자 함수와 이스케이프 시퀀스

- 문자 함수 ord()와 chr()

- 한글 문자 '가'의 유니코드 번호는 내장 함수 ord('가')로 알 수 있다.
- 반대로 유니코드 44032로 내장 함수 chr(44032)를 호출하면 문자를 반환한다.

```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> ord('가') #44032
44032
>>> chr(44032) #'가'
'가'
>>> hex(ord('가')) #44032의 16진수 ac00
'0xac00'
>>>
```

- 이스케이프 시퀀스 문자

- 하나의 문자를 역슬래시로 시작하는 조합으로 문자를 이스케이프 시퀀스 문자라고 한다.
- 문자열 내부에도 사용할 수 있다.

## Chap 03 일상에서 활용되는 문자열과 논리 연산

### Section 1.3 문자 함수와 이스케이프 시퀀스

- 이스케이프 시퀀스 문자

Escape Sequence

\a	경고음
\b	백 스페이스
\f	폴 피드 (프린트 전용)
\n	개행
\r	캐리지 리턴
\t	수평 탭
\v	수직 탭 (프린트 전용)
\'	작은 따옴표 출력
\"	큰 따옴표 출력
\?	물음표 출력
\\	역슬래시 출력
\ooo	아스키 문자 8진수 표시
\xhhh	아스키 문자 16진수 표시

## Chap 03 일상에서 활용되는 문자열과 논리 연산

### Section 1.3 문자 함수와 이스케이프 시퀀스

- 문자열의 최대와 최소
  - Min()은 최솟값을 반환하는 함수이다.
  - Max()는 최댓값을 반환하는 함수이다.
  - 인자가 문자열 1개이면 문자열을 구성하는 문자에서 코드 값으로 최대와 최소인 문자를 반환한다.
  - 인자가 문자열 2개 이상이면 문자열 중 최대와 최소인 문자열을 반환한다.
  - 2개 이상의 숫자도 인자가 될 수 있으며 최대와 최소를 반환한다.

```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> min('ipython')
'h'
>>> max('ipython')
'y'
>>> min(3, 96.4, 13)
3
>>> max(3, 96.4, 13)
96.4
>>> |
```



## Chap 03 일상에서 활용되는 문자열과 논리 연산

- 중간 점검

1. 다음 출력값을 기술하시오.

- ① `print(ord('D') - ord('A'))`      `#3`
- ② `print(min('java'), max('java'))`      `#a v`
- ③ `print(len('python') - len('java'))`      `#2`
- ④ `print(len('abWtd'))`      `#4`
- ⑤ `print('abcWbd')`      `#abd`

2. 다음 문자열 출력값을 기술하시오.

- ① `print('Hello_python'[5])`      `#_`
- ② `print('Hello_python'[:5])`      `#Hello`
- ③ `print('Hello_python'[6:])`      `#python`
- ④ `print('Hello_python'[::-2])`      `#Hlopto`
- ⑤ `print('Hello_python'[:len('Hello_python')])`      `#Hello_python`

## Chap 03 일상에서 활용되는 문자열과 논리 연산

### Section 2.1 문자열 대체와 부분 문자열 출현 횟수, 문자열 삽입

- 문자열 바꿔 반환하는 메소드 `replace()`
  - 클래스에 소속된 함수를 메소드라 한다.
  - 메소드의 호출은 '문자열 객체명.함수 이름(인자)'와 같이 사용한다.
  - 문자열 객체가 `str`이라면 `str.replace(a,b)`로 호출
  - `str.replace(a,b)`는 문자열 `str`에서 `a`가 나타나는 모든 부분을 `b`로 모두 바꾼 문자열을 반환한다.
  - `replace(old, new, count)`는 문자열 `old`를 `new`로 대체하는데, 옵션인 `count`는 대체 횟수를 지정한다.
  - 옵션인 `count`가 없으면 모두 바꾸고, 있으면 앞에서부터 지정한 횟수만큼 바꾼다.

```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> str = '파이썬 파이썬 파이썬'
>>> str.replace('파이썬', 'python!')
'python! python! python!'
>>>
===== RESTART: Shell =====
>>> str = '파이썬 파이썬 파이썬'
>>> str.replace('파이썬', 'python!', 2)
'python! python! 파이썬'
>>> |
```

## Chap 03 일상에서 활용되는 문자열과 논리 연산

### Section 2.1 문자열 대체와 부분 문자열 출현 횟수, 문자열 삽입

- 부분 문자열 출현 횟수를 반환하는 메소드 `count()`
  - 문자열 `str`에서 문자나 부분 문자열의 출현 횟수를 알려면 메소드 `str.count(부분 문자열)`를 사용해야 한다.
- 문자와 문자 사이에 문자열을 삽입하는 메소드 `join()`
  - 문자열의 문자와 문자 사이에 원하는 문자열을 삽입하려면 메소드 `join()`을 사용한다.
  - 메소드 호출 `'->'.join('12345')`은 문자열 `'12345'` 사이에 `'->'`을 삽입한 문자열을 반환한다.

```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> str = '단순한 것이 복잡한 것보다 낫다.'
>>> str.count('복잡')
1
>>>
===== RESTART: Shell =====
>>> num = '12345'
>>> '->'.join(num)
'1->2->3->4->5'
>>> |
```

# Chap 03 일상에서 활용되는 문자열과 논리 연산

## Section 2.2 문자열 찾기

- 문자열을 찾는 메소드 find()와 index()
  - 클래스 str에서 부분 문자열 sub가 맨 처음에 위치한 첨자를 반환받으려면 메소드 str.find(sub) 또는 str.index(sub)를 사용한다.
  - 메소드 index()는 찾는 문자열이 없을 경우, ValueError를 발생시키지만 find()는 오류가 발생하지 않고 -1을 반환한다.

## Section 2.3 문자열 나누기

- 문자열을 여러 문자열로 나누는 split() 메소드
  - 문자열 메소드 str.split()는 문자열 str에서 공백을 기준으로 문자열을 나눠준다.
  - 만일 str.split(',')처럼 괄호 안에 특정한 문자열 값이 있을 경우에는 이 부분 문자열 값을 구분자를 이용해 문자열을 나눈다.

```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> str = '자바 C 파이썬 코틀린'
>>> str.find('자바')
0
>>> str.index('파이썬')
5
>>> str.find('C++')
-1
>>> str.index('C++')
Traceback (most recent call last):
  File "<pyshe11#4>", line 1, in <module>
    str.index('C++')
ValueError: substring not found
>>>
===== RESTART: Shell =====
>>> '사과 배 복숭아 딸기 포도'.split()
['사과', '배', '복숭아', '딸기', '포도']
>>> '데스크톱 1000000', '노트북 1800000', '스마트폰 1200000'.split(',')
('데스크톱 1000000', '노트북 1800000', ['스마트폰 1200000'])
>>> |
```

# Chap 03 일상에서 활용되는 문자열과 논리 연산

## Section 2.4 다양한 문자열 변환 메소드

- 영문자 알파벳의 다양한 변환 메소드
  - 문자열 변환 메소드는 upper()와 lower()등과 같이 매우 다양하다.
- 폭을 지정하고 중앙에 문자열 배치하는 메소드 center()
  - 전체 문자열의 폭을 '30'으로 지정한 후 중앙에 '파이썬 강좌'를 배치하고 좌우의 나머지는 문자\*를 채우려면 메소드 '중앙 문자열'.center(폭, 채울 문자)처럼 사용한다.
  - 두번째 인자는 반드시 하나의 채울 문자로 선택적이며, 없으면 공백 문자로 채워진다.

```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (In
tel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> 'python'.upper()      #모두 대문자로 변환
'PYTHON'
>>> 'PYTHON'.lower()     #모두 소문자로 변환
'python'
>>> 'python lecture'.capitalize()      #첫 문자만 대문자로 변환
'Python lecture'
>>> 'python lecture'.title()      #단어마다 첫 문자를 대문자로 변환
'Python Lecture'
>>> 'PyTHon'.swapcase()      #소문자와 대문자를 서로 변환
'pYthoN'
>>>
===== RESTART: Shell =====
>>> '파이썬 강좌'.center(30, '*')
'*****파이썬 강좌*****'
>>> '파이썬 강좌'.center(30)
'   파이썬   강좌   '
>>> |
```

## Chap 03 일상에서 활용되는 문자열과 논리 연산

### Section 2.4 다양한 문자열 변환 메소드

- 폭을 지정하고 왼쪽 또는 오른쪽 정렬하는 메소드 `ljust()`와 `rjust()`
  - 문자열의 폭을 지정하고 왼쪽으로 정렬하려면 `ljust()`, 오른쪽에 정렬하려면 `rjust()`로 호출한다.
  - 빈 공간을 문자로 채우려면 두 번째 인자로 입력한다.
- 문자열 앞뒤의 특정 문자들을 제거하는 `strip()` 메소드
  - `str.strip()` 메소드는 문자열 `str`에서 맨 앞뒤의 공백 문자를 제거해 반환하는 메소드다.
  - `str.strip('문자들')`은 문자열 `str`의 맨 앞뒤에서 '문자들'에 속한 모든 문자를 제거해 반환하는 메소드다.

```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> 'python'.ljust(10)
'python'
>>> 'python'.rjust(10, '*')
'***python'
>>>
===== RESTART: Shell =====
>>> 'python'.lstrip()
'python'
>>> 'python'.rstrip()
'python'
>>> '***python---'.strip('* -')
'python'
>>> |
```

## Chap 03 일상에서 활용되는 문자열과 논리 연산

### Section 2.4 다양한 문자열 변환 메소드

- 제로 0을 채워 넣는 `zfill()` 메소드
  - 문자열의 지정한 폭 앞 빈 부분에 0을 채워 넣고 싶다면 메소드 `zfill(폭)`을 사용한다.

```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> '234'.zfill(5)
'00234'
>>> '-345'.zfill(8)
'-0000345'
>>> |
```

## Chap 03 일상에서 활용되는 문자열과 논리 연산

### Section 2.5 출력을 정형화하는 함수 format()

- 문자열의 format() 메소드를 이용해 간결한 출력 처리
  - 문자열 '{} + {} = {}' 내부에서 중괄호(curly brace) {}이 위치한 부분에 format(3, 4, 3+4) 인자인 3, 4, 7이 순서대로 출력된다.
  - 문자열에서 {}를 제외한 나머지 부분인 '+ ='은 쓰여 있는 그대로 출력된다.
  - 메소드 format()을 호출한 문자열을 print()함수의 인자로 사용하면 원하는 결과가 표시된다.
  - 인자는 상수뿐 아니라 변수도 가능하다.
    - > {0}, {1}처럼 인자의 순서를 나타내는 정수를 0부터 삽입하면 인자의 순서와 출력 순서를 조정할 수 있다.

```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> '3 + 4 = 7'
'3 + 4 = 7'
>>> str = '{} + {} = {}'.format(3, 4, 3 + 4)
>>> print(str)
3 + 4 = 7
>>> a, b = 10, 4
>>> print('{} * {} = {}'.format(a, b, a * b))
10 * 4 = 40
>>> |
```



## Chap 03 일상에서 활용되는 문자열과 논리 연산

### Section 2.5 출력을 정형화하는 함수 format()

- 문자열의 중괄호 {n:md}로 정수를 형식 유형으로 출력 처리
  - {1:5d}처럼 중괄호의 내부에서 인자의 순번 다음 콜론 이후에 5d, 10f 등의 출력 폭(5, 10등)과 출력 형식(d, f)등을 지정할 수 있다.
  - 10진수 정수는 d, 부동소수는 f로 표기한다.
  - 정수를 2진수와 8진수, 16진수로 출력하려면 각각 b, o, x로 형식유형을 지정해 출력할 수 있다.
  - 인자의 순번은 모두 적어야 하며, 빼려면 모두 빼야 한다.

```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> a, b = 10, 3
>>> print('{0:d} / {1:d} = {2:f}'.format(a, b, a / b))
10 / 3 = 3.333333
>>> a, b = 19, 8
>>> print('{:5b} / {:5o} = {:5d}'.format(a, b, a // b))
10011 / 10 = 2
>>> |
```

## Chap 03 일상에서 활용되는 문자열과 논리 연산

### Section 2.5 출력을 정형화하는 함수 format()

- 문자열의 중괄호 {n:mf}로 실수를 형식 유형 출력 처리
  - 형식 유형 f와 F는 모두 실수를 소수점 형태로 출력한다.
  - 폭을 10.3f로 지정하면 전체 폭은 10, 반올림해 소수점 이하 세 자리까지 출력한다.
  - 다만 F는 무한대를 의미하는 INF와 계산이 불가능한 수를 표현하는 NAN을 대문자로 표시한다.

```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (In
tel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> print('1234567890' * 2)
12345678901234567890
```

## Chap 03 일상에서 활용되는 문자열과 논리 연산

### Section 2.5 출력을 정형화하는 함수 format()

- 메소드 format()의 다양한 형식 지정

형식 유형	의미
d, n	10진수 정수이며, n은 국가에 맞는 구분자를 추가
c	유니코드 수에 대응하는 문자 출력
f, F	기본적으로 소수점 여섯 자리까지 실수로 출력하며, F인 경우는 'inf', 'nan' 표시를 대문자 'INF', 'NAN'로 표시
b	2진수 정수
o	8진수 정수
x, X	16진수 표현으로 a ~ f까지 소문자와 대문자로 각각 표시
e, E	지수 형식 3.141592e + 00으로 지수 표현이 각각 소문자 e와 대문자 E
g, G	실수를 일반적으로는 소수점 형식으로 출력하지만 커지면 지수승으로 표시
%	퍼센트 형식, 인자의 100배를 소수점으로 출력하고 맨 뒤에 %를 출력
s	문자열 형식이며, 기본적으로 왼쪽 정렬, 그러나 수 형식은 모두 기본이 가운데 정렬

## Chap 03 일상에서 활용되는 문자열과 논리 연산

### Section 2.6 C언어의 포매팅 스타일인 %d와 %f등으로 출력

- 전통적인 정형화 방식인 %d와 %f를 사용한 출력 처리
  - 프로그래밍 언어 C의 printf()에서 사용하는 형식 지정자 %d와 같은 스타일도 지원한다.
  - 문자열 중간에 변수나 상수를 출력하는 부분을 %d, %x, %o, %f, %c, %s등을 사용해 %로 이어지는 뒤의 상수나 변수를 순서대로 10진수, 16진수, 8진수, 소수점, 문자, 문자열로 출력한다.
  - %%는 %를 출력한다.

```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> '%d - %x = %o' % (30, 20, 30 - 20)
'30 - 14 = 12'
>>> print('%d ** %x = %o' % (3, 2, 3 ** 2))
3 ** 2 = 11
>>> print('%d%%' % 99)
99%
>>> |
```

## Chap 03 일상에서 활용되는 문자열과 논리 연산

- 중간 점검

3. 다음 출력값을 기술하시오.

- ① `print('python'.replace('p', 'P'))`     `#Python`
- ② `print('#'.join('C++'))`     `#C#++`
- ③ `print('python'.find('th'))`     `#2`
- ④ `print('python'.index('py'))`     `#0`
- ⑤ `print('정수, 실수, 문자열, 논리'.split(','))`     `#['정수', '실수', '문자열', '논리']`

4. 다음 문자열 출력값을 기술하시오.

- ① `print('p{}'.format('y', 'charm'))`     `#pycharm`
- ② `print('{0:d} {0:b} {0:o}'.format(7))`     `#7 111 7`
- ③ `print('{2} {1} {0}'.format(5, 20, 20, -5))`     `#15 20 5`
- ④ `print('{0:6:3f} {0:5.2f}'.format(31.456))`     `#31.456 31.46`
- ⑤ `print('%d %f' % (3.14, 3.14))`     `#3 3.140000`

## Chap 03 일상에서 활용되는 문자열과 논리 연산

### Section 3.1 논리 값과 논리 연산

- 논리 유형 bool과 함수 bool()
  - 파이썬은 논리 값으로 참과 거짓을 의미하는 True와 False를 키워드로 제공한다.
  - 파이썬은 이러한 논리 값의 자료형을 클래스 bool로 제공한다.
  - 정수의 0, 실수의 0.0, 빈 문자열''등은 False이며, 음수와 양수, 뭔가가 있는 'java'등의 문자열은 모두 True다.
  - 내장 함수 bool(인자)로 인자인 변수나 상수의 논리 값을 알 수 있다.
  - 논리 값 True와 False는 int(논리값) 함수에 의해 각각 1과 0으로 변환된다.

```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> print(True, False)
True False
>>> type(True)
<class 'bool'>
>>> bool(10), bool(3.14), bool('python')
(True, True, True)
>>> |
```

## Chap 03 일상에서 활용되는 문자열과 논리 연산

### Section 3.1 논리 값과 논리 연산

- 논리곱과 논리합 연산자 and와 or
  - 논리곱인 and와 &연산자는 두 항이 모두 참이어야 True다. 하나라도 거짓이면 False다.
  - 논리합이라는 or과 | 연산자는 두 항이 모두 거짓이어야 False다. 하나라도 참이면 True다.
- 배타적 논리합 연산자^와 not 연산자
  - 배타적 논리합 연산자인 ^은 두 항이 다르면 True, 같으면 False다.
  - 연산자 not은 뒤에 위치한 논리 값을 바꾼다.

```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> True & True, True and True
(True, True)
>>> True & False, True and True
(False, True)
>>> False | True, False or True
(True, True)
>>>
===== RESTART: Shell =====
>>> True ^ True
False
>>> True ^ False
True
>>> |
```

## Chap 03 일상에서 활용되는 문자열과 논리 연산

### Section 3.1 논리 값과 논리 연산

- 논리 연산 우선순위 not and or
  - 논리 연산은, not 연산, 논리곱 and, 논리합 or 순이다.

### Section 3.2 관계 연산

- 수학에서 자주 다뤘던 관계 연산자
  - 관계 연산자는 수나 문자열 등의 크기 비교에 사용되는 연산자로, 결과는 논리 값이다.
  - 문자열의 비교는 문자열을 구성하는 하나하나의 문자와 문자를 왼쪽부터 순서대로 비교한다. 비교 값은 유니코드다.

```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (In
tel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> not False and True
True
>>> (not False) and True
True
>>> not True or True and False
False
>>> (not True) or (True and False)
False
>>>
===== RESTART: Shell =====
>>> ord('a'), ord('A'), ord('#0'), ord('B')
(97, 65, 0, 66)
>>> 8 <= 2, 'a' <= 'aB'
(False, True)
>>> |
```



## Chap 03 일상에서 활용되는 문자열과 논리 연산

### Section 3.3 멤버십 연산 in

- 문자열에서 부분 문자열인지 검사
  - 파이썬 키워드인 in은 멤버의 소속을 알 수 있는 멤버십 연산으로, 결과는 True, False의 논리 값이다.
  - 키워드 in 뒤의 문자열에서 in 앞의 부분 문자열이 있는지 확인할 수 있다.
  - not in문은 부분 문자열이 아니면 True를 반환한다.

연산	결과
x in s	문자열 s에 부분 문자열 x가 있으면 True, 없으면 False
x not in s	문자열 s에 부분 문자열 x가 없으면 True, 있으면 False

## Chap 03 일상에서 활용되는 문자열과 논리 연산

### Section 3.4 비트 논리 연산

- 비트 논리곱  $\&$ , 비트 논리합  $|$ , 비트 배타적 논리합  $\wedge$ 
  - 비트 연산은 정수로 저장된 메모리에서 비트와 비트의 연산을 말한다.

m	n	논리합 $m \& n$	논리합 $m   n$	배타적 논리합 $m \wedge n$
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

## Chap 03 일상에서 활용되는 문자열과 논리 연산

### Section 3.4 비트 논리 연산

- 비트 배타적 논리합  $\wedge$ 을 사용해 암호와
  - 비트 배타적 논리합은 다음과 같은 특성이 있다.
    1.  $a \wedge a == 0, a \wedge 0 == a, a \wedge 1 == \sim a$
    2.  $a \wedge b == b \wedge a, (a \wedge b) \wedge c == a \wedge (b \wedge c)$
    3.  $(a \wedge b) \wedge b == a \wedge (b \wedge b) == a \wedge 0 == a$
  - 위 마지막 식을 활용하면  $\wedge$  연산으로 간단한 암호화에 활용할 수 있다.

## Chap 03 일상에서 활용되는 문자열과 논리 연산

### Section 3.5 비트 이동 연산

- 비트 이동 연산자 >>와 <<
  - 비트 이동 연산자 >>와 <<는 연산자의 방향인 오른쪽 또는 왼쪽으로 비트 단위로 뒤 피연산자인 지정된 횟수만큼 이동시키는 연산자다.
  - <<에 의해 생기는 오른쪽 빈 자리는 모두 0으로 채워지며, >>에 의한 빈 자리는 원래의 정수 부호에 따라 0이나 양수이면 0, 음수이면 1이 채워진다.
- 연산자 우선 순위 : 지수, 단항 산술, 비트, 관계 비교, 논리 연산 순서
  - 파이썬 연산자는 지수, 단항, 산술, 비트, 관계, 비교, 논리 연산 순이다.
  - 괄호를 사용해 명확한 우선 순위를 나타내는 것이 좋다.

```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (In
tel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> a = 0b00010111
>>> print('10진수 {0:3d}, 2진수 {0:08b}'.format(a))
10진수 23, 2진수 00010111
>>> print('10진수 {0:3d}, 2진수 {0:08b}'.format(a >> 1))
10진수 11, 2진수 00001011
>>> print('10진수 {0:3d}, 2진수 {0:08b}'.format(a // 2))
10진수 11, 2진수 00001011
>>> print('10진수 {0:3d}, 2진수 {0:08b}'.format(a >> 2))
10진수 5, 2진수 00000101
>>> print('10진수 {0:3d}, 2진수 {0:08b}'.format(a // 2 ** 2))
10진수 5, 2진수 00000101
>>> |
```

## Chap 03 일상에서 활용되는 문자열과 논리 연산

### Section 3.5 비트 이동 연산

- 연산자 우선 순위 : 지수, 단항 산술, 비트, 관계 비교, 논리 연산 순서
  - 파이썬 연산자는 지수, 단항, 산술, 비트, 관계, 비교, 논리 연산 순이다.
  - 괄호를 사용해 명확한 우선 순위를 나타내는 것이 좋다.

## Chap 03 일상에서 활용되는 문자열과 논리 연산

- 중간 점검

5. 다음 출력값을 기술하시오.

- ① `print('python' < 'python')` #False
- ② `Print(50 <= 50 < 60)` #True
- ③ `print(3 < 5 and 10 < 20)` #True
- ④ `print(3 >= 5 or not (10 < 20))` #False
- ⑤ `print('셋' not in '리스트 튜플 딕셔너리 셋 ')` #False

6. 다음 문자열 출력값을 기술하시오.

- ① `Print(0b100 & 0b011)` #0
- ② `print(5 | 2)` #7
- ③ `print(~5)` #-6
- ④ `print('{0:6:3f} {0:5.2f}'.format(31.456))` #3
- ⑤ `print('%d %f' % (3.14, 3.14))` #40

## Chap 04 일상생활과 비유되는 조건과 반복



## Chap 04 일상생활과 비유되는 조건과 반복

### Section 1.1 조건의 논리 값에 따른 선택 if

- 조건에 따른 선택을 결정하는 if 문
  - 조건에 따라 해야 할 일을 처리해야 하는 경우 if문을 사용
  - if문에서 논리 표현식 이후에는 반드시 콜론이 있어야 한다.
  - 콜론 이후 다음 줄부터 시작되는 블록은 반드시 들여쓰기를 해야 한다. 그렇지 않으면 오류가 발생한다.

```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (In
tel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> PM = 90 #미세먼지
>>> if 81 < PM:
    print('마스크를 착용합시다!')

마스크를 착용합시다!
>>> |
```



## Chap 04 일상생활과 비유되는 조건과 반복

### Section 1.2 조건에 따라 하나를 선택하는 if...else

- 논리 표현식 결과인 True와 False에 따라 나뉘는 if...else문
  - If문에서 논리 표현식 결과가 True이면 논리 표현식 콜론 이후 블록을 실행한다.
  - 논리 표현식의 결과가 False이면 else: 이후의 블록을 실행한다.

```
n = int(input('정수 입력 >> '))
if n % 2 == 0:
    print('%d는 짝수다.' % n)
else:
    print('%d는 홀수다.' % n)
```

Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (Intel)] on win32  
Type "help", "copyright", "credits" or "license()" for more information.  
>>>  
===== RESTART: C:/Users/win/AppData/Local/Programs/Python/Python38-32/z.py =====  
정수 입력 >> 11  
11는 홀수다.  
>>> |

## Chap 04 일상생활과 비유되는 조건과 반복

### Section 1.3 여러 조건 중에서 하나를 선택하는 구문 if...elif

- 다중 택일 결정 구조인 if...elif
  - 논리 표현식 1이 True이면 문장 1, 문장 2의 블록을 실행하고 종료된다.
  - 논리 표현식 1이 False여야 elif 논리 표현식 2로 진행되며, 그 이후도 마찬가지다.
  - elif는 필요한 만큼 늘릴 수 있으며, 마지막 else는 선택적으로 생략할 수 있다.
  - 블록은 단 하나만 선택돼 실행

```
point = 82
if 90 <= point:
    print('점수 {}, 성적 {}'.format(point, 'A'))
elif 80 <= point:
    print('점수 {}, 성적 {}'.format(point, 'B'))
elif 70 <= point:
    print('점수 {}, 성적 {}'.format(point, 'C'))
elif 60 <= point:
    print('점수 {}, 성적 {}'.format(point, 'D'))
else:
    print('점수 {}, 성적 {}'.format(point, 'F'))
```

```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (In
tel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Users/win/AppData/Local/Programs/Python/Python38-32/A.py =====
점수 82, 성적 B
>>> |
```

## Chap 04 일상생활과 비유되는 조건과 반복

### Section 1.4 중첩된 조건

- 일상 코딩 : 커피와 주스의 메뉴 선택

```
category = int(input('원하는 음료는? 1. 커피 2. 주스 '))  
  
if category == 1:  
    menu = int(input('번호 선택? 1. 아메리카노 2. 카페라테 3. 카푸치노'))  
    if menu == 1:  
        print('1. 아메리카노 선택')  
    elif menu == 2:  
        print('2. 카페라테 선택')  
    elif menu == 3:  
        print('3. 카푸치노 선택')  
else:  
    menu = int(input('번호 선택? 1. 키위주스 2. 토마토주스 3. 오렌지주스'))  
    if menu == 1:  
        print('1. 키위주스 선택')  
    elif menu == 2:  
        print('2. 토마토주스 선택')  
    elif menu == 3:  
        print('3. 오렌지주스 선택')
```

```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (In  
tel)] on win32  
Type "help", "copyright", "credits" or "license()" for more information.  
>>>  
===== RESTART: C:/Users/win/AppData/Local/Programs/Python/Python38-32/A.py =====  
원하는 음료는? 1. 커피 2. 주스 2  
번호 선택? 1. 키위주스 2. 토마토주스 3. 오렌지주스 3  
>>> |
```

## Chap 03 일상에서 활용되는 문자열과 논리 연산

- 중간 점검

1. 다음 출력값을 기술하시오.

- ① 

```
if 'o' in 'python':  
    print('o')
```

#o
- ② 

```
if not 27 % 3:  
    print('27은 3의 배수이다.')
```

#27은 3의 배수이다.

2. 다음 코드에서 오류를 수정하시오.

- ① 

```
a = 30  
if a <= 50  
    print(a / 2)
```

#a = 30  
if a <= 50  
 print(a / 2)
- ② 

```
speed = 60  
if 60 <= speed <= 100:  
    print('적정 속도')  
else if 100 < speed:  
    print('속도 초과')
```

#speed = 60  
if 60 <= speed <= 100:  
 print('적정 속도')  
elif 100 < speed:  
 print('속도 초과')

# Chap 04 일상생활과 비유되는 조건과 반복

## Section 2.1 시퀀스 내부 값으로 반복을 실행하는 for문

- 일상생활과 같은 반복의 실행

- 반복문을 사용하면 중복을 줄여 프로그램이 간결해지고 프로그램 절차를 효과적으로 수행할 수 있다.
- While문은 <반복조건>에 따라 반복을 결정하며, for문은 항목의 나열인 <시퀀스>의 구성 요소인 모든 항목이 순서대로 변수에 저장돼 반복을 수행한다.

- 정해져 있는 시퀀스의 항목 값으로 반복을 실행하는 for문

- 여러 개의 값을 갖는 시퀀스에서 변수에 하나의 값을 순서대로 할당한다.
- 할당된 변수값을 갖고 블록의 문장들을 순차적으로 실행한다.
- 반복 몸체인 문장1, 문장 2에서 변수를 사용할 수 있다.
- 시퀀스의 그 다음 값을 변수에 할당해 다시 반복 블록을 실행한다.
- 이러한 과정을 시퀀스의 마지막 항목까지 수행한다.
- 시퀀스의 마지막 항목까지 실행한 후 선택 사항인 else: 블록을 실행하고 반복을 종료한다.

```
for s in 'python':  
    print(s, ord(s))  
else:  
    print('반복 for문 완료')
```

```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (Intel)] on win32  
Type "help", "copyright", "credits" or "license()" for more information.  
>>>  
===== RESTART: C:/Users/win/AppData/Local/Programs/Python/Python38-32/A.py =====  
p 112  
y 121  
t 116  
h 104  
o 111  
n 110  
반복 for문 완료  
>>>|
```

## Chap 04 일상생활과 비유되는 조건과 반복

### Section 2.2 횟수를 정하지 않은 반복에 적합한 while 반복

- 반복 구조가 간단한 while 반복
  - while문은 for문에 비해 간결하며 반복 조건인 논리 표현식의 값에 따라 반복을 수행한다.
  - while문은 횟수를 정해놓지 않고 어떤 조건이 False가 될 때까지 반복을 수행하는 데 적합하다.
  - 논리 표현식이 True이면 반복 몸체인 문장 1, 문장 2를 실행한 후 다시 논리 표현식을 검사해 실행한다.
  - 논리 표현식이 False이면 반복 몸체를 실행하지 않고, 선택 사항인 else: 이후의 블록을 실행한 후 반복을 종료한다.

```
n = 1
while n <= 5 :
    print(n, end = ' ')
    n += 1
else :
    print("\n반복 while 종료: n => %d" %n)
```

```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (In
tel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Users/win/AppData/Local/Programs/Python/Python38-32/A.py =====
1 2 3 4 5
반복 while 종료: n => 6
>>> |
```

## Chap 04 일상생활과 비유되는 조건과 반복

- 중간 점검

3. 다음 출력값을 기술하시오.

① `for i in range(1, 10, 2):`  
`print(i, end = ' ')`      #1 3 5 7 9

① `n = 0`  
`while n <= 5:`  
`n += 1`  
`print(n, end = ' ')`      #1 2 3 4 5 6

4. 다음 코드의 결과값을 쓰시오.

① `for i in range(7):`  
`if i == 4:`  
`break`  
`print(i, end = ' ')`  
  
`print()`      #0 1 2 3

## Chap 04 일상생활과 비유되는 조건과 반복

### Section 3.1 임의의 수인 난수 발생과 반복에 활용

- 임의의 수를 발생하는 난수

- 파이썬에서는 모듈 random의 함수 randint(시작, 끝)을 사용해 정수 시작과 끝 수 사이에서 임의의 정수를 얻을 수 있다.
- random.randint(1, 3)는 import random으로 모듈 활용을 선언한 후 1, 2, 3 중 한 가지 수를 임의로 얻을 수 있다.

### Section 3.2 반복을 제어하는 break문과 continue문

- 반복을 종료하는 break문

- 반복 while의 논리 표현식이 True이면 무한히 반복된다. 이를 무한 반복이라고 한다.
- For나 while 반복 내부에서 문장 break는 else: 블록을 실행시키지 않고 반복을 무조건 종료한다.
- 반복 while이나 for문에서 break 문장은 반복을 종료하고 반복 문장 이후를 실행한다.
- 즉 break문은 특정한 조건에서 즉시 반복을 종료할 경우에 사용한다.

```
Python 3.8.3 (tags/v3.8.3:6f8b8f7, May 10 2020) on win32
Type "help", "copyright", "credits() or license() for more details
>>> while True:
>>>     menu = input('[0]을 종료 [1] 계속 ? ')
>>>     if menu == '0':
>>>         break
>>>     print('종료')
>>>
=====
>>> import random
>>> random.randint(1, 3)
1
>>> random.randint(1, 3)
2
>>>
=====
>>>
===== RESTART: C:/Users/win/A
>>>
[0] 종료 [1] 계속 ? 1
[0] 종료 [1] 계속 ? 1
[0] 종료 [1] 계속 ? 0
>>>
```



## Chap 04 일상생활과 비유되는 조건과 반복

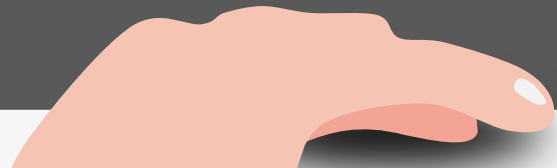
### Section 3.2 반복을 제어하는 break문과 continue문

- Continue 이후의 반복 몸체를 실행하지 않고 다음 반복을 계속 실행
  - 반복 for와 while문 내부에서 continue 문장은 이후의 반복 몸체를 실행하지 않고 다음 반복을 위해 논리 조건을 수행한다.
  - 다음 코드는 continue 문장을 사용해 단어 'python'에서 영어의 모음인 'aeiou'를 제외한 자음을 출력하는 코드다.
  - 영어 문자가 모음에 해당하는 조건 검사인 if s in 'aeiou'를 통과하면 continue가 실행돼 뒤의 print()가 실행되지 않는다.

```
for s in 'python':  
    if s in 'aeiou':  
        continue  
    print(s, end = ' ')  
else:  
    print()
```

```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (In  
tel)] on win32  
Type "help", "copyright", "credits" or "license()" for more information.  
>>>  
===== RESTART: C:/Users/win/AppData/Local/Programs/Python/Python38-32/A.py =====  
p y t h n  
>>> |
```

## Chap 05 항목의 나열인 리스트와 튜플



# Chap 05 항목의 나열인 리스트와 튜플

## Section 1.1 리스트의 개념과 생성

- 관련된 나열 항목을 관리하는 리스트

- 프로그래밍 언어는 일반적으로 여러 항목을 한 번에 관리할 수 있는 자료형을 지원한다.
- 파이썬도 여러 항목을 하나의 단위로 묶어 손쉽게 사용하는 복합 자료형을 여러 개 제공하는데 그 중 대표적인 것이 바로 리스트이다.

- 빈 리스트의 생성과 항목 추가

- 빈 대괄호로 빈 리스트를 만들 수 있다.
- 출력하면 빈 리스트 표기인 []로 표시된다.

```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> menu = ['COFFEE', 'BEVERAGE', 'ADE']
>>> menu
['COFFEE', 'BEVERAGE', 'ADE']
>>> print(menu)
['COFFEE', 'BEVERAGE', 'ADE']
>>>
===== RESTART: Shell =====
>>> pl = []
>>> print(pl)
[]
>>> pl = list()
>>> pl.append('C++')
>>> pl.append('java')
>>> print(pl)
['C+', 'java']
...!
```

## Chap 05 항목의 나열인 리스트와 튜플

### Section 1.1 리스트의 개념과 생성

- 문자열의 문자로 구성되는 리스트와 리스트의 항목 수를 반환하는 함수 len()
  - 문자열 시퀀스는 함수 list('문자열')로, 문자열의 항목인 문자로 구성되는 리스트로 변환된다.

```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> py = ['p', 'y', 't', 'h', 'o', 'n']
>>> py
['p', 'y', 't', 'h', 'o', 'n']
>>> py = list('python')
>>> py
['p', 'y', 't', 'h', 'o', 'n']
>>> len(py)
6
.
```

## Chap 05 항목의 나열인 리스트와 튜플

### Section 1.2 리스트의 항목 참조

- 문자열 시퀀스와 같이 첨자로 리스트의 항목 참조
  - 리스트에서 첨자(index)를 사용해 항목 하나하나를 참조할 수 있다.
  - 첨자는 첫 요소가 0부터 시작하며, 순차적으로 1씩 증가한다.
  - 마지막 요소의 첨자는 역순으로 -1부터 시작하며, 반대로 1씩 감소한다.
  - 그러므로 첨자의 범위는 0에서 [len(시퀀스)-1]까지, -1에서 [-len(시퀀스)]까지 가능하다.

```
>>> py = list('python')
>>> print(py[0], py[5])
p n
>>> print(py[-3], py[-1])
h n
>>> print(py[-len(py)], py[len(py)-1])
p n
>>>
```

# Chap 05 항목의 나열인 리스트와 튜플

## Section 1.3 리스트의 항목 수정

- 리스트의 메소드 count()와 index()
  - 리스트의 메소드 count(값)는 값을 갖는 항목의 수, index(값)는 인자인 값의 항목이 위치한 첨자를 반환한다.
  - 동일한 값이 여러 개이면 첫 번째로 나타난 위치의 첨자다.
- 리스트의 첨자로 항목 수정
  - 리스트의 첨자를 이용한 항목을 대입 연산자의 오른쪽에 위치시켜 리스트의 항목을 수정할 수 있다.
  - 첨자는 유효한 것이어야 하므로 적어도 하나 이상의 항목이 있는 경우에 수정할 수 있다.

```
===== RESTART: Shell =====
>>> top = ['BTS', '볼빨간사춘기', 'BTS', '블랙핑크', '태연', 'BTS']
>>> top.count('BTS')
3
>>> top.index('볼빨간사춘기')
1
>>> top.index('BTS')
0
>>>

===== RESTART: Shell =====
>>> top = ['BTS', '볼빨간사춘기', 'BTS', '블랙핑크', '태연', 'BTS']
>>> top[1] = '장범준'
>>> top[3] = '잔나비'
>>> print(top)
['BTS', '장범준', 'BTS', '잔나비', '태연', 'BTS']
>>> |
```

## Chap 05 항목의 나열인 리스트와 튜플

### Section 1.4 리스트의 항목으로 리스트 구성

- 리스트의 항목으로 리스트 구성
  - 리스트 내부에 다시 리스트가 항목으로 올 수 있다.

```
>>> animal = [['사자', '코끼리', '호랑이'], '조류', '어류']
>>> print(animal)
[['사자', '코끼리', '호랑이'], '조류', '어류']
>>> print(animal[0])
['사자', '코끼리', '호랑이']
>>> print(animal[0][1])
코끼리
```

## Chap 05 항목의 나열인 리스트와 튜플

- 중간 점검

1. 다음 출력값을 기술하시오.

① 

```
java = list('java')  
print(type(java))  
print(len(java))
```

<class 'list'>  
4

② 

```
numstr = '01234567890'  
string = 'hellopython'  
print(string[2:11:3])
```

lph

2. 다음 코드에서 오류를 수정하시오.

① 

```
lst = [1, 3, 7, 9]  
lst.index(5)
```

1, 3, 7, 9 중 1개 입력  
>>> lst.index(5)  
Traceback (most recent call last):  
File "<stdin>", line 1, in <module>  
ValueError : 5 is not in list

② 

```
Llst = [10, 30, 40, 50]  
lst[:3] = 100
```

[100]과 같이 리스트로 입력해야 함.  
>>> lst[:3] = 100  
Traceback (most recent call last) :  
File "<stdin>", line 1, in <module>  
TypeError : can only assign an iterable



## Chap 05 항목의 나열인 리스트와 튜플

### Section 2.1 리스트의 부분 참조인 슬라이싱

- 첨자 3개로 리스트 일부분을 참조하는 슬라이싱
  - 0에서 시작하는 오름차순(수가 차례로 늘어가는 것).
  - 마지막 요소 -1에서 시작하는 내림차순(수가 차례로 줄어가는 것)의 첨자도 가능하며,
  - 다소 복잡하지만 두 순차의 첨자를 함께 사용 가능

```
>>> alp = list('abcdefghij')
>>> print(alp)
['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']
>>> len(alp)
10
>>> print(alp[:])
['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']
>>> print(alp[::])
['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']
>>> print(alp[::-1])
['j', 'i', 'h', 'g', 'f', 'e', 'd', 'c', 'b', 'a']
>>> print(alp[:])
['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']
>>> |
```

## Chap 05 항목의 나열인 리스트와 튜플

### Section 2.1 리스트의 부분 참조인 슬라이싱

- 0부터 시작되는 순차 첨자
  - 알파벳 리스트 alp에서 alp[1:5]는 첨자 1에서 4까지인 b에서 e까지 4개로 구성되는 부분 문자열이다.
  - 슬라이싱 alp[1:10:2]는 b에서 j까지 2씩 증가하는 항목으로 구성된 리스트다.
- -1부터 시작되는 역순 첨자
  - 슬라이싱 alp[-2:-9:-1]는 역순으로 첨자 -2에서 -8까지인 i에서부터 c까지 구성되는 부분 리스트이고, alp[-3::-2]는 h에서 b까지씩 2씩 감소하는 역순의 항목으로 구성된 리스트다.

```
>>> alp = list('abcdefghij')
>>> print(alp[1:5])
['b', 'c', 'd', 'e']
>>> print(alp[1:10:2])
['b', 'd', 'f', 'h', 'j']
>>>
===== RESTART: Shell =====
>>> alp = list('abcdefghij')
>>> print(alp[-2:-9:-1])
['i', 'h', 'g', 'f', 'e', 'd', 'c']
>>> print(alp[-3::-2])
['h', 'f', 'd', 'b']
>>> |
```

## Chap 05 항목의 나열인 리스트와 튜플

### Section 2.1 리스트의 부분 참조인 슬라이싱

- 0 또는 양수인 정순 첨자와 역순인 음수 첨자를 함께 사용하는 슬라이싱
  - 알파벳 리스트 alp에서 첨자 1은 정순으로 두 번째인 b이다.
  - 그러므로 alp[1:-1]은 b에서 i까지의 정순으로 구성되는 슬라이싱이다.
  - alp[-1:1:-1]은 역순으로 j에서부터 c까지의 구성되는 부분 리스트이다.
  - alp[-2:2:-2]는 i에서 e까지 2씩 감소하는 역순의 항목으로 구성된 리스트다.

```
>>> alp = list('abcdefghij')
>>> print(alp[1:-1])
['b', 'c', 'd', 'e', 'f', 'g', 'h', 'i']
>>> print(alp[-1:1:-1])
['j', 'i', 'h', 'g', 'f', 'e', 'd', 'c']
>>> print(alp[-2:2:-2])
['i', 'g', 'e']
>>> |
```

## Chap 05 항목의 나열인 리스트와 튜플

### Section 2.2 리스트의 부분 수정

- 리스트의 슬라이스로 리스트의 일부분을 수정
  - 리스트의 일부분을 다른 리스트로 수정하려면 슬라이스 방식에 대입해야 한다.
- 리스트 슬라이스에 슬라이스 대입
  - 리스트의 슬라이스에 동일한 리스트의 슬라이스를 대입해도 아무런 문제가 없다.

```
>>> sports = ['풋살', '족구', '비치사커', '야구', '농구', '배구']
>>> sports[0:3] = ['축구']
>>> print(sports)
['축구', '야구', '농구', '배구']
>>>
===== RESTART: Shell =====
>>> sports = ['풋살', '족구', '비치사커', '야구', '농구', '배구']
>>> sports[1:3] = sports[4:6]
>>> print(sports)
['풋살', '농구', '배구', '야구', '농구', '배구']
>>> sports = ['풋살', '족구', '비치사커', '야구', '농구', '배구']
>>> others = ['축구', '미식축구', '골프']
>>> sports[1:4] = others[:2]
>>> print(sports)
['풋살', '축구', '미식축구', '농구', '배구']
>>> |
```

## Chap 05 항목의 나열인 리스트와 튜플

### Section 2.3 리스트의 항목 삽입과 삭제

- 리스트 메소드 insert(첨자, 항목)으로 삽입
  - 리스트의 첨자 위치에 항목을 삽입하려면 리스트.insert(첨자, 항목)을 이용한다.
  - 삽입되는 항목은 무엇이든 가능하며, 빈 리스트에도 삽입할 수 있다.
- 리스트 메소드 remove(항목)과 pop(첨자), pop()로 항목 삭제

- 리스트에서 하나의 항목을 삭제하려면 메소드 remove(값) 또는 pop(첨자), pop()를 사용한다.
- 메소드 remove(값)은 리스트에서 지정된 값의 항목을 삭제한다.

```
>>> kpop = []
>>> kpop.insert(0, '블랙핑크')
>>> kpop.insert(0, 'BTS')
>>> kpop
['BTS', '블랙핑크']
>>> kpop.insert(1, '장범준')
>>> kpop
['BTS', '장범준', '블랙핑크']
>>> |
```

```
>>> kpop = ['BTS', '장범준', '블랙핑크', '잔나비']
>>> kpop.remove('장범준')
>>> print(kpop)
['BTS', '블랙핑크', '잔나비']
>>> kpop.remove('장준')
Traceback (most recent call last):
  File "<pyshell#57>", line 1, in <module>
    kpop.remove('장준')
ValueError: list.remove(x): x not in list
>>> if '장준' in kpop:
>>>     kpop.remove('장준')
```

```
>>> print(kpop)
['BTS', '블랙핑크', '잔나비']
>>> kpop = ['BTS', '장범준', '블랙핑크', '잔나비']
>>> print(kpop.pop(1))
장범준
>>> print(kpop.pop())
잔나비
>>> print(kpop)
['BTS', '블랙핑크']
>>>
```

## Chap 05 항목의 나열인 리스트와 튜플

### Section 2.3 리스트의 항목 삽입과 삭제

- 문장 `del()`에 의한 항목이나 변수의 삭제
  - 문장 `del`은 뒤에 위치한 변수나 항목을 삭제한다.
- 리스트의 모든 항목을 제거해 빈 리스트로 만드는 메소드 `clear()`
  - `kpop.clear()`로 리스트 `kpop`의 모든 항목이 삭제되고, 이후 리스트 `kpop`은 빈 리스트가 된다.

```
>>> kpop = ['BTS', '장범준', '블랙핑크', '잔나비']
>>> del kpop[0]
>>> print(kpop)
['장범준', '블랙핑크', '잔나비']
>>> kpop = ['BTS', '장범준', '블랙핑크', '잔나비']
>>> del kpop[0:2]
>>> print(kpop)
['블랙핑크', '잔나비']
>>> kpop = ['BTS', '장범준', '블랙핑크', '잔나비']
>>> del kpop
>>> print(kpop)
Traceback (most recent call last):
  File "<pyshell#74>", line 1, in <module>
    print(kpop)
NameError: name 'kpop' is not defined
>>>
===== RESTART: Shell =====
>>> kpop = ['BTS', '장범준', '블랙핑크', '잔나비']
>>> kpop.clear()
>>> print(kpop)
[]
>>> |
```

## Chap 05 항목의 나열인 리스트와 튜플

### Section 2.4 리스트의 추가, 연결과 반복

- 리스트에 리스트를 추가하는 메소드 extend()
  - 리스트 메소드 리스트.extend(list)는 리스트에 인자인 list를 가장 뒤에 추가한다.
- 리스트를 연결하는 + 연산자
  - 더하기 연산자인 +는 리스트와 리스트를 연결해준다.

```
>>> day = ['월', '화', '수']
>>> day2 = ['목', '금', '토', '일']
>>> day.extend(day2)
>>> print(day)
['월', '화', '수', '목', '금', '토', '일']
>>> korean = ['불고기', '설렁탕']
>>> chinese = ['탕수육', '기스면']
>>> food = korean + chinese
>>> print(food)
['불고기', '설렁탕', '탕수육', '기스면']
>>> days = ['월', '화']
>>> print(days * 3)
['월', '화', '월', '화', '월', '화']
>>> |
```

# Chap 05 항목의 나열인 리스트와 튜플

## Section 2.5 리스트 항목의 순서와 정렬

- 리스트 항목 순서를 뒤집는 메소드 reverse()
  - 리스트 메소드 reverse()는 항목 순서를 반대로 뒤집는다.
- 리스트 항목 순서를 정렬하는 메소드 sort()
  - 나열된 여러 항목을 기준에 따라 순서대로 재배열하는 것이다.
  - 정렬의 방식에는 작은 것부터 배열하는 오름차순과 큰 것부터 배열하는 내림차순이 있다.
  - sort()는 리스트 항목의 순서를 오름차순으로 정렬한다.

```
>>> one = '잣밤배굴감'
>>> wlist = list(one)
>>> print(wlist)
['잣', '밤', '배', '굴', '감']
>>> wlist.reverse()
>>> print(wlist)
['감', '굴', '배', '밤', '잣']
>>>
===== RESTART: Shell =====
>>> one = '잣밤배굴감'
>>> wlist = list(one)
>>> wlist.sort()
>>> print(wlist)
['감', '굴', '밤', '배', '잣']
>>> wlist.sort(reverse = True)
>>> print(wlist)
['잣', '배', '밤', '굴', '감']
>>> |
```



## Chap 05 항목의 나열인 리스트와 튜플

### Section 2.5 리스트 항목의 순서와 정렬

- 리스트 항목의 순서를 정렬한 리스트를 반환하는 내장 함수 `sorted()`
  - 내장 함수 `sorted(리스트)`는 리스트의 항목 순서를 오름차순으로 정렬한 새로운 리스트를 반환한다.
  - 그러므로 반환 값을 새 변수에 대입해 사용할 수 있다.
  - 원래의 리스트 자체는 변화되지 않는다는 점에 주의

```
>>> fruit = ['사과', '귤', '복숭아', '파인애플']
>>> s_fruit = sorted(fruit)
>>> print(s_fruit)
['귤', '복숭아', '사과', '파인애플']
>>> print(fruit)
['사과', '귤', '복숭아', '파인애플']
>>> rs_fruit = sorted(fruit, reverse=True)
>>> print(rs_fruit)
['파인애플', '사과', '복숭아', '귤']
>>> |
```

## Chap 05 항목의 나열인 리스트와 튜플

### Section 2.6 리스트 컴프리헨션

- 조건을 만족하는 항목으로 리스트를 간결히 생성하는 컴프리헨션
  - 리스트 컴프리헨션은 리스트를 만드는 간결한 방법을 제공한다.
  - 리스트 컴프리헨션은 리스트를 만들므로 대괄호로 감싸고, 구성 항목인 `i`를 가장 앞에 배치하며, `for`문이 오는데 `range(2, 11, 2)` 뒤에 콜론이 빠진다.
  - 리스트 컴프리헨션을 사용하면 한 리스트의 모든 항목 각각에 대해 어떤 조건을 적용한 후 그 반환값을 항목으로 갖는 다른 리스트를 쉽게 만들 수 있다.
  - 컴프리헨션은 함축, 축약, 내포, 내장 등으로도 불린다.

```
>>> even = []
>>> for i in range(2, 11, 2):
>>>     even.append(i)

>>> print(even)
[2, 4, 6, 8, 10]
>>> |
```

## Chap 05 항목의 나열인 리스트와 튜플

### Section 2.6 리스트 컴프리헨션

- 조건이 있는 컴프리헨션
  - 이번에는 조건이 있는 컴프리헨션이다. 다음 리스트 odd는 1에서 9까지의 홀수가 저장된다.
  - 변수 i의 조건을 뒤에 붙이면 되고, 조건에서 콜론이 빠진다.

```
>>> odd = []
>>> for i in range(10):
>>>     if i%2 == 1:
>>>         odd.append(i)

>>> print(odd)
[1, 3, 5, 7, 9]
>>> odd = [i for i in range(10) if i%2 == 1]
>>> print(odd)
[1, 3, 5, 7, 9]
>>> |
```

## Chap 05 항목의 나열인 리스트와 튜플

### Section 2.6 리스트 컴프리헨션

- 0에서 9까지의 리스트와 10 이하의 홀수의 제곱을 구하는 리스트 컴프리헨션
  - 일반 방식과 축약된 리스트 컴프리헨션 방식으로 리스트를 만들어 각각 출력하는 프로그램을 작성해 보자.

```
>>> a = []
>>> for i in range(10):
>>>     a.append(i)

>>> print(a)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>>
>>> s = []
>>> for i in range(10):
>>>     if i%2 == 1:
>>>         s.append(i**2)

>>> print(s)
[1, 9, 25, 49, 81]
>>> |
```

# Chap 05 항목의 나열인 리스트와 튜플

## Section 2.7 리스트 대입과 복사

- 리스트 대입에 의한 동일 리스트의 공유
  - 리스트의 대입에는 주의가 필요하다. 리스트에서 대입 연산자 =은 얇은 복사라고 해서 대입되는 변수가 동일한 시퀀스를 가리킨다.
- 리스트의 깊은 복사에 의한 대입으로 새로운 리스트의 생성
  - 리스트에서 완전히 새로운 리스트를 만들어 복사하려면 슬라이스[:]나 copy() 또는 list() 함수를 사용해야 한다.
- 변수의 동일 객체 여부를 검사하는 is
  - 문장 is는 피연산자인 변수 2개가 동일한 메모리를 공유하는지 검사한다.
  - 그러므로 같으면 True, 다르면 False를 반환한다.

```
>>> f1 = ['사과', '귤', '복숭아', '파인애플']
>>> f2 = f1
>>> f2.pop()
'파인애플'
>>> print(f1)
['사과', '귤', '복숭아']
>>> print(f2)
['사과', '귤', '복숭아']
>>> |
```

```
>>> f1 = ['사과', '귤', '복숭아', '파인애플']
>>> f2 = f1[:]
>>> f2.pop()
'귤'
>>> print(f1, f2)
['사과', '귤', '복숭아', '파인애플'] ['사과', '복숭아', '파인애플']
>>> f3 = f1.copy()
>>> f3.pop()
'파인애플'
>>> print(f1, f3)
['사과', '귤', '복숭아', '파인애플'] ['사과', '귤', '복숭아']
>>> f4 = list(f1)
>>> f4.append('감')
>>> print(f1, f4)
['사과', '귤', '복숭아', '파인애플'] ['사과', '귤', '복숭아', '파인애플', '감']
>>> |
```

```
>>> f1 = ['사과', '귤', '복숭아', '파인애플']
>>> f2 = f1
>>> print(f1 is f2)
True
>>> f3 = f1[:]
>>> print(f1 is f3)
False
>>> |
```

## Chap 05 항목의 나열인 리스트와 튜플

- 중간 점검

3. 다음 출력값을 기술하시오.

① `lst = [1, 5]`      `[3, 10]`  
`lst[0] = 3`  
`lst.append(7)`  
`lst[1:3] = [10]`  
`print(lst)`

② `lst = [1, 5, 1, 7, 1]`      `[1, 50, 5, 1, 70, 80]`  
`lst[lst.count(1)] = 70`  
`lst[len(lst)-1] = 80`  
`lst.insert(1, 50)`  
`print(lst)`

4. 다음 코드의 결과값을 쓰시오.

① `squares = [x **2 for x in`      `[1, 9, 25, 49, 81]`  
`range(1, 11, 2)]`  
`print(squares)`

## Chap 05 항목의 나열인 리스트와 튜플

### Section 3.1 괄호로 정의하는 시퀀스 튜플

- 수정할 수 없는 항목의 나열인 튜플

- 튜플은 문자열, 리스트와 같은 항목의 나열인 시퀀스다.
- 튜플은 리스트와 달리 항목의 순서나 내용의 수정이 불가능하다.
- 각각의 항목은 정수, 실수, 문자열, 리스트, 튜플 등 제한이 없다.

```
>>> singer = ['BTS', '불뽀간사춘기', 'BTS', '블랙핑크', '태연']
>>> credit = ([2020, 1, 18], [2020, 2, 17])
>>> space = '밤', '낮', '해', '달'
```

- 튜플의 생성

- 빈 튜플은 ()로 만든다.
- 튜플의 자료형 이름은 tuple이다.
- 빈 튜플은 함수 tuple()로도 생성할 수 있다.

```
>>> empty1 = ()
>>> type(empty1)
<class 'tuple'>
>>> print(empty1)
()
>>> empty2 = tuple()
>>> print(empty1)
()
>>> credit = (16, 17)
>>> credit = 16, 17
>>> print(credit)
(16, 17)
>>> inta = 1
>>> tupa = 1,
>>> print(tupa)
(1,)
```

## Chap 05 항목의 나열인 리스트와 튜플

### Section 3.1 괄호로 정의하는 시퀀스 튜플

- 튜플 항목 참조와 출력
  - 튜플은 수정이 불가능하므로 첨자와 슬라이스로 수정할 수 없다.

```
>>> nation = '대한민국', '뉴질랜드', '캐나다'
>>> city = ('부산', '웰링턴', '몬트리올')
>>> nation[1]
'뉴질랜드'
>>> city[1:3]
('웰링턴', '몬트리올')
>>> |
```



## Chap 05 항목의 나열인 리스트와 튜플

### Section 3.2 튜플 연결과 반복, 정렬과 삭제

- 튜플연결 + 연산자의 반복 \* 연산자
  - 리스트와 같이 +와 \*는 각각 튜플을 연결하고 항목이 횟수만큼 반복된 튜플을 반환한다.
- 튜플항목의 순서를 정렬한 리스트를 반환하는 내장 함수 sorted()
  - 내장 함수 sorted(튜플) 항목의 순서를 오름차순으로 정렬한 새로운 리스트를 반환한다.
  - 반환 값을 리스트 변수에 대입해 사용할 수 있다.

```
>>> kpop = ('BTS', '블랙핑크')
>>> num = (7, 4)
>>> print(kpop + num)
('BTS', '블랙핑크', 7, 4)
>>> days = ('1학기', '2학기')
>>> print(days * 4)
('1학기', '2학기', '1학기', '2학기', '1학기', '2학기', '1학기', '2학기')
>>> |

>>> fruit = ('사과', '귤', '복숭아', '파인애플')
>>> tup = sorted(fruit)
>>> print(type(tup), tup)
<class 'list'> ['귤', '복숭아', '사과', '파인애플']
>>> print(sorted(fruit, reverse=True))
['파인애플', '사과', '복숭아', '귤']
>>> print(fruit)
('사과', '귤', '복숭아', '파인애플')
>>> |
```

## Chap 05 항목의 나열인 리스트와 튜플

### Section 3.2 튜플 연결과 반복, 정렬과 삭제

- 문장 del()에 의한 튜플 변수 자체의 제거
  - 문장 del에서 변수 kpop을 지정하면 변수 자체가 사라진다.
  - 삭제 이후에는 변수 자체가 메모리에서 제거되므로 참조하면 오류가 발생한다.

```
>>> kpop = ('BTS', '장범준', '블랙핑크', '잔나비')
>>> del kpop
>>> print(kpop)
Traceback (most recent call last):
  File "<pyshell#213>", line 1, in <module>
    print(kpop)
NameError: name 'kpop' is not defined
>>> |
```

## Chap 05 항목의 나열인 리스트와 튜플

- 중간 점검

5. 다음 출력값을 기술하시오.

- ① `a = 1`      `<class 'int'>`  
`print(type(a))`
- ② `b = 1,`      `<class 'tuple'>`  
`print(type(a))`

6. 다음 코드의 결과값을 쓰시오.

- ① `print(sorted((5, 4, 1)))`      `[1, 4, 5]`
- ② `print(sorted((5, 4, 1),  
reverse = True))`      `[5, 4, 1]`

## Chap 06 키와 값의 나열인 딕셔너리와 중복을 불허하는 집합



# Chap 06 키와 값의 나열인 딕셔너리와 중복을 불허하는 집합

## Section 1.1 딕셔너리의 개념과 생성

- 키와 값의 쌍을 항목으로 관리하는 딕셔너리
  - 딕셔너리는 키와 값의 쌍인 항목을 나열한 시퀀스다.
  - 딕셔너리는 콤마로 구분된 항목들의 리스트로 표현된다.
  - 각각의 항목은 키 : 값과 같이 키와 값을 콜론으로 구분하고 전체는 중괄호 {}로 묶는다.
  - 딕셔너리의 항목 순서는 의미가 없으며, 키는 중복될 수 없다.
  - 키는 수정될 수 없지만, 값은 수정될 수 있다.
  - 값은 키로 참조된다.

```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> groupnumber = {'엑소': 9, '트와이스': 9, '블랙핑크': 4, '방탄소년단': 7}
>>> print(groupnumber)
{'엑소': 9, '트와이스': 9, '블랙핑크': 4, '방탄소년단': 7}
>>> |
```

# Chap 06 키와 값의 나열인 딕셔너리와 중복을 불허하는 집합

## Section 1.1 딕셔너리의 개념과 생성

- 빈 딕셔너리의 생성과 항목 추가
  - 빈 중괄호로 빈 딕셔너리를 만들 수 있다.
  - 인자가 없는 내장 함수 dict() 호출로도 빈 딕셔너리를 생성할 수 있다.
  - 딕셔너리 항목 값으로 리스트나 튜플 등이 가능하다.

```
lect = {}  
print(lect)  
  
print("\n")  
  
lect = dict()  
lect['강좌명'] = '파이썬 기초';  
print(lect)
```

```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (Intel)] on win32  
Type "help", "copyright", "credits" or "license()" for more information.  
>>>  
===== RESTART: C:/Users/win/AppData/Local/Programs/Python/Python38-32/aa.py =====  
{}  
  
{'강좌명': '파이썬 기초'}  
>>> |
```

## Chap 06 키와 값의 나열인 딕셔너리와 중복을 불허하는 집합

### Section 1.2 다양한 인자의 함수 dict()로 생성하는 딕셔너리

- 리스트 또는 튜플로 구성된 키-값을 인자로 사용
  - 내장 함수 dict() 함수에서 인자로 리스트나 튜플 1개를 사용해 딕셔너리를 만들 수 있다.
  - 함수 dict()의 리스트나 튜플 내부에서 일련의 키-값 쌍으로 [키, 값] 리스트 형식과 (키, 값) 튜플 형식을 모두 사용할 수 있다.
- 키가 문자열이면 키 = 값 항목의 나열로도 딕셔너리 생성
  - 키가 단순 문자열이면 간단히 월 = 'monday'처럼 키 = 값 항목 나열로도 지정할 수 있다.

```
day = dict([])
print(day)
day = dict({})
print(day)

print("\n")

day = dict([['월', 'monday'], ['화', 'tuesday'], ['수', 'wednesday']])
print(day)

print("\n")

day = dict(월 = 'monday', 화 = 'tuesday', 수 = 'wednesday')
print(day)
```

```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Users/win/AppData/Local/Programs/Python/Python38-32/aa.py =====
>>> {}
{'월': 'monday', '화': 'tuesday', '수': 'wednesday'}

{'월': 'monday', '화': 'tuesday', '수': 'wednesday'}
>>> |
```

## Chap 06 키와 값의 나열인 딕셔너리와 중복을 불허하는 집합

### Section 1.3 딕셔너리 키는 수정 불가능한 객체로 사용

- 딕셔너리의 키로 정수, 실수 등 사용 가능
  - 딕셔너리의 키는 수정 불가능한 객체는 모두 가능하다. 따라서 정수는 물론 실수도 가능하다.
  - 이전에 없던 키는 새로운 항목으로 삽입된다.
  - 새로운 키로 대입하면 항상 새로운 키 - 값의 항목이 삽입된다.
  - 이미 있는 키로 항목을 참조하면 값이 반환된다.
  - 키로 정수도 가능하다.

```
real = {3.14 : '원주율'}
real[2.71] = '자연수'
print(real)

real[2.72] = '자연수'
print(real)

real[1.61] = '황금비'
print(real)

print(real[3.14])
print(real[2.71])
print(real[1.61])
```

Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (Intel)] on win32  
Type "help", "copyright", "credits" or "license()" for more information.  
>>>  
==== RESTART: C:/Users/win/AppData/Local/Programs/Python/Python38-32/aa.py ====  
{3.14: '원주율', 2.71: '자연수'}  
{3.14: '원주율', 2.71: '자연수', 2.72: '자연수'}  
{3.14: '원주율', 2.71: '자연수', 2.72: '자연수', 1.61: '황금비'}  
원주율  
자연수  
황금비  
>>> |



## Chap 06 키와 값의 나열인 딕셔너리와 중복을 불허하는 집합

### Section 1.3 딕셔너리 키는 수정 불가능한 객체로 사용

- 튜플은 키로 간으하지만 리스트는 키로 사용 불가능
  - 수정 불가능한 튜플은 딕셔너리의 키로 사용될 수 있다.
  - 수정 가능한 리스트는 키로 사용할 수 없다.

### Section 1.4 딕셔너리 항목의 순회

- 딕셔너리 메소드 keys()
  - 딕셔너리 메소드 keys()는 키로만 구성된 리스트를 반환한다.
  - for문에서 시퀀스 위치에 메소드 keys()를 사용하면 딕셔너리의 모든 항목을 참조하는 구문을 사용할 수 있다.

```
city = {(37.33, 126.58): '서울', (35.06, 129.03): '부산'}
print(city)

print("\n")

day = dict(월 = 'monday', 화 = 'tuesday', 수 = 'wednesday', 목 = 'thursday')
print(day)
print(day.keys())
```

```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (In
tel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
==== RESTART: C:/Users/win/AppData/Local/Programs/Python/Python38-32/aa.py ====
{((37.33, 126.58): '서울', (35.06, 129.03): '부산')}

{'월': 'monday', '화': 'tuesday', '수': 'wednesday', '목': 'thursday'}
dict_keys(['월', '화', '수', '목'])
>>>
```

# Chap 06 키와 값의 나열인 딕셔너리와 중복을 불허하는 집합

## Section 1.4 딕셔너리 항목의 순회

- 딕셔너리 메소드 items()
  - 딕셔너리 메소드 items()는 (키, 값) 쌍의 튜플이 들어 있는 리스트를 반환한다.
  - 각 튜플의 첫 번째 항목은 키, 두 번째 항목은 키 값이다.
  - for문에서 변수 위치에 키 값을 저장할 2개의 변수와 시퀀스 위치에 메소드 items()를 사용하면 딕셔너리의 모든 항목을 참조하는 간단한 구문을 사용할 수 있다.
- 딕셔너리 메소드 values()
  - 딕셔너리 메소드 values()는 값으로 구성된 리스트를 반환한다.

```
day = dict(월 = 'monday', 화 = 'tuesday', 수 = 'wednesday', 목 = 'thursday')
print(day.items())
for key, value in day.items():
    print('%s요일 %s' % (key, value))

print("#n")
day = dict(월 = 'monday', 화 = 'tuesday', 수 = 'wednesday', 목 = 'thursday')
print(day.values())
```

```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Users/win/AppData/Local/Programs/Python/Python38-32/aa.py =====
dict_items([('월', 'monday'), ('화', 'tuesday'), ('수', 'wednesday'), ('목', 'thursday')])
monday
tuesday
wednesday
thursday

dict_values(['monday', 'tuesday', 'wednesday', 'thursday'])
>>>
```

# Chap 06 키와 값의 나열인 딕셔너리와 중복을 불허하는 집합

## Section 1.4 딕셔너리 항목의 순회

- 반복 for문에서 딕셔너리 변수로만 모든 키 순회
  - 반복 for문에서는 시퀀스 위치에 있는 딕셔너리 변수만으로도 모든 키를 순회할 수 있다.

## Section 1.5 딕셔너리 항목의 참조와 삭제

- 키로 조회하는 딕셔너리 메소드 get(키[, 키가\_없을\_때\_반환\_값])
  - 딕셔너리 메소드 get(키)는 키의 해당 값을 반환한다.
  - 메소드 get(키)는 딕셔너리에 키가 없어도 오류가 발생하지 않고 None을 반환한다.
  - 만일 키 뒤에 다른 인자를 넣으면 딕셔너리에 키가 없을 때 이 지정된 값을 반환한다.

```
game = dict(일월 = '소나무', 미월 = '매화', 삼월 = '벚꽃', 사월 = '동나무')
for key in game:
    print('%s : %s' % (key, game[key]))
```

```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Users/win/AppData/Local/Programs/Python/Python38-32/aa.py =====
일월 : 소나무
미월 : 매화
삼월 : 벚꽃
사월 : 동나무
>>>
===== RESTART: Shell =====
>>> city = {'대한민국': '부산', '뉴질랜드': '웰링턴', '캐나다': '몬트리올'}
>>> city.get('대한민국')
```

# Chap 06 키와 값의 나열인 딕셔너리와 중복을 불허하는 집합

## Section 1.5 딕셔너리 항목의 참조와 삭제

- 키로 삭제하는 딕셔너리 메소드 `pop(키[, 키가_없을_때_반환_값])`
  - 딕셔너리 메소드 `pop(키)`는 키인 항목을 삭제하고, 삭제되는 키의 해당 값을 반환한다.
  - 만일 삭제할 키가 없다면 두 번째에 지정한 값이 반환된다.
  - 인자로 키만 적었는데, 삭제할 키가 없다면 `KeyError`가 발생하므로 주의하자.
- 임의의 항목을 삭제하는 딕셔너리 메소드 `popitem()`
  - 딕셔너리 메소드인 `popitem()`은 임의의 (키, 값)의 튜플을 반환하고 삭제한다.
  - 만일 데이터가 하나도 없다면 오류가 발생한다.

```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> city = {'대한민국': '부산', '뉴질랜드': '웰링턴', '캐나다': '몬트리올'}
>>> print(city.pop('뉴질랜드'))
웰링턴
>>> city
{'대한민국': '부산', '캐나다': '몬트리올'}
>>> print(city.pop('중국', '없네요'))
없네요
>>> city
{'대한민국': '부산', '캐나다': '몬트리올'}
>>> print(city.pop('중국'))
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    print(city.pop('중국'))
  KeyError: '중국'
>>>
===== RESTART: Shell =====
>>> city = {'대한민국': '부산', '뉴질랜드': '웰링턴'}
>>> print(city.popitem())
('뉴질랜드', '웰링턴')
>>> city
{'대한민국': '부산'}
>>> print(city.popitem())
('대한민국', '부산')
>>> city
{}
>>> print(city.popitem())
Traceback (most recent call last):
  File "<pyshell#11>", line 1, in <module>
    print(city.popitem())
  KeyError: 'popitem(): dictionary is empty'
>>>|
```

## Chap 06 키와 값의 나열인 딕셔너리와 중복을 불허하는 집합

### Section 1.5 딕셔너리 항목의 참조와 삭제

- 문장 del로 딕셔너리 항목 삭제
  - 딕셔너리를 문장 del에 이어 키로 지정하면 해당 항목이 삭제된다.

### Section 1.6 딕셔너리 항목 전체 삭제와 변수 제거

- 모든 항목을 제거하는 딕셔너리 메소드 clear()
  - 딕셔너리 메소드 clear()는 기존의 모든 키:값 항목을 삭제한다.

```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> city = {'대한민국' : '부산', '뉴질랜드' : '웰링톤'}
>>> del city['뉴질랜드']
>>> city
{'대한민국' : '부산'}
>>>
===== RESTART: Shell =====
>>> city = {'대한민국' : '부산', '뉴질랜드' : '웰링톤', '캐나다' : '몬트리올'}
>>> city.clear()
>>> city
{}
>>> |
```

## Chap 06 키와 값의 나열인 딕셔너리와 중복을 불허하는 집합

### Section 1.6 딕셔너리 항목 전체 삭제와 변수 제거

- 문장 del로 딕셔너리 변수 자체 제거
  - 딕셔너리를 문장 del에 이어 키로 지정하면 변수 자체가 메모리에서 제거된다. 따라서 제거 이후에는 변수를 참조할 수 없다.

### Section 1.7 딕셔너리 결합과 키의 멤버십 검사 연산자 in

- 딕셔너리를 결합하는 메소드 update()
  - 딕셔너리의 메소드 update(다른 딕셔너리)는 인자인 다른 딕셔너리를 합병한다.
  - 인자 딕셔너리에 원 딕셔너리와 동일한 키가 있다면 인자 딕셔너리의 값으로 대체된다.

```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (In
tel)] on win32
Type "help()", "copyright()", "credits()" or "license()" for more information.
>>> city = {'대한민국': '부산', '뉴질랜드': '웰링턴'}
>>> del city
>>> city
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    city
NameError: name 'city' is not defined
>>>
===== RESTART: Shell =====
>>> kostock = {'Samsung Elec.': 40000, 'Daum KAKAO': 80000}
>>> usstock = {'MS': 150, 'Apple': 180}
>>> kostock.update(usstock)
>>> kostock
{'Samsung Elec.': 40000, 'Daum KAKAO': 80000, 'MS': 150, 'Apple': 180}
>>> |
```

## Chap 06 키와 값의 나열인 딕셔너리와 중복을 불허하는 집합

### Section 1.7 딕셔너리 결합과 키의 멤버십 검사 연산자 in

- 문장 in으로 쉽게 검사

- 문장 in으로 딕셔너리에 키가 존재하는지 간단히 검사할 수 있다.
- 값의 존재 여부는 확인할 수 없으므로 값으로 조회하면 항상 False다.
- not in도 가능하다.

```
>>> cp = {'한국' : '서울', '중국' : '북경', '독일' : '베를린'}
>>> '한국' in cp
True
>>> 'USA' in cp
False
>>> |
```

### Section 2.1 수학에서 배운 집합을 처리하는 자료형

- 원소는 유일하고 순서는 의미 없는 집합

- 집합은 중복되는 요소가 없으며, 순서도 없는 원소의 모음이다.
- 원소는 불변 값으로 중복될 수 없으며 서로 다른 값이어야 한다.
- 원소는 중복을 허용하지 않으며 원소의 순서는 의미가 없다.
- 집합의 원소는 정수, 실수, 문자열, 튜플 등 수정 불가능한 것이어야 하며 리스트나 딕셔너리처럼 가변적인 것은 허용되지 않는다.

## Chap 06 키와 값의 나열인 딕셔너리와 중복을 불허하는 집합

### Section 2.2 내장 함수 set()을 활용한 집합 생성

- 집합을 만드는 내장 함수 set()
  - 집합은 내장 함수 set()으로도 생성할 수 있다.
  - 인자가 없으면 빈 집합인 공집합이 생성된다. 인자가 있으면 하나이며, 리스트와 튜플, 문자열 등이 올 수 있다.
- 함수 set() 호출로 공집합 만들기
  - 내장 함수 set()으로 만들어지는 집합은 공집합이다.
  - 공집합을 출력해보면 set()인 것을 알 수 있다.
  - 빈 리스트와 튜플이 각각 [], () 이며, 빈 딕셔너리는 {}이다.

```
>>> s = set()
>>> type(s)
<class 'set'>
>>> s
set()
>>> d = {}
>>> type(d)
<class 'dict'>
>>> |
```



## Chap 06 키와 값의 나열인 딕셔너리와 중복을 불허하는 집합

### Section 2.2 내장 함수 set()을 활용한 집합 생성

- 리스트나 튜플을 인자로 사용하는 함수 set()
  - 함수 set()에서는 인자로 리스트 또는 튜플 자체를 사용할 수 있으며, 결과는 시퀀스 항목에서 중복을 제거한 원소로 구성된다.
  - 집합은 중괄호 안에 원소가 콤마로 구분돼 표시된다.
- 문자열을 인자로 사용하는 함수 set()
  - 함수 set()의 인자로 문자열이 사용되면 각각의 문자가 원소인 집합이 생성된다.
  - 순서는 의미없다.

```
>>> set([1, 2, 3])
{1, 2, 3}
>>> set((1, 2, 3))
{1, 2, 3}
>>> set(['a', 'b'])
{'b', 'a'}
>>>
===== RESTART: Shell =====
>>> set('abc')
{'b', 'a', 'c'}
>>> |
```

## Chap 06 키와 값의 나열인 딕셔너리와 중복을 불허하는 집합

### Section 2.2 내장 함수 set()을 활용한 집합 생성

- 수정 가능한 리스트와 딕셔너리는 집합의 원소로 사용 불가
  - 함수 set()의 인자에서 리스트나 튜플의 항목으로 수정될 수 있는 리스트나 딕셔너리는 허용되지 않으며, TypeError가 발생한다.

### Section 2.3 중괄호로 직접 원소를 나열해 집합 생성

- {원소1, 원소2, ...}로 생성
  - 집합을 생성하는 다른 방법은 중괄호{} 안에 직접 원소를 콤마로 구분해 나열하는 방법이다.
  - 집합의 원소는 문자, 문자열, 숫자, 튜플 등과 같이 변할 수 없는 것이어야 한다.

```
>>> set([1, [2, 3]])
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    set([1, [2, 3]])
TypeError: unhashable type: 'list'
>>>
===== RESTART: Shell =====
>>> {1, 2, 3}
{1, 2, 3}
>>> {1, 'seoul', 'a', (1,2, 3,4)}
{1, 'a', (1,2, 3,4), 'seoul'}
>>> |
```

## Chap 06 키와 값의 나열인 딕셔너리와 중복을 불허하는 집합

### Section 2.3 중괄호로 직접 원소를 나열해 집합 생성

- {원소1, 원소2, ...}에서 리스트나 딕셔너리는 원소로 사용 불가
  - 리스트나 딕셔너리와 같이 가변적인 것은 원소로 사용할 수 없다.

### Section 2.4 집합의 원소 추가와 삭제

- 집합 메소드 add(원소)로 추가
  - 이미 만들어진 집합에 원소를 추가하는 메소드다.

```
>>> {'a', 'b'}
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    {'a', 'b'}
TypeError: unhashable type: 'list'
>>> {[1:'a', 2:'b']}
Traceback (most recent call last):
  File "<pyshell#8>", line 1, in <module>
    {[1:'a', 2:'b']}
TypeError: unhashable type: 'dict'
>>>
===== RESTART: Shell =====
>>> odd = {1, 3, 5}
>>> odd.add(7)
>>> odd.add(9)
>>> print(odd)
{1, 3, 5, 7, 9}
>>> |
```

## Chap 06 키와 값의 나열인 딕셔너리와 중복을 불허하는 집합

### Section 2.4 집합의 원소 추가와 삭제

- 집합 메소드 remove(원소)와 discard(원소), pop()으로 항목 삭제

- 원소의 삭제는 remove(원소)를 사용한다. 원소가 없으면 KeyError가 발생한다.
- discard(원소)로도 원소를 삭제할 수 있으며, 원소가 없어도 오류가 나지 않는다.
- 임의의 원소를 삭제하려면 pop()을 사용해야 한다.

- 메소드 clear()로 집합의 모든 원소 삭제

- 집합의 모든 원소를 삭제하려면 clear()를 사용해야 한다.

```
>>> odd = {1, 3, 5}
>>> odd.remove(3)
>>> print(odd)
{1, 5}
>>> odd.remove(9)
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    odd.remove(9)
KeyError: 9
>>> odd = {1, 3, 5}
>>> odd.discard(3)
>>> print(odd)
{1, 5}
>>> odd.discard(9)
>>> odd = {1, 3, 5}
>>> print(odd.pop())
1
>>> print(odd)
{3, 5}
>>> odd = {1, 3, 5}
>>> odd.clear()
>>> print(odd)
set()
>>> |
```

## Chap 06 키와 값의 나열인 딕셔너리와 중복을 불허하는 집합

### Section 2.5 집합의 주요 연산인 합집합, 교집합, 차집합, 여집합

- 합집합 연산자 `|`와 메소드 `union()`, `update()`
  - 양쪽 모든 원소를 합하는 합집합은 연산자 `|`와 메소드 `union()`을 사용한다.
  - `union()`은 합집합을 반환하며 `a` 자체가 수정되지 않는다.
  - 메소드 `a.update(b)`도 합집합과 같은 효과가 있으며,  
`a`와 `b`의 합집합 결과가 호출하는 집합 `a`에 반영돼 수정되는 차이점이 있다.
- 교집합 연산자 `&`와 메소드 `intersection()`
  - 양쪽 모든 집합에 속하는 원소로 구성되는  
교집합은 연산자 `&`와 메소드 `intersection()`을 사용한다.
  - 메소드 `a.intersection(b)`는 집합 `a`, `b`에 영향을 미치지 않는다.  
그러나 메소드 `a.intersection_update(b)`는 `a`를 교집합으로 수정한다.

```
>>> a = {4, 6, 8, 10, 12}
>>> b = {3, 6, 9, 12}
>>> a | b
{3, 4, 6, 8, 9, 10, 12}
>>> a.union(b)
{3, 4, 6, 8, 9, 10, 12}
>>> a
{4, 6, 8, 10, 12}
>>> a.update(b)
>>> a
{3, 4, 6, 8, 9, 10, 12}
>>>
```

```
===== RESTART: Shell =====
>>> a = {4, 6, 8, 10, 12}
>>> b = {3, 6, 9, 12}
>>> a & b
{12, 6}
>>> a.intersection(b)
{12, 6}
>>> a.intersection_update(b)
>>> a
{12, 6}
>>>
```

## Chap 06 키와 값의 나열인 딕셔너리와 중복을 불허하는 집합

### Section 2.5 집합의 주요 연산인 합집합, 교집합, 차집합, 여집합

- 여집합 연산자 ^와 메소드 `symmetric_difference()`
  - 한쪽 집합에는 소속되지만 교집합이 아닌 원소로 구성되는 여집합은 대칭 차집합으로도 부른다.
  - 연산자 ^와 메소드 `symmetric_difference()`를 사용한다.

```
>>> a = {4, 6, 8, 10, 12}
>>> b = {3, 6, 9, 12}
>>> a ^ b
{3, 4, 8, 9, 10}
>>> a.symmetric_difference(b)
{3, 4, 8, 9, 10}
>>> |
```

## Chap 06 키와 값의 나열인 딕셔너리와 중복을 불허하는 집합

### Section 2.5 집합의 주요 연산인 합집합, 교집합, 차집합, 여집합

- 집합 연산의 축약 대입 연산자 |=, &=, -=, ^=와 메소드 intersection\_update() 등
  - 다음은 합집합의 연산자 |=으로 합집합의 결과가 왼쪽 피연산자에 대입된다. 따라서 메소드 update()의 결과와 같다.

```
>>> A = set('abcd'); B = set('cde')
>>> A |= B
>>> A
{'d', 'a', 'c', 'e', 'b'}
```

- 다음은 교집합의 연산자 &=으로 교집합의 결과가 왼쪽 피연산자에 대입한다. 결과가 같은 메소드 intersection\_update()도 제공한다.

```
>>> A = set('abcd'); B = set('cde')
>>> A &= B
>>> A
{'c', 'd'}
>>> A = set('abcd'); B = set('cde')
>>> A.intersection_update(B)
>>> A
{'c', 'd'}
```

- 다음은 차집합 연산자 -=에 대한 코드이며, 메소드 difference\_update()도 제공한다.

- 다음은 여집합의 연산자 ^=에 대한 코드이며, 메소드 symmetric\_difference\_update()도 제공한다.

## Chap 06 키와 값의 나열인 딕셔너리와 중복을 불허하는 집합

### Section 2.5 집합의 주요 연산인 합집합, 교집합, 차집합, 여집합

- 집합 연산의 축약 대입 연산자  $|=$ ,  $\&=$ ,  $-=$ ,  $\wedge=$ 와 메소드 `intersection_update()` 등

- 다음은 차집합 연산자  $-=$ 에 대한 코드이며, 메소드 `difference_update()`도 제공한다.

```
>>> A = set('abcd'); B = set('cde')
>>> A -= B
>>> A
{'b', 'a'}
>>> |
```

- 다음은 여집합의 연산자  $\wedge=$ 에 대한 코드이며, 메소드 `symmetric_difference_update()`도 제공한다.

```
>>> A = set('abcd'); B = set('cde')
>>> A ^= B
>>> A
{'b', 'a', 'e'}
>>> |
```



감사합니다.

