파이썬프로그래밍 포트폴리오

과목명 : 파이썬프로그래밍

담당교수 : 강환수 교수님

학과 : 컴퓨터정보공학

학번 : 20192650

이름 : 배성연

제출일 : 20.05.27

목차

동양미래대학교 DONGYANG MIRAE UNIVERSITY

강 의 계 획 서

아시아 직업교육 허브대학

2020 학년도 1학기	전공	컴퓨터정보공학과	학부	컴퓨터공학부
과 목 명	파이썬프로그래밍(2019009-PD)			

강의실 과 강의시간	수:6(3-217),7(3-217),8(3-217)	학점	3
교과분류	이론/실습	시수	3

담당 교수	강환수 + 연구실 : 2호관-706 + 전 화 : 02-2610-1941 + 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 매우중요한 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, 신용현	홍릉과학출판사	

수업시 사용도구	파이썬 기본 도구, 파이참, 아나콘다와 주피터 노트북
평가방법	중간고사 30%, 기말고사 40%, 과제물 및 퀴즈 10% 출석 20%(학교 규정, 학업성적 처리 지침에 따름)
수강안내	1. 파이썬의 개발환경을 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.

1 주차	[개강일(3/16)]
학습주제	교과목 소개 및 강의 계획 1장 파이썬 언어의 개요와 첫 프로그래밍
목표및 내용	• 파이썬 언어란 무엇인지 이해하고 이 언어가 인기 있는 이유를 설명할 수 있다. • 파이썬 개발 도구를 설치해 프로그램을 구현할 수 있다. • 파이썬의 특징과 활용 분야를 설명할 수 있다.
미리읽어오기	교재 1장, 파이썬 개발환경 설치 파이썬 IDLE
과제,시험,기타	도전 프로그래밍
2 주차	[2주]
학습주제	2장 파이썬 프로그래밍을 위한 기초 다지기
목표및 내용	• 파이썬의 재료인 문자열과 수에 대해 이해하고 코드로 구현할 수 있다. • 변수를 이해하고 다양한 대입 연산자를 활용할 수 있다. • 표준 입력으로 문자열을 입력받은 후 원하는 자료로 변환해 활용할 수 있다. • 파이썬 IDLE을 활용할 수 있다.
미리읽어오기	교재 2장 리터럴과 변수의 이해 아나콘다의 주피터 노트북
과제,시험,기타	도전 프로그래밍
3 주차	[3주]
학습주제	3장 일상에서 활용되는 문자열과 논리 연산
목표및 내용	• 문자열에서 문자나 부분 문자열을 반환하는 여러 방법을 구현할 수 있다. • 문자열 객체에 소속된 다양한 메소드를 이해하고 활용할 수 있다. • 논리 값을 이해하고 다양한 연산을 사용해 실생활에서의 표현에 활용할 수 있다. • 아나콘다의 주피터 노트북을 활용할 수 있다.
미리읽어오기	교재 3장 문자열과 논리연산 파이참(pycharm)
과제,시험,기타	도전 프로그래밍
4 주차	[4주]
학습주제	4장 일상생활과 비유되는 조건과 반복
목표및 내용	• 조건에 따라 하나를 결정하는 if문을 구현할 수 있다. • 반복을 수행하는 while문과 for문을 구현할 수 있다. • 임의의 수인 난수를 이해하고 반복을 제어하는 break문과 continue문을 활용할 수 있다. • 파이참(pycharm)을 활용할 수 있다.
미리읽어오기	교재 4장 조건과 반복
과제,시험,기타	도전 프로그래밍

5 주차	[5주]
학습주제	5장 항목의 나열인 리스트와 튜플
목표및 내용	• 다양한 종류의 항목을 쉽게 나열하는 리스트를 구현할 수 있다. • 리스트에서 부분 참조 방법, 이를 이용한 수정, 리스트 연결, 삽입과 삭제 그리고 리스트 컴프리헨션 등을 구현할 수 있다. • 수정할 수 없는 다양한 종류의 항목 나열을 쉽게 처리하는 튜플을 구현할 수 있다.
미리읽어오기	교재 5장 배열과 리스트
과제,시험,기타	도전 프로그래밍
6 주차	[6주]
학습주제	6장 키와 값의 나열인 딕셔너리와 중복을 불허하는 집합
목표및 내용	• 키와 값의 쌍인 항목을 관리하는 딕셔너리를 생성하고 수정하는 방법을 이해하고, 다양한 방법으로 딕셔너리를 구현할 수 있다. • 집합의 특징을 이해하고, 합집합 등과 같은 다양한 집합의 연산을 구현할 수 있다. • 내장 함수 zip( )과 enumerate( ), 시퀀스 간의 변환을 이해하고, 구현할 수 있다.
미리읽어오기	교재 6장 집합
과제,시험,기타	도전 프로그래밍
7 주차	[7주]
학습주제	7장 특정 기능을 수행하는 사용자 정의 함수와 내장 함수
목표및 내용	• 함수의 내용과 필요성을 이해하고 함수를 직접 정의해 호출할 수 있다. • 인자의 기본 이해와 기본값 지정, 가변 인수와 키워드 인수를 활용할 수 있다. • 간편한 람다 함수와 표준 설치된 내장 함수를 사용할 수 있다.
미리읽어오기	교재 7장 함수의 정의와 호출
과제,시험,기타	도전 프로그래밍
8 주차	[중간고사]
학습주제	- 직무수행능력평가 1차(중간고사)
목표및 내용	직무수행능력평가, 서술형 평가
미리읽어오기	교재 1장에서 7장까지
과제,시험,기타	
9 주차	[9주]
학습주제	8장 조건과 반복, 리스트와 튜플 기반의 미니 프로젝트 I
목표및 내용	8개의 미니 프로젝트를 스스로 생각하고 프로그래밍해 코딩 능력뿐 아니라 문제 해결 능력을 키울 수 있다.
미리읽어오기	교재 8장
과제,시험,기타	

10 주차	[10주]
학습주제	9장 라이브러리 활용을 위한 모듈과 패키지
목표및 내용	• 표준 모듈을 이해하고 사용자 정의 모듈도 직접 구현해 사용할 수 있다. • 표준 모듈인 turtle을 사용해 기본적인 도형을 그릴 수 있다. • 써드파티 모듈 numpy와 matplotlib 등을 설치해 활용할 수 있다.
미리읽어오기	교재 9장
과제,시험,기타	도전 프로그래밍
11 주차	[11주]
학습주제	10장 그래픽 사용자 인터페이스 Tkinter와 Pygame
목표및 내용	• GUI를 이해하고 GUI 표준 모듈인 Tkinter를 사용해 필요한 위젯을 구성하고 윈도를 생성할 수 있다. • 이벤트 처리를 이해하고 Tkinter에서 이벤트 처리를 구현할 수 있다. • 써드파티 GUI 모듈인 pygame을 설치해 기본적인 윈도를 구현할 수 있다.
미리읽어오기	교재 10장
과제,시험,기타	도전 프로그래밍
12 주차	[12주]
학습주제	11장 실행 오류 및 파일을 다루는 예외 처리와 파일 입출력
목표및 내용	• 예외 처리의 필요성을 이해하고 try except 구문을 사용해 예외를 처리할 수 있다. • 프로그램에서 파일을 생성하는 필요성을 이해하고 필요한 파일을 만들 수 있다. • 이미 생성된 파일에서 내용을 읽어 처리할 수 있다
미리읽어오기	교재 11장
과제,시험,기타	도전 프로그래밍
13 주차	[13주]
학습주제	12장 일상생활의 사물 코딩인 객체지향 프로그래밍
목표및 내용	• 객체와 클래스를 이해하고 필요한 클래스를 정의하고 객체를 만들어 활용할 수 있다. • 클래스 속성과 인스턴스 속성, 정적 메소드와 클래스 메소드를 이해하고 정의할 수 있다. • 상속을 이해하고 부모 클래스와 자식 클래스를 정의할 수 있다. • 추상 메소드와 추상 클래스를 이해하고 정의할 수 있다
미리읽어오기	교재 12장
과제,시험,기타	도전 프로그래밍
14 주차	[14주]
학습주제	13장 GUI 모듈과 객체지향 기반의 미니 프로젝트 II
목표및 내용	학습한 파이썬 문법 구조와 프로그래밍 기법을 활용해 8개의 미니 프로젝트를 스스로 생각하고 프로그래밍해 코딩 능력뿐 아니라 문제 해결 능력을 키울 수 있다.
미리읽어오기	교재 1장
과제,시험,기타	

15 주차	[기말고사]
학습주제	직무수행능력평가 2차(기말고사)
목표및 내용	직무수행능력평가, 서술형평가
미리읽어오기	8장에서 13장까지
과제,시험,기타	

수업지원 안내	장애학생을 위한 별도의 수강 지원을 받을 수 있습니다. 언어가 문제가 되는 학생은 글로 된 과제 안내, 확대문자 시험지 제공 등의 지원을 드립니다.

제 1 장 파이썬이란

파이썬은 배우기 쉽고 누구나 무료로 사용할 수 있는 오픈소스 프로그래밍 언어다. 1980 년대 후반에 네덜란드의 귀도 반 로섬(Guido Van Rossum)이 개발했다.

현재 미국과 우리나라의 대학 등 전 세계적으로 가장 많이 가르치는 프로그래밍 언어 중 하나다. 특히 비전공자의 컴퓨터 사고력을 키우기 위한 프로그래밍 언어로도 많이 활용되고 있다. 파이썬은 배우기 쉽고 간결하며, 개발 속도가 빠르고 강력하기 때문이다. 또한 파이썬은 라이브러리가 풍부하고 다양한 개발 환경을 제공하고 있어 개발자가 빠르게 소프트웨어를 개발하는 데 도움을 준다.

파이썬은 베이직 언어와 마찬가지로 인터프리터(해석기) 위에서 실행되는 인터프리트 방식의 언어이다. 인터프리트 방식이란, 동시 번역처럼 파이썬의 문장 한 줄 한 줄마다 즉시 번역해 실행하는 방식이다.

제 2 장 파이썬의 기초

1. 문자열과 수

파이썬에서는 문자 하나 또는 문자가 모인 단어나 문장 또는 단락 등을 문자열이라 한다. 작은따옴표로 앞뒤를 둘러싸 '문자열' 또는 "문자열"처럼 표시한다.

1) 문자열 연산자 +,*,주석

■ **+ 기호 (문자열에서 문자열을 연결하는 역할)**

```
print("안녕" + "하세요")
```

```
안녕하세요
```

■ *** 기호 (문자열에서 문자열을 지정된 수만큼 반복)**

```
print("안녕" * 4)
```

```
안녕안녕안녕안녕
```

■ **''' or """ 기호 (여러 줄의 문자열 처리)**

```
print(""" 안녕하세요 컴퓨터정보공학과를 다니고 있습니다.
파이썬을 배우고 있어요""")
```

```
 안녕하세요 컴퓨터정보공학과를 다니고 있습니다
파이썬을 배우고 있어요
```

■ **# 기호 (주석)**

```
print("안녕하세요!") # 주석사용. 반영x
```

```
안녕하세요!
```

2) 정수와 실수의 연산

연산자	의미
+	두 피연산자를 더하거나 수의 부호
-	두 피연산자를 빼거나 수의 부호
*	두 피연산자를 곱하기
/	왼쪽을 오른쪽 피연산자로 나누기
%	왼쪽을 오른쪽 피연산자로 나눈 나머지
//	왼쪽을 오른쪽 피연산자로 나눈 결과에서 작거나 같은 정수
**	왼쪽을 오른쪽 피연산자로 거듭제곱

예제)

```
a = 3
b = 4

print('a + b = ', a + b)    # 7
print('a - b = ', a - b)    # -1
print('a * b = ', a * b)    # 12
print('a / b = ', a / b)    # 0.75
print('a % b = ', a % b)    # 3
print('a // b = ', a // b)  # 0
print('a ** b = ', a ** b)  # 81
```

```
a + b =  7
a - b =  -1
a * b =  12
a / b =  0.75
a % b =  3
a // b =  0
a ** b =  81
```

■ **eval() 함수**

표현식 문자열 실행 함수로 함수 인자가 연산 가능한 문자열이어야 한다.

```
>>> eval('3 + 20 / 2')
13.0
>>> eval('"hello " * 4')
'hello hello hello hello '
```

문자열 연산도 가능하다. 내부 문자열은 큰따옴표 사용한다.

2. 변수, 키워드, 대입 연산자

1) 자료형 type() 함수

파이썬 자료형 : 정수 = int, 실수 = float, 문자열 = str

대화형 모드에서 자료형을 알아보려면 type() 함수를 사용한다.

```
>>> type(2)
<class 'int'>
>>> type(2.2)
<class 'float'>
>>> type('python')
<class 'str'>
```

2) 변수와 대입 연산자

변수란 자료를 저장하는 공간이다. 대입 연산자인 = 를 사용하여 저장한다.

변수를 구성하는 문자는 대소문자의 영문자, 숫자, _로 구성되며 숫자는 앞에 올 수 없고 키워드는 사용할 수 없다.

ex) data = 2 라고 하면 data 라는 변수에 2 라는 값을 저장할 수 있다.

- 키워드 : 프로그래밍 언어 문법에서 사용하는 이미 예약된 단어로 총 33 개.
- 식별자 : 프로그래머가 이름을 짓는 단어.

■ **divmod() 함수**

나누기 몫 연산자 //와 나머지 연산 %를 수행한 결과를 반환한다.

```
>>> divmod(31,3)
(10, 1)
```

3. 표준 입력

1) 표준 입력

쉘이나 콘솔에서 사용자의 입력을 받아 처리하는 방식을 말한다.

■ **input() 함수**

입력되는 표준입력을 문자열로 읽어 반환하는 함수이다. 입력 문자열을 변수에 저장하려면 대입 연산자 = 을 사용해 저장해야 한다.

```
name = input('이름? ')
print('이름 : ',name)

이름? 배성연
이름 :  배성연
```

■ **str(), int(), float() 함수**

str() 함수 : 정수와 실수를 문자열로 반환

int() 함수 : 정수 형태의 문자열을 정수로 반환

float() 함수 : 실수 형태의 문자열을 실수로 변환

```
>>> str(20)
'20'
>>> int(2.1)
2
>>> float(3.4)
3.4
```

주의! 문자열이 정수나 실수가 아니면 오류!

```
Traceback (most recent call last):
  File "<pyshell#13>", line 1, in <module>
    int('python')
ValueError: invalid literal for int() with base 10: 'python'
>>>
```

★ 숫자형태의 문자열을 정수로 변환해보기

```
cm = input('키를 입력해주세요 ')
print('키 : ',int(cm))
```

```
키를 입력해주세요 158
키 :  158
```

2) 16 진수, 10 진수, 8 진수, 2 진수

■ **상수표현법**

진수	상수 표현법	10진수 변환 함수
16진수	0x (0x1f, 0X1E, 0Xa)	hex()
8진수	0o (0o17, 0O16, 0O12)	oct()
2진수	0b (0b11, 0B10, 0B1010)	bin()

★ 10 진수를 변환해서 출력해보기

```
data = int(input('정수입력 >> '))

print('2진수 : ',bin(data)) #2진수로
print('8진수 : ',oct(data)) #8진수로
print('10진수 : ',data)     #10진수로
print('16진수 : ',hex(data))#16진수로
```

```
정수입력 >> 25
2진수 :  0b11001
8진수 :  0o31
10진수 :  25
16진수 :  0x19
```

■ **int(정수, 진법기수)**

함수 int(strnum), int(strnum,10)은 10 진수 형태의 문자열을 10 진수 정수로 변환한다. (16, 8, 2 진수도 가능)

★ 16 진수 입력으로 출력해보기

```
invar = input('16진수 입력 >> ')
data = int(invar,16)

print('2진수 : ',bin(data)) #2진수로
print('8진수 : ',oct(data)) #8진수로
print('10진수 : ',data)     #10진수로
print('16진수 : ',hex(data))#16진수로
```

```
16진수 입력 >> 0x1b
2진수 :  0b11011
8진수 :  0o33
10진수 :  27
16진수 :  0x1b
```

2 장 요점정리!

1. 파이썬은 문자형식과 문자열 형식의 구분이 없고 다 문자열이다.
2. 산술연산자가 java 나 c 언어와 다르다.

(/연산자는 결과가 실수형이 나오고 //연산자는 정수형이 나온다.)

3. eval 함수를 사용할 때 함수인자는 반드시 문자열이어야 한다.
4. 표준입력으로 받은 문자를 정수형이나 실수형으로 만들 때 input 앞에 타입을 붙여줘야 한다. (int(input)), float(input))형식)

2 장 과제

```
#2번
km = int(input('차의 속도를 입력(km) >> '))
result = km/1.61
print(km,'(km)는',result,'마일(miles)이다.')
```

```
=== RESTART: C:/Users/pOmOq/AppData/Local/Programs/Python/Python38-32/ch1.py ===
차의 속도를 입력(km) >> 135
135 (km)는 83.85093167701862 마일(miles)이다.
>>>
```

```
#4번
dule1 = 40120
dule2 = 40074.77587040001
print('알려진 지구 둘레 : ',dule1)
print('지구와 같은 원둘레 : ',dule2)
result = dule1-dule2
print('차이 : ',result,'(km)')
```

```
=== RESTART: C:/Users/pOmOq/AppData/Local/Programs/Python/Python38-32/ch2.py ===
알려진 지구 둘레 :  40120
지구와 같은 원 둘레 :  40074.77587040001
차이 :  45.22412959999201 (km)
>>>
```

```
#6번
num1 = int(input('Enter First Number : '))
num2 = int(input('Enter Second Number : '))
print(num1,'/',num2,'==>',num1/num2)
print(num1,'%',num2,'==>',num1%num2)
print(num1,'//',num2,'==>',num1//num2)
print(num1,'**',num2,'==>',num1**num2)
```

```
=== RESTART: C:/Users/pOmOq/AppData/Local/Programs/Python/Python38-32/ch3.py ===
Enter First Number : 12
Enter Second Number : 5
12 / 5 ==> 2.4
12 % 5 ==> 2
12 // 5 ==> 2
12 ** 5 ==> 248832
>>>
```

```
#8번
num = int(input('네 자릿수 정수 입력 >> '))
num1 = int(num % 10)
num2 = int((num/10)%10)
num3 = int((num/100)%10)
num4 = int((num/1000)%10)

print(num1,num2,num3,num4)
```

```
=== RESTART: C:/Users/pOmOq/AppData/Local/Programs/Python/Python38-32/ch4.py ===
네 자릿수 정수 입력 >> 5432
2 3 4 5
>>>
```

제 3 장 문자열과 논리 연산

1. 문자열 다루기

1) 문자열 참조

■ **len() 함수**

문자열의 길이를 알 수 있다.

```
>>> data='python'
>>> len(data)
6
```

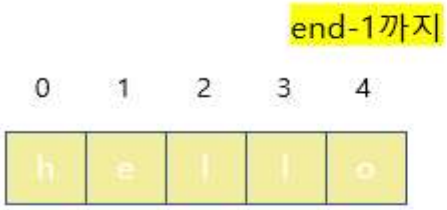
■ **문자열의 문자 참조**

문자열을 구성하는 문자는 0 부터 시작된다. 대괄호 안에 기술해 참조할 수 있다. -1 로 시작하면 첨자를 역순으로 참조한다.

2) 문자열 슬라이싱

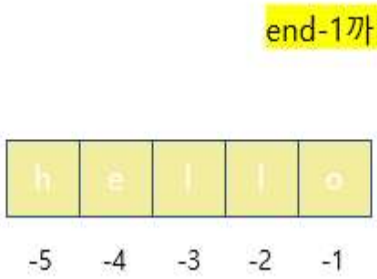
슬라이싱 : 문자열에서 일부를 참조하는 방법

■ **0 과 양수 슬라이싱 [start:end] (start 에서 end -1 까지 반환)**



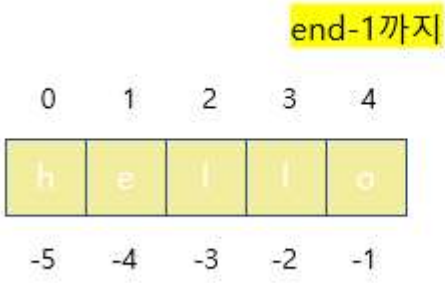

```
>>> 'hello' [1:3]
'el'
>>> 'hello'[0:5]
'hello'
>>> 'hello'[0:len('hello')]
'hello'
>>>
```

■ 음수 슬라이싱 [start:end] 음수사용가능 (start 에서 end -1 까지 반환)



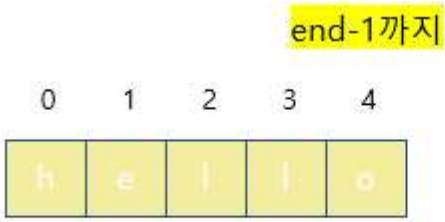

```
>>> 'hello'[-4:-2]
'el'
>>> 'hello'[-5:-1]
'hell'
>>> 'hello'[-len('hello'):-1]
'hell'
>>>
```

■ 혼합 슬라이싱 (순방향의 0, 양수, 역방향의 음수를 혼합



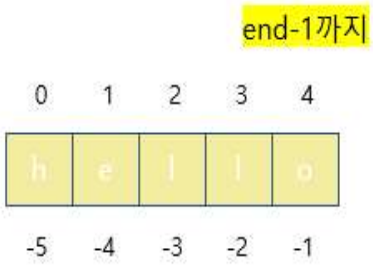

```
>>> 'hello'[1:-1]
'ell'
>>> 'hello'[2:-2]
'l'
>>> 'hello'[-5:4]
'hell'
>>>
```

■ start 와 end 를 비우면 처음부터 끝까지를 의미




```
>>> 'hello'[:]
'hello'
>>> 'hello'[1:]
'ello'
>>> 'hello'[:4]
'hell'
>>>
```

■ step 으로 조정 [start:end:step] ([:], [: :], [: :1]은 모두 전체 문자열 반환)
([: :-1]은 역순으로 출력)




```
>>> 'hello'[:]
'hello'
>>> 'hello'[1:]
'ello'
>>> 'hello'[:4]
'hell'
```

```
>>> 'hello'[0:5:2]
'hlo'
>>> 'hello'[::-1]
'olleh'
>>> 'hello'[-1:-5:-1]
'olle'
>>>
```

3) 문자함수, 이스케이프 시퀀스

■ **ord(), chr() 문자함수**

ord() : 유니코드 번호 반환

chr() : 문자를 반환

```
>>> ord('배')
48176
>>> chr(48176)
'배'
```

■ **이스케이프 시퀀스**

하 나의 문자를 역슬래시(\)로 시작하는 조합을 말한다.

이스케이프 시퀀스 문자	설명	이스케이프 시퀀스 문자	설명
\\	역슬래시	\n	새 줄
\'	작은따옴표	\t	수평탭
\"	큰따옴표	\N{name}	유니코드의 이름

```
str = '사과는 ₩'과일₩' 이다.'
print(str)
```

```
사과는 '과일' 이다.
```

■ **max(), min() 함수**

max : 인자의 최댓값 반환

min : 인자의 최솟값 반환

```
>>> max('python')
'y'
>>> min('python')
'h'
>>> max(3,5,2)
5
>>> min(3,5,2)
2
```

2. 문자열 메소드

1) 문자열 대체, 삽입, 부분 문자열

메소드 : 클래스에 소속된 함수

■ **replace() 함수**

문자열을 바꿔서 반환하는 메소드이다. (str.replace(a, b)는 a 를 b 로 바꿔서 반환)

```
>>> str = '사과는 과일이다.'
>>> str.replace('사과','바나나')
'바나나는 과일이다.'
```

■ **count() 메소드**

부분 문자열 출현 횟수를 반환한다.

```
>>> str = '가위 바위 보 가위 바위 보'
>>> str.count('가위')
2
```

■ **join() 메소드**

문자와 문자 사이에 문자열을 삽입한다.

```
>>> str = '가나다라마바사'
>>> '->'.join(str)
'가->나->다->라->마->바->사'
```

2) 문자열 찾기

■ **find(), index() 메소드**

찾는 문자열의 첨자를 반환한다. find() 함수는 찾는 문자열이 없다면 -1 을 반환하고 index()는 ValueError 를 발생시킨다.

```
>>> str = '사과 귤 바나나'
>>> str.find('사과')
0
>>> str.index('귤')
3
>>> str.find('복숭아')
-1
>>> str.index('수박')
Traceback (most recent call last):
  File "<pyshell#11>", line 1, in <module>
    str.index('수박')
ValueError: substring not found
```

3) 문자열 나누기

■ **split() 메소드**

문자열을 기준(spacebar, tab, enter)으로 나눠준다. ()안에 특정한 값이 있으면 그 부분을 구분자로 이용해서 문자열을 나눠준다.

```
>>> '사과 귤 바나나'.split()
['사과', '귤', '바나나']
>>> '사과, 귤, 바나나'.split(',')
['사과', ' 귤', ' 바나나']
```

4) 문자열 변환

■ **upper(), lower() 메소드**

upper() 메소드는 모두 대문자로 변환하고, lower() 메소드는 소문자로 변환한다.

```
>>> 'hello'.upper()
'HELLO'
>>> 'HELLO'.lower()
'hello'
```

■ **center() 메소드**

폭을 지정하고 중앙에 문자열을 배치한다. (문자열.center(폭, 채울 문자)로 사용)

```
>>> '파이썬 강좌'.center(30,'-')
'------------파이썬 강좌------------'
>>> '파이썬 강좌'.center(30) #채울문자 생략하면 공백으로 채움
'            파이썬 강좌            '
```

5) 출력을 정형화 하는 함수

■ **format() 메소드**

format() 메소드를 사용하면 문자열 중간에 변수나 상수를 같이 출력할 수 있다. 문자열 내부에 { }를 사용해서 변수와 상수를 넣으면 출력할 수 있다. {0},{1} 등으로 인자의 순서를 나타내는 정수를 입력하면 순서도 조정할 수 있다.

```
>>> str = '{} + {} = {}'.format(2,4,2+4)
>>> print(str)
2 + 4 = 6
>>> a, b = 2,4
>>> print('{} + {} = {}'.format(a,b,a+b))
2 + 4 = 6
```

6) C 언어 스타일

■ **%d 와 %f**

%d : 10 진수, %x : 16 진수. %o : 8 진수, %f : 실수, %c : 문자, %s : 문자열 등을 사용해 %로 이어진 뒤에 상수나 변수를 출력한다.

```
>>> print('%10.2f' % 2.71821)
      2.72
>>> print('%10c' % 'p')
         p
```

3. 논리 자료와 연산

1) 논리값, 논리연산

■ **논리 유형 bool, 함수 bool()**

파이썬은 논리값으로 True 와 False 를 키워드로 제공한다. 이러한 논리값의 자료형은 클래스 bool 로 제공한다.

정수 : 0, 실수 : 0.0, 빈 문자열 = False 와 음수와 양수, 'java' 등의 문자열 = True 는 내장 함수 bool()로 논리값을 알 수 있다.

```
>>> bool(0),bool(3.14),bool('java')
(False, True, True)
```

■ **논리곱 and 논리합 or**

논리곱인 and 와 & 연산자는 두 항이 모두 참이어야 True 이다.

```
>>> True & True, True and False
(True, False)
```

논리합 or 와 | 연산자는 하나라도 참이면 True 이다

```
>>> True | False, False or False, True | True
(True, False, True)
```

2) 관계 연산

■ **관계연산자**

30 < speed <= 80 사용 가능 하다.

연산자	연산 사용	의미
>	a > b	크다
>=	a >= b	크거나 같다
<	a < b	작다
<=	a <= b	작거나 같다
==	a == b	같다
!=	a != b	다르다

3) 멤버십 연산

■ **in 키워드**

키워드인 in 은 멤버의 소속을 알 수 있는 멤버십 연산이다. 결과는 True 아니면 False 이다. 키워드 in 뒤에 문자열에서 in 앞의 부분 문자열이 있으면 True 를 반환한다. 반대로 not in 뒤에 문자열에서 앞에 부분 문자열이 없으면 True 를 반환한다.

```
>>> str = ('사과')
>>> str2 = ('사과 귤')
>>> str in str2
True
>>> str not in str2
False
>>>
```

4) 비트 논리 연산

■ **비트 논리곱 &, 비트 논리합 |, 비트 배타적 논리합^**

비트 논리곱 : 비트 모두 1 이면 1

비트 논리합 : 비트가 하나라도 1 이면 1

비트 배타적 논리합 : 두 비트가 다르면 1 같으면 0

m	n	논리곱	논리합	배타적 논리합
		m & n	m & n	m & n
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

★ 비트 마스크

비트 논리곱 연산은 정수의 특정 비트 값을 알아 내는 데 사용된다. 원하는 특정 비트를 모두 1 로 지정한 마스크(mask) 값을 정수와 논리곱으로 연산한 정수 & mask 결과는 정수의 특정 비트 값만을 뽑아낸다.

5) 비트 이동 연산자

■ **비트 이동 연산자 >>와 <<**

비트 단위로 연산자의 방향인 오른쪽 또는 왼쪽으로 지정된 횟수만큼 이동시키는 연산이다. <<에 의해 생기는 오른쪽 빈자리는 모두 0 으로 채워지고 >> 에 의해 생긴 왼쪽 빈자리는 원래의 정수 부호에 따라 0 이나 양수이면 0, 음수면 1 로 채워진다.

```
>>> 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<<3))
10진수 184, 2진수 10111000
```

3 장 요점정리!

1. 슬라이싱 매우 중요함. [:] 공백이면 처음부터 끝까지를 의미하고 [: :-1]이면 역순으로 배열한다. step 을 줘서 간격도 조절할 수 있다. 양수와 음수를 혼합해서 사용할 수 있다.
2. format 메소드 중요함. 변수와 상수를 함께 사용하기 편리하다.
3. %d %f 등등 c 언어와 유사한 출력 방법이다.
4. 관계 연산. 50 < speed <= 80 으로 사용할 수 있다.

3 장 과제

ch1.py - C:₩Users₩p0m0q₩AppData₩Local₩Programs₩Python₩Python38-32...

File Edit Format Run Options Window Help

```
#2번
str = input('다섯 문자 이상의 문자열 입력 >> ')
le = len(str)
print('입력 문자열 : ',str)
print('첫 문자 : ',str[0:1])
print('마지막  문자 : ',str[le-1:])
print('첫 문자를 제외한  문자열 : ',str[1:])
print('마지막 문자를 제외한  문자열 : ',str[:le-1])
print('맨 앞과 뒤의 두 문자씩을 제외한 부분 문자열 : ',str[2:le-2])
print('문자 하나씩을 건너뛴 부분 문자열 : ',str[::2])
print('역문자열 : ',str[::-1])
```

Python 3.8.2 Shell

File Edit Shell Debug Options Window Help

```
Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb 25 2020, 22:45:29) [MSC v.1916 32 bit (In
tel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=== RESTART: C:₩Users₩p0m0q₩AppData₩Local₩Programs₩Python₩Python38-32₩ch1.py ===
다섯 문자 이상의 문자열 입력 >> Python String Slicing
입력 문자열 :  Python String Slicing
첫 문자 :  P
마지막  문자 :  g
첫 문자를 제외한  문자열 :  ython String Slicing
마지막 문자를 제외한  문자열 :  Python String Slicin
맨 앞과 뒤의 두 문자씩을 제외한 부분 문자열 :  thon String Slici
문자 하나씩을 건너뛴 부분 문자열 :  Pto tigSiig
역문자열 :  gnicilS gnirtS nohtyP
>>>
```

ch2.py - C:₩Users₩p0m0q₩AppData₩Local₩Programs₩Python₩Python38-32₩ch2.py (3.8.2)

File Edit Format Run Options Window Help

```
#4번
url = 'https://docs.python.org/3/tutorial'
p1 = url.find(':')
p2 = url.rfind('/')
print(url[:p1])
print(url[url.find('docs.python.org'):url.find('docs.python.org')+len('docs.python.org')])
print(url[p2+1:])
```

Python 3.8.2 Shell

File Edit Shell Debug Options Window Help

```
Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb 25 2020, 22:45:29) [MSC v.1916 32 bit (In
tel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=== RESTART: C:₩Users₩p0m0q₩AppData₩Local₩Programs₩Python₩Python38-32₩ch2.py ===
https
docs.python.org
tutorial
>>>
```

ch3.py - C:₩Users₩p0m0q₩AppData₩Local₩Programs₩Python₩Python38-32...

File Edit Format Run Options Window Help

```
#6번
a,b = input('실수 두개 입력 >> ').split()
aa = float(a)
bb = float(b)
print(aa,'>',bb, '결과 : ',aa>bb)
print(aa,'>=',bb, '결과 : ',aa>=bb)
print(aa,'<',bb, '결과 : ',aa<bb)
print(aa,'<=',bb, '결과 : ',aa<=bb)
print(aa,'==',bb, '결과 : ',aa==bb)
print(aa,'!=',bb, '결과 : ',aa!=bb)
```

Python 3.8.2 Shell

File Edit Shell Debug Options Window Help

```
Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb 25 2020, 22:45:29) [MSC v.1916 32 bit (In
tel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=== RESTART: C:₩Users₩p0m0q₩AppData₩Local₩Programs₩Python₩Python38-32₩ch3.py ===
실수 두개 입력 >> 5.4 2.7
5.4 > 2.7 결과 :  True
5.4 >= 2.7 결과 :  True
5.4 < 2.7 결과 :  False
5.4 <= 2.7 결과 :  False
5.4 == 2.7 결과 :  False
5.4 != 2.7 결과 :  True
>>>
```

ch4.py - C:₩Users₩p0m0q₩AppData₩Local₩Programs₩Python₩Python38-32...

File Edit Format Run Options Window Help

```
#8번
a = int(input('정수 입력 >> '))
b = int(input('지수승 입력 >> '))
print('정수{0:d} >> {1:d}, 결과:{2:d}'.format(a,b,a<<b))
print('정수{0:d} * 2**{1:d}, 결과:{2:d}'.format(a,b,a*2**b))
print('2진수(32비트): {0:32b} 정수:{1:d}'.format(a,a))
print('2진수(32비트): {0:32b} 정수:{1:d} << {2:d}'.format(a<<b,a,b))
print('2진수(32비트): {0:32b} 정수:{1:d} * 2**{2:d}'.format(a<<b,a,b))
```

Python 3.8.2 Shell

File Edit Shell Debug Options Window Help

```
Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb 25 2020, 22:45:29) [MSC v.1916 32 bit (In
tel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=== RESTART: C:₩Users₩p0m0q₩AppData₩Local₩Programs₩Python₩Python38-32₩ch4.py ===
정수 입력 >> 67
지수승 입력 >> 3
정수67 >> 3, 결과:536
정수67 * 2**3, 결과:536
2진수(32비트):                                1000011 정수:67
2진수(32비트):                             1000011000 정수:67 << 3
2진수(32비트):                             1000011000 정수:67 * 2**3
>>>
```

제 4 장 조건과 반복

1. 조건문 if, else

1) 조건

■ **if 문**

조건에 따라 해야 할 일을 처리해야 하는 경우 사용한다. if 문 논리 표현식 이후에는 반드시 콜론(:)이 있어야 한다. 콜론 이후 다음 줄부터 시작되는 블록은 반드시 들여쓰기해야 한다. (하지 않으면 오류 발생)

```
pm = 90
if 81 < pm: #콜론 필수
    print('마스크를 착용합시다')
```

```
마스크를 착용합시다
```

2) 조건을 선택

■ **if, else 문**

논리 표현식 결과가 True 와 False 에 따라 나뉘는 경우 사용한다. if 문에 조건이 False 라면 else 값이 반환된다. 역시 콜론을 사용해줘야 오류가 나지 않는다.

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

```
정수 입력 >> 11
11은 홀수다.
```

3) 여러 조건 중 하나를 선택

■ **if, elif**

조건이 여러 가지일 경우 하나를 선택하는 경우 사용한다. if 문이 True 면 if 의 값을 반환하고 if 문이 False 일 경우 elif 문이 실행된다. elif 문은 여러 개를 작성할 수 있다. if 와 elif 문에는 조건을 작성하고 마지막에는 else 를 작성하는데 else 는 생략 가능하다. 역시 콜론을 사용해줘야 오류가 나지 않는다.

```
num = int(input('점수는 ? '))
if 90 <= num:
    print('점수: {:d}, 등급: {}'.format(num, 'A'))
elif 80 <= num:
    print('점수: {:d}, 등급: {}'.format(num, 'B'))
elif 70 <= num:
    print('점수: {:d}, 등급: {}'.format(num, 'C'))
else:
    print('점수: {:d}, 등급: {}'.format(num, 'D'))
```

```
점수는 ? 85
점수: 85, 등급: B
```

4) 중첩된 조건

■ **중첩 조건**

조건문안에 조건이 들어가는 경우 사용한다.

```
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')
```

```
원하는 음료는? 1. 커피 2. 주스1
번호 선택? 1. 아메리카노 2. 카페라테 3. 카푸치노1
1. 아메리카노 선택
```

2. 반복문 제어

1) 내부 값으로 반복

■ **for 문**

항목의 나열인 시퀀스의 구성 요소인 모든 항목이 순서대로 변수에 저장돼 반복을 수행한다.

1. 여러 개의 값을 갖는 시퀀스에서 변수에 하나의 값을 순서대로 할당한다.
2. 할당한 변숫값을 갖고 블록의 문장들을 순차적으로 실행한다. 시퀀스의 그다음 값을 변수에 할당해 다시 반복을 실행한다.
3. 시퀀스의 마지막 항목까지 실행한 후 종료한다.

콜론과 반복 몸체인 블록 구성이 반드시 필요하다.

```
>>> for i in 1, 2, 3, 4, 5:
        print(i, end = ' ')

1 2 3 4 5
```

★ for 문 응용하기 (합과 평균 구하기)

```
sum = 0
for i in 1.1, 2.5, 3.6, 4.2, 5.4:
    sum += i
    print(i,sum)
else:
    print('합: %.2f, 평균: %.2f' % (sum, sum / 5))
```

```
1.1 1.1
2.5 3.6
3.6 7.2
4.2 11.4
5.4 16.8
합: 16.80, 평균: 3.36
```

■ **내장 함수 range()를 사용한 for 문**

반복 for 문 시퀀스에 내장 함수 range()를 활용하면 매우 간결하다. range(5)는 정수 0 에서 4 까지의 5 개의 항목인 정수로 구성되는 시퀀스를 만든다. 그러므로 총 5 번의 정수로 반복을 수행할 수 있다.

```
>>> for i in range(5):
        print(i, end=' ')
```

```
0 1 2 3 4
```

2) 횟수를 정하지 않은 반복

■ **while 문**

while 문은 for 문에 비해 간결하며 반복 조건 값에 따라 반복을 수행한다. while 문은 횟수를 정해 놓지 않고 조건이 False 가 될 때까지 반복을 수행하는 데 적합하다.

조건이 True 면 반복을 수행하고 False 면 반복을 하지 않고 종료한다. 반드시 콜론을 적어줘야 오류가 나지 않는다.

```
>>> n = 1
>>> while n <= 5:
        print(n, end=' ')
        n+=1
```

```
1 2 3 4 5
```

3) 중첩된 반복

■ **for 문안에 for 문 사용 (구구단)**

```
for i in range(2, 10):
    for j in range(1, 10):
        print('%d * %d = %2d' % (i, j, i * j), end= ' ')
    print()
```

```
2 * 1 =  2 2 * 2 =  4 2 * 3 =  6 2 * 4 =  8 2 * 5 = 10 2 * 6 = 12 2 * 7 = 14 2 * 8 = 16 2 * 9 = 18
3 * 1 =  3 3 * 2 =  6 3 * 3 =  9 3 * 4 = 12 3 * 5 = 15 3 * 6 = 18 3 * 7 = 21 3 * 8 = 24 3 * 9 = 27
4 * 1 =  4 4 * 2 =  8 4 * 3 = 12 4 * 4 = 16 4 * 5 = 20 4 * 6 = 24 4 * 7 = 28 4 * 8 = 32 4 * 9 = 36
5 * 1 =  5 5 * 2 = 10 5 * 3 = 15 5 * 4 = 20 5 * 5 = 25 5 * 6 = 30 5 * 7 = 35 5 * 8 = 40 5 * 9 = 45
6 * 1 =  6 6 * 2 = 12 6 * 3 = 18 6 * 4 = 24 6 * 5 = 30 6 * 6 = 36 6 * 7 = 42 6 * 8 = 48 6 * 9 = 54
7 * 1 =  7 7 * 2 = 14 7 * 3 = 21 7 * 4 = 28 7 * 5 = 35 7 * 6 = 42 7 * 7 = 49 7 * 8 = 56 7 * 9 = 63
8 * 1 =  8 8 * 2 = 16 8 * 3 = 24 8 * 4 = 32 8 * 5 = 40 8 * 6 = 48 8 * 7 = 56 8 * 8 = 64 8 * 9 = 72
9 * 1 =  9 9 * 2 = 18 9 * 3 = 27 9 * 4 = 36 9 * 5 = 45 9 * 6 = 54 9 * 7 = 63 9 * 8 = 72 9 * 9 = 81
```

3. 임의의 수인 난수와 반복을 제어

1) 난수 발생

■ **random() 함수**

random 의 함수 randint(시작, 끝)를 사용해 정수 시작과 끝수 사이에서 임의의 정수를 얻을 수 있다. 여기서는 시작과 끝을 모두 포함한다.

random.randint(1,3)는 import random 으로 모듈 활용을 선언한 후 1, 2, 3 중 한 가지 수를 임의로 얻을 수 있다.

★ 난수로 로또 만들어 보기

```
winnumber = 11, 17, 28, 30, 33, 35
print(' 모의 로또 당첨 번호 '.center(28, '='))
print(winnumber)
print()
print(' 내 번호 확인 '.center(30, '-'))
cnt = 0
import random
for i in range(6):
    n = random.randint(1, 45)
    if n in winnumber:
        print(n, 'O ', end = ' ')
        cnt += 1
    else:
        print(n, 'X ', end = ' ')

print()
print(cnt, '개 맞음')
```

```
======= 모의 로또 당첨 번호 ========
(11, 17, 28, 30, 33, 35)

---------- 내 번호 확인 -----------
35 O  20 X  1 X  33 O  17 O  45 X
3 개 맞음
```

2) 반복을 제어

- **break**

  반복을 종료한다. for 문이나 while 문 내부에서 break 는 반복을 무조건 종료한다.

```
while True:
    menu = input('[0]종료 [1]계속 ? ')
    if menu == '0':
        break
print('종료')
```

```
[0]종료 [1]계속 ? 1
[0]종료 [1]계속 ? 0
종료
```

- **continue**

  반복 몸체를 실행하지 않고 다음 반복을 계속 실행한다. for 문이나 while 문 내부에서 continue 는 이후의 반복을 실행하지 않고 다음 반복을 위한 조건을 수행한다.

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

```
p y t h n
```

4 장 요점정리!

1. if, else, elif, while 문 사용 시 콜론(:)을 꼭 사용해줘야 오류가 나지 않는다. 또한 콜론 이후 들여쓰기를 반드시 해야 한다.
2. range 함수를 사용하면 for 문을 간결하게 사용할 수 있다. 시작은 0 부터이다.
3. 조건문과 반복문은 중첩으로 사용이 가능하며 반복문안에 조건문을 사용할 수 있고 반대로 조건문안에 반복문도 사용할 수 있다.
4. break 는 반복을 무조건 종료시키고 continue 는 그 반복 몸체를 실행하지 않고 다음으로 넘어가는 것이다.

4 장 과제

2번

ch1.py - C:₩Users₩p0m0q₩AppData₩Local₩Programs₩Python₩Python38-32...

File Edit Format Run Options Window Help

```
from random import randint
m = 12000

for i in range(5):
    time = randint(35,51)
    if time > 40:
        w = (40 * m) + (time - 40) * (m * 1.5)
        w2 = int(w)
    else:
        w = time*m
        w2 = int(w)
    print('근로시간 ', time,'시간, 주급 ', w2)
```

Python 3.8.2 Shell

File Edit Shell Debug Options Window Help

```
Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb 25 2020, 22:45:29) [MSC v.1916 32 bit (In
tel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:₩Users₩p0m0q₩AppData₩Local₩Programs₩Python₩Python38-32₩0408₩ch1.py
근로시간  51 시간, 주급  678000
근로시간  51 시간, 주급  678000
근로시간  46 시간, 주급  588000
근로시간  36 시간, 주급  432000
근로시간  47 시간, 주급  606000
>>>
```

4번

ch2.py - C:₩Users₩p0m0q₩AppData₩Local₩Programs₩Python₩Python38-32...

File Edit Format Run Options Window Help

```
h, w = input('당신의 키(cm)와 몸무게(kg)는? >>').split()
h2 = float(h)
w2 = float(w)
print('키:',h2,'(cm), 몸무게:',w2,'(kg)')
bmi = w2 / (h2/100)**2

if  bmi >=40:
    print('bmi:%6.1f 고도 비만' % bmi)
elif bmi >=35:
    print('bmi:%6.1f 중등도 비만' % bmi)
elif bmi >=30:
    print('bmi:%6.1f 비만' % bmi)
elif bmi >=25:
    print('bmi:%6.1f 과체중' % bmi)
elif bmi >=18.5:
    print('bmi:%6.1f 정상' % bmi)
else:
    print('bmi:%6.1f 저체중' % bmi)
```

Python 3.8.2 Shell

File Edit Shell Debug Options Window Help

```
Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb 25 2020, 22:45:29) [MSC v.1916 32 bit (In
tel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:₩Users₩p0m0q₩AppData₩Local₩Programs₩Python₩Python38-32₩0408₩ch2.py
당신의 키(cm)와 몸무게(kg)는? >>171 72
키: 171.0 (cm), 몸무게: 72.0 (kg)
bmi:  24.6 정상
>>>
```

6번

ch3.py - C:/Users/p0m0q/AppData/Local/Programs/Python/Python38-32/0408...

File Edit Format Run Options Window Help

```
from random import randint

while True:
    r = randint(1,101)
    r2 = randint(1,101)
    print(r,'*',r2,'=',r*r2)
    print()
    a = input('계속 y / n ? ')
    if a == 'n':
        break
```

Python 3.8.2 Shell

File Edit Shell Debug Options Window Help

```
Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb 25 2020, 22:45:29) [MSC v.1916 32 bit (In
tel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/p0m0q/AppData/Local/Programs/Python/Python38-32/0408/ch3.py
54 * 4 = 216

계속 y / n ? y
33 * 83 = 2739

계속 y / n ? n
>>>
```

8번

ch4.py - C:/Users/p0m0q/AppData/Local/Programs/Python/Python38-32/0408...

File Edit Format Run Options Window Help

```
from random import randint
r = randint(1,101)
print('첫 값은',r,'이다')
c = ['+','-','*','/','%']
while True:

    a = input('산술 연산의 종류를 입력하세요 ( + , - , * , / , % ) >> ')
    if a not in c:
        print(' 종료 '.center(32,'*'))
        break
    b = int(input('두 번째 피연산자를 입력하세요 >> '))
    if a == '+':
        print(r,'+',b,'=',r+b)
    elif a == '-':
        print(r,'-',b,'=',r-b)
    elif a == '*':
        print(r,'*',b,'=',r*b)
    elif a == '/':
        print(r,'/',b,'=',r/b)
    elif a == '%':
        print(r'%',b,'=',r%b)
    print()
```

Python 3.8.2 Shell

File Edit Shell Debug Options Window Help

```
Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb 25 2020, 22:45:29) [MSC v.1916 32 bit (In
tel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/p0m0q/AppData/Local/Programs/Python/Python38-32/0408/ch4.py
첫 값은 25 이다
산술 연산의 종류를 입력하세요 ( + , - , * , / , % ) >> *
두 번째 피연산자를 입력하세요 >> 20
25 * 20 = 500

산술 연산의 종류를 입력하세요 ( + , - , * , / , % ) >> -
두 번째 피연산자를 입력하세요 >> 45
25 - 45 = -20

산술 연산의 종류를 입력하세요 ( + , - , * , / , % ) >> /
두 번째 피연산자를 입력하세요 >> 5
25 / 5 = 5.0

산술 연산의 종류를 입력하세요 ( + , - , * , / , % ) >> &
********** 종료 **********
>>>
```

제 5 장 리스트와 튜플

1. 리스트

1) 리스트의 개념과 생성

파이썬에서 여러 항목을 하나의 단위로 묶어 손쉽게 사용하는 복합 자료형이다. 리스트는 항목의 나열인 시퀀스이다. 콤마로 구분된 항목(원소)들을 리스트로 표현하며 각 항목은 모두 같은 자료형일 필요 없다. 항목 순서는 의미가 있고 항목 자료 값은 중복돼도 상관없다. 리스트는 대괄호 사이에 항목을 기술한다.

```
>>> fruit = ['사과', '귤', '바나나', '수박']
>>> print(fruit)
['사과', '귤', '바나나', '수박']
```

리스트도 시퀀스이므로 반복 for 문에서 사용할 수 있다.

```
fruit = ['사과', '귤', '바나나', '수박']
print(fruit)
print(type(fruit))

num = 0
for s in fruit:
    num+=1
    print('%d. %s' % (num,s))
['사과', '귤', '바나나', '수박']
<class 'list'>
1. 사과
2. 귤
3. 바나나
4. 수박
```

■ **빈 리스트의 생성**

빈 대괄호로 빈 리스트를 만들 수 있다.

```
>>> p1 = []
>>> print(p1)
[]
```

■ **len() 함수**

리스트의 항목 수를 알 수 있다.

```
>>> py = list('python')
>>> py
['p', 'y', 't', 'h', 'o', 'n']
>>> len(py)
6
```

■ **append() 메소드**

리스트의 메소드이다. 리스트의 가장 뒤에 항목을 추가해준다.

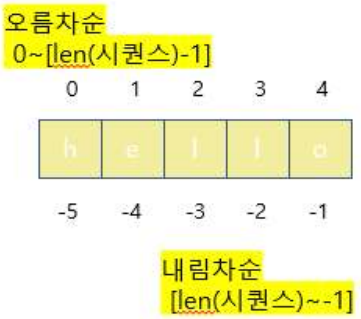
```
goods = []
for i in range(3):
    item = input('구입할 품목은 ? ')
    goods.append(item)
    print(goods)
print('길이: %d' % len(goods))
```

```
구입할 품목은 ? 과자
['과자']
구입할 품목은 ? 우유
['과자', '우유']
구입할 품목은 ? 아이스크림
['과자', '우유', '아이스크림']
길이: 3
```

2) 리스트의 항목 참조

■ **첨자로 리스트의 항목 참조**

문자열 시퀀스와 같이 첨자를 사용해 항목 하나하나를 참조할 수 있다. 첨자는 정수여야 한다.




```
>>> 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
```

★ 리스트로 가위 바위 보 게임 만들기

```
rsp = ['가위', '바위', '보']
for i in range(len(rsp)):
    print(rsp[i], end=' ')
print()

from random  import choice
print('컴퓨터의 가위 바위 보 5개')
for i in range(5):
    print(choice(rsp))
```

```
가위 바위 보
컴퓨터의 가위 바위 보 5개
가위
보
보
바위
가위
```

3) 리스트 항목 수정

■ **count(), index() 함수**

count() 함수 : 값을 갖는 항목 수 반환

index() 함수 : 인자의 값이 항목이 위치한 첨자를 반환

```
>>> top = ['BTS', '볼빨간사춘기', 'BTS', '블랙핑크', '태연', 'BTS']
>>> top.count('BTS')
3
>>> top.index('볼빨간사춘기')
1
>>> top.index('BTS') # 여러 개면 가장 처음 index반환
0
```

■ **첨자로 항목 수정**

리스트의 첨자를 이용한 항목을 대입 연산자의 오른쪽에 위치 시켜 리스트의 항목을 수정할 수 있다. 적어도 하나 이상의 항목이 있는 경우에 수정할 수 있다.

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

첨자 1 인 볼빨간사춘기가 장범준으로 수정되고 첨자 3 인 블랙핑크가 잔나비로 수정됐다.

4) 중첩 리스트

■ **리스트 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
['사자', '코끼리', '호랑이']
>>> print(animal[0][1]) # 0첨자의 1항목
코끼리
```

2. 리스트 부분 참조와 항목의 삽입과 삭제

1) 부분 참조 슬라이싱

■ **첨자 3 개로 리스트 일부분을 참조하는 슬라이싱**

리스트[start:stop:step]로 사용하며 첨자 start 에서 stop-1 까지 step 이 간격의 요소로 구성된 부분 리스트를 반환한다. start 와 stop 은 생략하면 처음부터 마지막 항목까지 참조하고 steep 은 생략하면 1 이다. step 이 -1 이면 역순으로 반환한다.

```
>>> alp = list('abcdefghij')
>>> 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']
```

■ **0 부터 시작되는 순차 첨자**

alp[1:5]는 첨자 1 에서 4 까지의 항목을 반환하고 alp[1:10:2]는 1 부터 9 까지 2 씩 증가하는 항목을 반환한다.

```
>>> print(alp[1:5])
['b', 'c', 'd', 'e']
>>> print(alp[1:10:2])
['b', 'd', 'f', 'h', 'j']
```

■ **-1 부터 시작되는 역순첨자**

alp[-2:-9:-1]은 역순으로 첨자 -2 에서 -8 까지의 항목을 반환하고 alp[-3::-2]는 -3 부터 끝까지 2 씩 감소하는 항목을 반환한다.

```
>>> print(alp[-2:-9:-1])
['i', 'h', 'g', 'f', 'e', 'd', 'c']
>>> print(alp[-3::-2])
['h', 'f', 'd', 'b']
```

■ **혼합한 슬라이싱**

alp[1:-1]은 첨자 1 에서 첨자-1 까지 항목을 반환하고 alp[-1:1:-1]은 첨자 -1 부터 첨자 -2 까지 역순으로 항목을 반환한다.

```
>>> 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']
```

2) 리스트 부분 수정

■ **리스트의 슬라이스로 리스트의 일부분을 수정**

리스트의 일부분을 다른 리스트로 수정하려면 슬라이스 방식에 대입한다.

```
>>> fruit = ['사과', '귤', '바나나', '수박']
>>> fruit[0:2] = ['오렌지'] #0첨자부터 1첨자(2-1)까지 오렌지를 대입
>>> print(fruit)
['오렌지', '바나나', '수박']
```

3) 리스트의 항목 삽입과 삭제

■ **insert() 리스트 메소드**

리스트의 첨자 위치에 항목을 삽입하기 위해서 리스트.insert(첨자, 항목)을 이용한다. 빈리스트에도 삽입할 수 있다.

```
>>> fruit = []
>>> fruit.insert(0,'사과')
>>> fruit.insert(0,'귤')
>>> fruit.insert(1,'바나나')
>>> fruit
['귤', '바나나', '사과']
```

■ **remove(항목), pop(첨자), pop()로 항목 삭제**

remove() : 리스트에서 지정된 값의 항목을 삭제한다.

pop() : 마지막 항목을 삭제하고 삭제된 값을 반환하고 첨자를 사용하면 지정된 첨자의 항목을 삭제하고 반환한다.

리스트에서 하나의 항목을 삭제하는 경우 사용한다.

```
>>> fruit = ['사과', '귤', '바나나', '수박', '파인애플']
>>> fruit.remove('귤')
>>> print(fruit)
['사과', '바나나', '수박', '파인애플']
>>> print(fruit.pop())
파인애플
>>> print(fruit.pop(2))
수박
>>> print(fruit)
['사과', '바나나']
```

■ **문장 del()**

뒤에 위치한 변수나 항목이 삭제된다. 슬라이스로 리스트의 일부분을 삭제할 수도 있다.

```
>>> fruit = ['사과', '귤', '바나나', '수박', '파인애플']
>>> del fruit[0]
>>> print(fruit)
['귤', '바나나', '수박', '파인애플']
>>> del fruit[0:2]
>>> print(fruit)
['수박', '파인애플']
```

■ **clear() 메소드**

리스트의 모든 항목을 제거해준다.

```
>>> fruit = ['사과', '귤', '바나나', '수박', '파인애플']
>>> fruit.clear()
>>> print(fruit)
[]
```

★ 리스트 항목 삽입과 삭제해 보기

```
item = ['연필', '볼펜']
# 현재 학용품 품목 출력
print(item)

# 연필 1개와 볼펜 세 자루 삽입
item.insert(1, 2)
item.insert(3, 3)
# 맨 뒤에 지우개, 1개 삽입
item.insert(4, '지우개')
item.insert(5, 1)
# 현재 학용품 품목 출력
print(item)

# 연필 두 자루 삭제
print(item.pop(1)) # 첨자 1 항목 삭제
item.remove('연필') # 항목 연필 항목 삭제
del item[2:] # 지우개와 수 품목 삭제

# 현재 학용품 품목 출력
print(item)
```

```
['연필', '볼펜']
['연필', 2, '볼펜', 3, '지우개', 1]
2
['볼펜', 3]
```

4) 리스트 추가, 연결, 반복

■ **extend() 메소드**

리스트에 인자인 list 를 가장 뒤에 추가한다.

```
>>> day = ['월', '화', '수']
>>> day2 = ['목', '금', '토', '일']
>>> day.extend(day2)
>>> print(day)
['월', '화', '수', '목', '금', '토', '일']
```

■ **+ 연산자**

리스트와 리스트를 연결해준다.

```
>>> fruit1 = ['사과', '귤']
>>> fruit2 = ['오렌지', '수박']
>>> fruit3 = fruit1 + fruit2
>>> print(fruit3)
['사과', '귤', '오렌지', '수박']
```

5) 리스트 항목 순서와 정렬

■ **reverse() 메소드**

리스트 항목의 순서를 뒤집는다.

```
>>> one = '잣밤배귤감'
>>> wlist = list(one)
>>> print(wlist)
['잣', '밤', '배', '귤', '감']
>>> wlist.reverse()
>>> print(wlist)
['감', '귤', '배', '밤', '잣']
```

■ **sort() 메소드**

리스트 항목의 순서를 기준에 따라 재배열한다. sort()는 리스트를 오름차순으로 정렬하고 sort(reverse=True)는 내림차순으로 정렬한다.

```
>>> one = '잣밤배귤감'
>>> wlist = list(one)
>>> wlist.sort()
>>> print(wlist)
['감', '귤', '밤', '배', '잣']
>>> wlist.sort(reverse=True)
>>> print(wlist)
['잣', '배', '밤', '귤', '감']
```

■ **sorted() 내장 함수**

리스트의 항목 순서를 오름차순으로 정렬한 새로운 리스트를 반환한다. 반환 값을 새 변수에 대입해 사용할 수 있다. 원래 리스트 자체는 변하지 않는다.

```
>>> fruit = ['사과', '귤', '바나나', '수박', '파인애플']
>>> s_fruit = sorted(fruit)
>>> print(fruit)
['사과', '귤', '바나나', '수박', '파인애플']
```

★ extend()와 sort(), sorted() 같이 사용해보기

```
word = list('삶꿈정')
word.extend('복빛')
print(word)
word.sort()
print(word)

fruit = ['복숭아', '자두', '골드키위', '귤']
print(fruit)
fruit.sort(reverse=True)
print(fruit)

mix = word + fruit
print(sorted(mix))
print(sorted(mix, reverse=True))
```

```
['삶', '꿈', '정', '복', '빛']
['꿈', '복', '빛', '삶', '정']
['복숭아', '자두', '골드키위', '귤']
['자두', '복숭아', '귤', '골드키위']
['골드키위', '귤', '꿈', '복', '복숭아', '빛', '삶', '자두', '정']
['정', '자두', '삶', '빛', '복숭아', '복', '꿈', '귤', '골드키위']
...
```

6) 리스트 컴프리헨션(매우 중요!)

■ **조건이 만족한 항목으로 리스트를 생성하는 컴프리헨션**

리스트를 만드는 간결한 방법을 제공한다. 대괄호로 감싸고 구성항목을 가장 앞에 배치하며 for 문의 range 뒤에 콜론이 빠진다.

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

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

더 간결하게 사용 가능

```
>>> even = [i for i in range(2,11,2)]
>>> print(even)
[2, 4, 6, 8, 10]
...
```

■ **조건이 있는 컴프리헨션**

변수의 조건을 뒤에 붙이고 조건에서 콜론이 빠진다.

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

★ 컴프리헨션으로 간단하게 리스트 만들기

```
# for문으로 리스트 생성
a = []
for i in range(10):
    a.append(i)
print(a)

# 컴프리헨션으로 리스트 생성
seq = [i for i in range(10)]
print(seq)

# for문으로 리스트 생성
s = []
for i in range(10):
    if i%2 == 1:
        s.append(i**2)
print(s)

# 컴프리헨션으로 리스트 생성
squares = [i**2 for i in range(10) if i%2 == 1]
print(squares)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[1, 9, 25, 49, 81]
[1, 9, 25, 49, 81]
```

7) 리스트 대입과 복사

■ **리스트 대입에 의한 동일 리스트 공유 (얕은 복사)**

리스트에서 대입 연산자 =은 얕은 복사라고 해서 대입되는 변수가 동일한 시퀀스를 가리킨다. 기존 변수의 리스트를 대입 연산자로 받은 새로운 변수가 있으면 새로운 변수는 기존 변수의 리스트를 가리킨다. 결국 둘은 같은 리스트이다. 이런 경우 새로운 변수를 pop 으로 마지막 항목을 삭제하면 두 개의 변수의 리스트 마지막 항목이 삭제된다.

```
>>> f1 = ['사과', '귤' ,'오렌지', '수박']
>>> f2 = f1
>>> f2.pop()
'수박'
>>> print(f1)
['사과', '귤', '오렌지']
>>> print(f2)
['사과', '귤', '오렌지']
```

■ **리스트 대입에 의한 새로운 리스트 생성 (깊은 복사)**

리스트에 새로운 리스트를 만들어서 복사할 때는 슬라이스[:]나 copy() 또는 list() 함수를 이용한다. 복사할 변수 = 기존변수[:]를 사용하면 기존 변수와 같은 새로운 리스트가 된다. 이런 복사를 깊은 복사라고 한다.

```
>>> f1 = ['사과', '귤' ,'오렌지', '수박']
>>> f2 = f1[:]
>>> f2.pop(1)
'귤'
>>> print(f1,f2)
['사과', '귤', '오렌지', '수박'] ['사과', '오렌지', '수박']
```

같은 결과를 얻는 다른 방법

```
>>> f1 = ['사과', '귤' ,'오렌지', '수박']
>>> f3 = f1.copy()
>>> f3.pop()
'수박'
>>> print(f1,f3)
['사과', '귤', '오렌지', '수박'] ['사과', '귤', '오렌지']
>>> f4 = list(f1)
>>> f4.append('감')
>>> print(f1,f4)
['사과', '귤', '오렌지', '수박'] ['사과', '귤', '오렌지', '수박', '감']
```

■ 문장 is

피연산자인 변수 2 개가 동일한 메모리를 공유하는지 검사한다.

(같으면 True, 다르면 False 반환)

```
>>> f1 = ['사과', '귤' ,'오렌지', '수박']
>>> f2 = f1
>>> print(f1 is f2)
True
>>> f3 = f1[:]
>>> print(f1 is f3)
False
```

3. 수정 불가능한 튜플

1) 튜플

튜플은 모두 콤마로 구분된 항목들의 리스트로 표현되며, 각각의 항목은 정수, 실수, 문자열, 리스트, 튜플 등 제한이 없다.

■ **튜플의 생성**

빈 튜플은 ()로 만든다. 함수 tuple()로도 만들 수 있다.

```
>>> empty1 = ()
>>> print(empty1)
()
>>> empty2 = tuple()
>>> print(empty2)
()
```

튜플은 항목이 하나인 것도 가능한데 하나의 튜플을 표현할 때는 마지막에 반드시 콤마(,)를 붙여야 한다.

```
>>> data1 = 1 #정수
>>> data2 = 1, #튜플
>>> print(data1,data2)
1 (1,)
```

■ **sorted() 내장함수**

튜플 항목의 순서를 오름차순으로 정렬한 새로운 리스트를 반환한다. 반환 값을 리스트 변수에 대입해 사용할 수 있다. sorted(튜플,reverse=True)로 호출하면 역순인 내림차순으로 정렬된 리스트를 반환한다. 절대 튜플 자체가 수정되지 않는다.

5 장 요점정리!

1. 리스트는 문자열처럼 첨자를 사용해서 참조할 수 있다. 리스트 안에 리스트 사용이 가능하다. 또한 슬라이싱으로 참조도 가능하다.
2. 리스트 컴프리헨션. 매우 중요함. for 문을 아주 간결하게 사용할 수 있게 해준다. 조건이 있는 경우도 사용할 수 있다.
3. 튜플은 항목의 순서나 내용을 수정할 수 없다.

5 장 문제 풀어 보기

```
#1번

import random

a=[]

for i in range(10):

    b=random.randint(1,99)

    a.append(b)
print('리스트 : ',a)
a.sort()
print('정렬 리스트 : ', a)
a.sort(reverse=True)
print('역순 리스트 : ', a)
```

```
리스트 :  [49, 21, 21, 28, 56, 94, 24, 96, 83, 79]
정렬 리스트 :  [21, 21, 24, 28, 49, 56, 79, 83, 94, 96]
역순 리스트 :  [96, 94, 83, 79, 56, 49, 28, 24, 21, 21]
```

```
# 4번

data = [[1, 2, 3],
        [4, 5, 6],
        [7, 8, 9]]

sum1, sum2 = [], [0] * len(data[0])

for i in data:
    sum1.append(sum(i))
    for j, k in enumerate(i):
        sum2[j] += k

print('각 행의 합 : ',sum1)
print('각 열의 합 : ',sum2)
```

```
각 행의 합 :  [6, 15, 24]
각 열의 합 :  [12, 15, 18]
```

```
# 6번

a = [[1,2], [3,4], [5,6], [7,8]]
print('원 행렬(m) 출력:')
for i in a:

    print(i)

print()
print('전차 행렬 출력:')
for j in range(len(a[0])):
    for i in range(len(a)):
        print(a[i][j], end = ' ')
    print()
```

```
원 행렬(m) 출력:
[1, 2]
[3, 4]
[5, 6]
[7, 8]

전차 행렬 출력:
1 3 5 7
2 4 6 8
```

제 6 장 딕셔너리와 집합

1. 딕셔너리

1) 딕셔너리의 개념과 생성

딕셔너리란 키와 값의 쌍인 항목을 나열한 시퀀스다. 콤마로 구분된 항목(또는 요소, 원소)들의 리스트로 표현된다. 각각의 항목은 키:값과 같이 콜론으로 구분하고 전체는 중괄호로 묶는다.

딕셔너리는 항목 순서는 의미가 없으며 키는 중복될 수 없다. 키는 중복될 수 없지만, 값은 수정될 수 있으며 값은 키로 참조된다.

```
>>> mycar = {"brand": "현대", "model" : "제네시스", "year" : 2016}
>>> print(mycar)
{'brand': '현대', 'model': '제네시스', 'year': 2016}
```

■ **빈 딕셔너리 생성과 항목 추가**

빈 중괄호로 빈 딕셔너리를 만들 수 있다. 출력하면 빈 딕셔너리인 {}로 표시된다.

인자가 없는 dict() 내장 함수 호출로도 빈 딕셔너리를 생성할 수 있다. 항목을 딕셔너리에 넣으려면 '딕셔너리[키] = 값'의 문장을 사용해야 한다. 딕셔너리 항목 값으로 리스트나 튜플도 가능하다.

```
>>> lect = {}
>>> print(lect)
{}
>>> lect = dict()
>>> lect['강좌명'] = '파이썬 기초';
>>> lect['개설년도'] = [2020,1];
>>> print(lect)
{'강좌명': '파이썬 기초', '개설년도': [2020, 1]}
```

2) dict() 함수로 생성하는 딕셔너리

■ **리스트 또는 튜플로 구성된 키-값을 인자로 사용**

내장 함수 dict() 함수에서 인자로 리스트나 튜플 1 개를 사용해 딕셔너리를 만들 수 있다. 함수 dict()의 리스트나 튜플 내부에서 일련의 키-값 쌍으로 [키, 값] 리스트 형식과 (키, 값) 튜플 형식에서 모두 사용할 수 있다.

```
>>> day = dict([]) #빈 딕셔너리
>>> day = dict(()) #빈 딕셔너리
>>> day = dict([['월', 'monday'], ['화', 'tuesday'], ['수', 'wednesday']])
>>> day = dict([('월', 'monday'), ('화', 'tuesday'), ('수', 'wednesday')])
>>> day = dict((['월', 'monday'], ['화', 'tuesday'], ['수', 'wednesday']))
>>> day = dict((('월', 'monday'), ('화', 'tuesday'), ('수', 'wednesday')))
>>> print(day) # 전부 사용 가능
{'월': 'monday', '화': 'tuesday', '수': 'wednesday'}
```

■ **키가 문자열인 경우**

키가 단순 문자열이면 간단히 월='Monday'처럼 키=값 항목 나열로도 지정할 수 있다.

```
>>> print(day) # 키가 문자열이면 따옴표 없이 기술할 수 있다.
{'월': 'monday', '화': 'tuesday', '수': 'wednesday'}
```

■ **정보를 저장하는 다양한 딕셔너리 방법**

```
bts1={'그룹명': '방탄소년단', '인원수': 7, '리더': '김남준'}
bts1['소속사']='빅히트 엔터테인먼트';
print(bts1)
bts2=dict([['그룹명', '방탄소년단'], ['인원수', 7]])
print(bts2)
bts3=dict((('리더', '김남준'), ('소속사', '빅히트 엔터테인먼트')))
print(bts3)

bts=dict(그룹명='방탄소년단', 인원수=7, 리더='김남준', 소속사='빅히트 엔터테인먼트')
#구성원 추가
bts['구성원']=['RM', '진', '슈가', '제이홉', '지민', '뷔', '정국']

print(bts) #전체 출력
print(bts['구성원']) #구성원 출력

{'그룹명': '방탄소년단', '인원수': 7, '리더': '김남준', '소속사': '빅히트 엔터테인먼트'}
{'그룹명': '방탄소년단', '인원수': 7}
{'리더': '김남준', '소속사': '빅히트 엔터테인먼트'}
{'그룹명': '방탄소년단', '인원수': 7, '리더': '김남준', '소속사': '빅히트 엔터테인먼트',
'구성원': ['RM', '진', '슈가', '제이홉', '지민', '뷔', '정국']}
['RM', '진', '슈가', '제이홉', '지민', '뷔', '정국']
```

3) 딕셔너리 키는 수정 불가능한 객체로 사용

■ **딕셔너리의 키로 정수, 실수 등 사용 가능**

딕셔너리의 키는 수정 불가능한 객체는 모두 가능하다. 정수는 물론 실수도 가능하다.

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

만약 기존에 없는 키를 입력하면 없던 키이므로 새로운 항목으로 삽입된다.

```
>>> real[2.72] = '오일러수'
>>> print(real)
{3.14: '원주율', 2.71: '자연수', 2.72: '오일러수'}
```

키는 정수도 가능하며 그 정수를 불러서 출력할 때는 대괄호 안에 넣어서 출력한다.

```
>>> month = {1:'January', 2:'February', 3:'March'}
>>> print(month[2]) # 첨자가 아니라 키를 의미
February
```

■ **튜플은 키로 가능, 리스트는 불가능**

수정 불가능한 튜플은 딕셔너리의 키로 사용될 수 있다. 그러나 수정 가능한 리스트는 키로 사용할 수 없다.

4) 딕셔너리 항목의 순회

■ **keys() 메소드**

키로만 구성된 리스트를 반환한다.

```
>>> day = dict(월='monday', 화='tuesday', 수='wednesday')
>>> print(day.keys())
dict_keys(['월', '화', '수'])
```

■ **items() 메소드**

(키, 값) 쌍의 튜플이 들어 있는 리스트를 반환한다. 각 튜플의 첫 번째 항목은 키, 두 번째 항목은 키 값이다.

```
>>> day = dict(월='monday', 화='tuesday', 수='wednesday')
>>> print(day.items())
dict_items([('월', 'monday'), ('화', 'tuesday'), ('수', 'wednesday')])
```

for 문에서 변수 위치에 키 값을 저장할 2 개의 변수와 시퀀스 위치에 메소드 items()를 사용하면 딕셔너리의 모든 항목을 참조하는 간단한 구문을 사용할 수 있다.

```
>>> for key, value in day.items():
        print('%s요일 %s' % (key, value))


월요일 monday
화요일 tuesday
수요일 wednesday
```

■ **values() 메소드**

값으로 구성된 리스트를 반환한다.

```
>>> day = dict(월='monday', 화='tuesday', 수='wednesday')
>>> print(day.values())
dict_values(['monday', 'tuesday', 'wednesday'])
```

■ **반복 for 문에서 딕셔너리 변수로만 모든 키 순회**

반복 for 문에서는 시퀀스 위치에 있는 딕셔너리 변수만으로도 모든 키를 순회할 수 있다.

```
>>> day = dict(월='monday', 화='tuesday', 수='wednesday')
>>> for key in day:
        print('%s: %s' % (key, day[key]))


월: monday
화: tuesday
수: wednesday
```

★ 딕셔너리로 영어 사전 만들어보기

```
season={'봄': 'spring', '여름': 'summer', '가을': 'autumn', '겨울':'winter'}
print(season.keys())
print(season.items())
print(season.values())

#메소드 keys()로 항목순회
for key in season.keys():
    print('%s %s  ' % (key, season[key]))

for item in season.items():
    print('{} {}  '.format(item[0], item[1]), end=' ')
print()
#메소드 items()의 반환 값인 튜플을 한 변수에 저장한 경우, 항목 순회 2
for item in season.items():
    print('{} {}  '.format(*item), end=' ')
print()
```

```
dict_keys(['봄', '여름', '가을', '겨울'])
dict_items([('봄', 'spring'), ('여름', 'summer'), ('가을', 'autumn'), ('겨울', 'winter')])
dict_values(['spring', 'summer', 'autumn', 'winter'])
봄 spring
여름 summer
가을 autumn
겨울 winter
봄 spring   여름 summer   가을 autumn   겨울 winter
봄 spring   여름 summer   가을 autumn   겨울 winter
```

5) 딕셔너리 항목의 참조와 삭제

■ **get() 메소드**

키의 해당 값을 반환한다. 만약 딕셔너리에 키가 없어도 오류가 발생하지 않고 아무것도 없다는 의미인 None 을 반환한다. 만일 키 뒤에 다른 인자를 넣으면 키가 없을 때 그 값이 반환된다.

```
>>> city = {'대한민국':'부산', '뉴질랜드':'웰링톤', '캐나다':'몬트리올'}
>>> city.get('대한민국')
'부산'
>>> city.get('미국', '없네요')
'없네요'
```

■ **pop() 메소드**

키인 항목을 삭제하고 삭제되는 키의 해당 값을 반환한다. 만약 삭제할 키가 없다면 두 번째에 지정한 값이 반환된다. 인자로 키만 적었는데 삭제할 키가 없다면 KeyError 가 발생한다.

```
>>> city = {'대한민국':'부산', '뉴질랜드':'웰링톤', '캐나다':'몬트리올'}
>>> print(city.pop('중국', '없네요'))
없네요
>>> print(city.pop('중국'))
Traceback (most recent call last):
  File "<pyshell#63>", line 1, in <module>
    print(city.pop('중국'))
KeyError: '중국'
```

■ **popitem() 메소드**

임의의 (키, 값)의 튜플을 반환하고 삭제한다. 만약 데이터가 하나도 없다면 오류가 발생한다.

```
>>> city = {'대한민국':'부산', '뉴질랜드':'웰링톤'}
>>> print(city.popitem())
('뉴질랜드', '웰링톤')
>>> city
{'대한민국': '부산'}
>>> print(city.popitem())
('대한민국', '부산')
>>> city
{}
>>> print(city.popitem())
Traceback (most recent call last):
  File "<pyshell#69>", line 1, in <module>
    print(city.popitem())
KeyError: 'popitem(): dictionary is empty'
```

■ **문장 del() 항목 삭제**

문장 del 에 이어 키로 지정하면 해당 항목이 삭제된다.

```
>>> city = {'대한민국':'부산', '뉴질랜드':'웰링톤'}
>>> del city['뉴질랜드']
>>> city
{'대한민국': '부산'}
```

6) 딕셔너리 항목 전체 삭제와 변수 제거

■ **clear() 메소드**

기존의 모든 키:값 항목을 삭제한다. 리스트에도 존재하지만, 튜플에는 존재하지 않는다.

```
>>> city = {'대한민국':'부산', '뉴질랜드':'웰링톤', '캐나다':'몬트리올'}
>>> city.clear()
>>> city
{}
```

■ **문장 del() 딕셔너리 변수 자체 제거**

문장 del 에 이어 키로 지정하면 변수 자체가 메모리에서 제거된다. 제거 이후에는 변수를 참조할 수 없다.

```
>>> city = {'대한민국':'부산', '뉴질랜드':'웰링톤'}
>>> del city
>>> city
Traceback (most recent call last):
  File "<pyshell#78>", line 1, in <module>
    city
NameError: name 'city' is not defined
```

7) 딕셔너리 결합과 키의 멤버십 검사 연산자 in

■ **update() 메소드**

인자인 다른 딕셔너리를 합병한다. 인자 딕셔너리에 원 딕셔너리와 동일한 키가 있다면 인자 딕셔너리의 값을 대체된다.

```
>>> 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}
>>> usstock.update({'MS':200})
>>> usstock
{'MS': 200, 'Apple': 180}
```

■ **문장 in**

딕셔너리에 키가 존재하는지 간단히 검사할 수 있다. 값의 존재 여부는 확인할 수 없으므로 값으로 조회하면 항상 False 이다. not in 도 가능하다.

```
>>> cp = {'한국':'서울', '중국':'북경', '독일':'베를린'}
>>> '한국' in cp
True
>>> 'USA' in cp
False
>>> if '미국' not in cp:
        cp['미국'] = '워싱턴'


>>> cp
{'한국': '서울', '중국': '북경', '독일': '베를린', '미국': '워싱턴'}
```

2. 중복과 순서가 없는 집합

1) 집합을 처리하는 자료형

■ **원소는 유일하고 순서는 의미 없는 집합**

집합은 중복되는 요소가 없으며, 순서도 없는 원소의 모임이다.

파이썬에서 집합은 수학과 같이 원소를 콤마로 구분하며 중괄호로 둘러싸 표현한다.

원소는 불변 값으로 중복될 수 없으며 서로 다른 값이어야 한다.

집합의 원소는 정수, 실수, 문자열, 튜플 등 수정 불가능한 것이어야 하며, 리스트나 딕셔너리처럼 가변적인 것은 허용되지 않는다. 집합의 원소는 중복을 허용하지 않기 때문에 멤버십 검사와 중복제거에 사용된다.

```
>>> s1 = {1, 2, 3, 4, 5}
>>> s2 = {'py', 'java', 'go'}
>>> s3 = {(1,1), (2, 4), (3,9)}
```

2) 내장 함수 set()을 활용한 집합 생성

- **set() 내장함수**

  집합을 생성할 수 있다. 인자는 리스트와 튜플, 문자열처럼 반복적이면 가능하다. 그러나 리스트나 튜플의 항목은 변할 수 없는 것이어야 하며, 항목이 리스트나 딕셔너리처럼 가변적인 것은 허용되지 않는다.

  set(원소로 구성된 리스트_or_튜플_or_문자열)

  인자가 없으면 빈 집합인 공집합이 생성되며 인자가 있으면 하나이며, 리스트, 튜플, 문자열 들이 올 수 있다.

- **set() 함수 (공집합)**

  공집합을 만들 수 있으며 집합의 자료형은 클래스 set 이다.

  빈 리스트 = [ ], 빈 튜플 = ( ), 빈 딕셔너리 = { }

- **set() 함수 (리스트, 튜플, 문자열을 인자로 사용)**

  인자로 리스트나 튜플 자체를 사용할 수 있으며 결과는 시퀀스 항목에서 중복을 제거한 원소로 구성된다. 중괄호 안에 원소가 콤마로 구분돼 표시된다.

  인자로 문자열이 사용되면 각각의 문자가 원소인 집합이 생성된다. 순서는 의미가 없다.

```
>>> set([1, 2, 3])
{1, 2, 3}
>>> set((1, 2, 2)) # 중복 제거
{1, 2}
>>> set(['a', 'b']) # 순서 상관없음
{'b', 'a'}
```

- **리스트와 딕셔너리는 집합의 원소로 사용 불가**

  리스트나 튜플의 항목으로 수정될 수 있는 리스트나 딕셔너리는 허용되지 않고 TypeError 발생한다.

```
>>> set([1, [2, 3]]) #오류
Traceback (most recent call last):
  File "<pyshell#115>", line 1, in <module>
    set([1, [2, 3]]) #오류
TypeError: unhashable type: 'list'
```

3) 중괄호로 직접 집합 생성

- **{원소 1, 원소 2,..} 로 생성**

  중괄호 안에 직접 원소를 콤마로 구분해 나열하는 방법이다. 집합의 원소는 문자, 문자열, 숫자, 튜플 등과 같이 변할 수 없는 것이어야 한다.

```
>>> {1, 2, 3}
{1, 2, 3}
>>> {1, 'seoul', 'a', (1.2, 3.4)}
{1, 'seoul', (1.2, 3.4), 'a'}
```

■ **{원소 1, 원소 2,..}에서 리스트나 딕셔너리는 원소로 사용 불가**

리스트나 딕셔너리 같이 가변적인 것은 원소로 사용할 수 없다.

```
>>> {['a', 'b']}
Traceback (most recent call last):
  File "<pyshell#118>", line 1, in <module>
    {['a', 'b']}
TypeError: unhashable type: 'list'
>>> {{1:'a', 2:'b'}}
Traceback (most recent call last):
  File "<pyshell#119>", line 1, in <module>
    {{1:'a', 2:'b'}}
TypeError: unhashable type: 'dict'
```

4) 집합의 원소 추가와 삭제

■ **add() 메소드**

집합에 원소를 추가할 수 있다.

```
>>> odd = {1, 3, 5}
>>> odd.add(7)
>>> print(odd)
{1, 3, 5, 7}
```

■ **remove(), discard(), pop() 메소드**

원소를 삭제한다. remove() 메소드는 삭제하려는 원소가 없으면 KeyError 를 발생하고 discard() 메소드는 오류가 발생하지 않는다. pop() 메소드는 임의의 원소를 삭제한다.

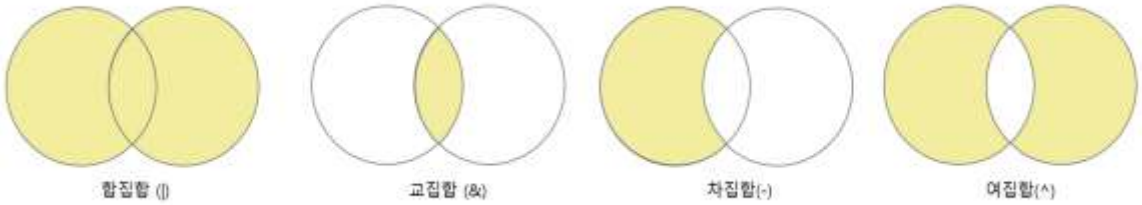
```
>>> odd = {1, 3, 5, 9, 11}
>>> odd.remove(3)
>>> print(odd)
{1, 5, 9, 11}
>>> odd.remove(2)
Traceback (most recent call last):
  File "<pyshell#134>", line 1, in <module>
    odd.remove(2)
KeyError: 2
>>> odd.discard(1)
>>> print(odd)
{5, 9, 11}
>>> odd.discard(2)
>>> print(odd.pop())
5
>>> print(odd)
{9, 11}
```

■ **clear() 메소드**

집합의 모든 원소를 삭제한다.

```
>>> odd = {1, 3, 5}
>>> odd.clear()
>>> print(odd)
set()
```

5) 합집합, 교집합, 차집합, 여집합



■ **합집합 연산자 |와 union(), update() 메소드**

양쪽 모두의 원소를 합하는 합집합은 연산자 |와 union() 메소드를 사용한다.

union() 메소드 : 합집합을 반환하며 a 자체가 수정되지 않는다.

update() 메소드 : 합집합과 같은 효과가 있으며 a 와 b 의 합집합 결과가 호출하는 집합에 반영돼 수정된다.

```
>>> 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}
```

■ **교집합 연산자 &와 intersection() 메소드**

양쪽 모든 집합에 속하는 원소로 구성되는 교집합을 연산자 &와 intersection() 메소드를 사용한다. intersection()은 a, b 모두에 영향을 미치지 않지만, intersection_update()는 a 를 교집합으로 수정한다.

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

■ **차집합 연산자 -와 difference() 메소드**

피연산자인 왼쪽 집합에는 있지만, 오른쪽에는 없는 원소로 구성되는 차집합은 연산자 -와 difference() 메소드를 사용한다. 피연산자의 순서에 따라 결과가 바뀌므로 교환 법칙이 성립하지 않는다. 4 개의 집합 연산 중 유일하게 교환 법칙이 성립하지 않는다.

```
>>> a = {4, 6, 8, 10, 12}
>>> b = {3, 6, 9, 12}
>>> a - b
{8, 10, 4}
>>> a.difference(b)
{8, 10, 4}
>>> b - a
{9, 3}
```

■ **여집합 연산자 ^와 symmetric_difference() 메소드**

한쪽 집합에는 소속되지만, 교집합이 아닌 원소로 구성된다. 대칭 차집합이라고도 부르며, 연산자 ^와 symmetric_difference() 메소드를 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}
```

■ **집합 연산의 축약 대입 연산자 |=, &= -=, ^=와 intersection_update() 메소드**

산술 연산처럼 집합 연산 네 가지도 축약 대입 연산자가 가능하다.

합집합 |= 연산자

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

교집합 &= 연산자

```
>>> A = set('abcd'); B = set('cde')
>>> A &= B
>>> A # intersection_update() 메소드와 같은 결과
{'d', 'c'}
```

차집합 -= 연산자

```
>>> A = set('abcd'); B = set('cde')
>>> A -= B
>>> A # difference_update() 메소드와 같은 결과
{'a', 'b'}
```

여집합 ^= 연산자

```
>>> A = set('abcd'); B = set('cde')
>>> A ^= B
>>> A # symmetric_difference_update() 메소드와 같은 결과
{'a', 'e', 'b'}
```

6) len() 함수와 소속 연산 in

■ **len() 함수**

집합에 들어 있는 원소의 개수를 확인할 수 있다.

```
>>> p1 = {'python','java'}
>>> len(p1)
2
```

■ **멤버십 연산자 in**

특정 원소가 집합에 있는지 확인할 수 있다.

```
>>> 'python' in p1
True
>>> 'c' in p1
False
```

3. 내장 함수 zip()과 enumerate(), 시퀀스 간의 변환

1) zip() 내장 함수

■ **리스트나 튜플 항목으로 조합된 항목을 생성하는 zip() 내장 함수**

동일한 수로 이뤄진 여러 개의 튜플 항목 시퀀스를 각각의 리스트로 묶어주는 역할을 한다. zip 의 결과를 내장 함수 list()의 인자로 사용하면 항목이 튜플인 리스트가 생성된다.

```
>>> a = ['FTP', 'telnet', 'SMTP', 'DNS']
>>> b = (20, 30, 25, 53)
>>> z = zip(a, b)
>>> list(zip(a, b))
[('FTP', 20), ('telnet', 30), ('SMTP', 25), ('DNS', 53)]
```

zip()을 tuple()의 인자로 사용하면 항목이 튜플인 튜플이 생성된다.

```
>>> tuple(zip(a,b))
(('FTP', 20), ('telnet', 30), ('SMTP', 25), ('DNS', 53))
```

zip()에서는 인자의 수는 2 개 이상 올 수 있다.

```
>>> list(zip('ABCD',a, b))
[('A', 'FTP', 20), ('B', 'telnet', 30), ('C', 'SMTP', 25), ('D', 'DNS', 53)]
```

zip()의 인자들의 길이가 맞지 않으면 짧은 쪽의 인자에 맞는 리스트를 구성하고 긴 것은 무시한다.

```
>>> tuple(zip('abcd', 'xy'))
(('a', 'x'), ('b', 'y'))
```

■ 2 개의 리스트나 튜플로 키-값 항목인 딕셔너리를 생성하는 내장 함수 zip()

기존 리스트나 튜플의 조합으로 딕셔너리를 간단하게 만들 수 있다.

```
>>> a = ['FTP', 'telnet', 'SMTP', 'DNS']
>>> b = (20, 30, 25, 53)
```

내장 함수 dict()에서 zip()의 인자 2 개를 사용하면, 앞은 키, 뒤는 값의 조화로 구성돼 딕셔너리가 생성된다.

```
>>> dict(zip(a, b))
{'FTP': 20, 'telnet': 30, 'SMTP': 25, 'DNS': 53}
```

함수 zip() 인자들의 길이가 같지 않으면 짧은 쪽에 인자의 맞는 키:값의 항목을 구성하고 긴 부분은 무시한다.

```
>>> dict(zip('abcd', 'xy'))
{'a': 'x', 'b': 'y'}
```

딕셔너리는 키:값의 항목이므로 인자가 2 개여야 한다. 아니면 오류가 난다.

```
>>> dict(zip(a, b, b))
Traceback (most recent call last):
  File "<pyshell#198>", line 1, in <module>
    dict(zip(a, b, b))
ValueError: dictionary update sequence element #0 has length 3; 2 is required
```

2) 내장 함수 enumerate()

■ **enumerate() 내장 함수**

첨자를 자동으로 만들어 첨자의 값과의 쌍인 튜플을 만들어 준다. 0 부터 시작하는 첨자와 항목 값의 튜플 리스트를 생성한다.

```
>>> lst = [10, 20, 30]
>>> list(enumerate(lst))
[(0, 10), (1, 20), (2, 30)]
```

키워드 인자인 start 를 사용하면 시작 첨자를 지정할 수 있다. 문자열 리스트도 사용할 수 있다.

```
>>> list(enumerate([10, 20, 30], start = 1))
[(1, 10), (2, 20), (3, 30)]
>>> lst = 'python'
>>> list(enumerate(lst))
[(0, 'p'), (1, 'y'), (2, 't'), (3, 'h'), (4, 'o'), (5, 'n')]
```

■ **반복에서 사용하는 내장 함수 enumerate()**

for 반복의 시퀀스에 사용하는 것이 좋다. 변수 tp 로 지정하면 tp[0]는 첨자, tp[1]은 값이 된다. format() 메소드에서 *tp 로 지정하면 자동으로 나뉘어 앞부분에는 첨자 뒤에는 값이 출력된다.

```
>>> subj = ['국어', '영어']
>>> subj = ['국어', '영어']
>>> for tp in enumerate(subj):
        print('lst[{}]: {}'.format(tp[0], tp[1]))
        print('lst[{}]: {}'.format(*tp))


lst[0]: 국어
lst[0]: 국어
lst[1]: 영어
lst[1]: 영어
```

3) 시퀀스 간의 변환

■ **튜플과 시퀀스 간의 변환**

내장함수 list()와 tuple(), set(), dict() 등을 사용하면 시퀀스를 간단하게 변환할 수 있다.

list(튜플) 함수를 사용해 튜플을 리스트로 변환하기

```
>>> space = '밤', '낮', '해
>>> print(space)
('밤', '낮', '해')
>>> print(list(space))
['밤', '낮', '해']
```

tuple(리스트) 함수를 사용해 리스트를 튜플로 변환하기

```
>>> space = ['밤', '낮', '해']
>>> print(space)
['밤', '낮', '해']
>>> print(tuple(space))
('밤', '낮', '해')
```

■ **리스트와 집합 간의 변환**

set() 함수를 이용해 리스트를 집합으로 변환할 수 있다. 순서는 무의미하다.

```
>>> fruit = ['사과', '귤', '바나나', '사과']
>>> print(fruit)
['사과', '귤', '바나나', '사과']
>>> print(set(fruit)) # 집합으로 변환하면서 중복이 제거
{'귤', '바나나', '사과'}
```

■ **딕셔너리를 다른 시퀀스로 변환**

list() 함수로 변환하면 항목이 키로만 구성된 리스트가 반환된다.

```
>>> game = dict(일월 = '소나무', 이월 = '매화', 삼월 = '벚꽃')
>>> lgame = list(game)
>>> print(lgame)
['일월', '이월', '삼월']
```

tuple 로 변환하면 항목이 키로만 구성된 튜플이 반환된다. 집합도 가능하다.

```
>>> print(tuple(game))
('일월', '이월', '삼월')
>>> print(set(game))
{'일월', '이월', '삼월'}
```

★ 딕셔너리로 구기 종목과 팀원 수 만들기

```
#구기 종목 리스트
sports=['축구', '야구', '농구', '배구']
#위 종목에 대응하는 팀원 수를 항목으로 구성
num=[11, 9, 5, 6]
print(sports)
print(num)
print()

print('함수 zip():')
for s, i in zip(sports, num):
    print('%s: %d명' % (s, i), end=' ')
print()
for tp in zip(sports, num):
    print('{}: {}명'.format(*tp), end=' ')
print(); print()

#dict()와 zip() 함수로 종목의 이름을 키, 인원수를 값으로 저장
print('함수 dict(zip()):')
sportsnum=dict(zip(sports, num))
print(sportsnum)
```

```
['축구', '야구', '농구', '배구']
[11, 9, 5, 6]

함수 zip():
축구: 11명 야구: 9명 농구: 5명 배구: 6명
축구: 11명 야구: 9명 농구: 5명 배구: 6명

함수 dict(zip()):
{'축구': 11, '야구': 9, '농구': 5, '배구': 6}
```

6 장 요점정리!

1. 딕셔너리는 키와 값이 쌍으로 이루어진다. 튜플은 키로 사용할 수 있지만, 리스트는 사용할 수 없다.

2. get() 메소드는 딕셔너리에 키가 없어도 오류가 발생하지 않는다. 하지만 pop() 메소드는 KeyError 가 발생한다.

3. 집합은 중복이 없고 순서도 의미없다. 집합의 원소는 중복을 허용하지 않기 때문에 멤버십 검사와 중복제거에 사용된다. 리스트와 딕셔너리는 집합의 원소로 사용이 불가능하다. 집합 연산도 산술 연산처럼 축약 대입 연산이 가능하다.

5. zip()으로 딕셔너리를 만들 때는 인자가 2 개여야 한다. 아니면 오류가 발생한다.

6 장 문제 풀어 보기

```
# 1번

my = {'이름':'김영희', '전화번호':'010-3017-4468', '성별':'여자', '나이':22, '대학교':'한국대학교'}
print(my)
```

```
{'이름': '김영희', '전화번호': '010-3017-4468', '성별': '여자', '나이': 22, '대학교': '한국대학교'}
```

```
# 2번

a={'삼성에스디에스':242000, '삼성전자':47000, '엔씨소프트':52600, '현디소프트':5120, '골프존':215000, '기아':56300}

while True:
    aa=input('주식 이름 ? ')

    if aa not in a:
        print('주식 이름이 없습니다.')
        break
    else:
        print(aa,a.get(aa))
```

```
주식 이름 ? 엔씨소프트
엔씨소프트 52600
주식 이름 ? 골프존
골프존 215000
주식 이름 ? 현대
주식 이름이 없습니다.
```

```
# 5번

fruits = ['apple', 'banana', 'grapes', 'prar']
prices = (1000, 500, 1200, 1500)

aa=dict(zip(fruits, prices))

print(aa)


all = list(zip(fruits, prices))


all2 = dict(enumerate(all, start=1))
print(all2)
```

```
{'apple': 1000, 'banana': 500, 'grapes': 1200, 'prar': 1500}
{1: ('apple', 1000), 2: ('banana', 500), 3: ('grapes', 1200), 4: ('prar', 1500)}
```

후기

파이썬 언어를 배우면서 처음 든 생각은 간결한데 어렵다였다. 같은 프로그래밍 언어라 1 학년 때 배운 자바나 C 언어와 비슷하겠다고 생각했는데 내 예상을 빗나갔다. 파이썬이 다른 언어들보다 쉽고 간결하다는 건 이해됐지만, 문법이나 용어들이 다른 언어들과는 너무 달라서 많이 헷갈렸다. 하지만 강의를 듣고 혼자서 포트폴리오를 만들어보면서 두 번, 세 번 보고 듣다 보니 이해가 가기 시작했다. 메소드나 함수들이 너무 많아서 아직은 코딩 연습을 하고자 할 때 다시 한번 더 찾아보고 하는 과정을 가지고 있지만 조금만 더 익숙해진다면 다른 언어들보다 훨씬 간편하게 쓸 수 있을 것 같다는 생각이 들었다. 온라인 강의가 이어지면서 집에서 듣기 때문에 조금 이해가 안 가도 넘어가는 부분이 있었는데 이렇게 포트폴리오를 작성하면서 정리를 하는 것이 많은 도움이 됐다.