2020학년도 파이썬프로그래밍
교과목 포트폴리오

학부	컴퓨터공학부
학과	컴퓨터정보공학과
학번	20190740
성명	전지은

목 차

Chapter 01 파이썬 언어의 개요와 첫 프로그래밍

01 파이썬 언어와 컴퓨팅 사고력 1
02 파이썬 설치와 파이썬 쉘 실행 2
03 제4차 산업혁명 시대, 모두에게 필요한 파이썬 5

Chapter 02 파이썬 프로그래밍을 위한 기초 다지기

01 다양한 자료: 문자열과 수 6
02 변수와 키워드, 대입 연산자 7
03 자료의 표준 입력과 자료변환 함수 8
프로젝트 Lab 1 표준 입력한 실수와 연산식의 산술 연산 및 결과 출력 9
프로젝트 Lab 2 여러 진수를 표준 입력한 수를 여러 진수로 출력 9

Chapter 03 일상에서 활용되는 문자열과 논리 연산

01 문자열 다루기 10
02 문자열 관련 메소드 12
03 논리 자료와 다양한 연산 14
프로젝트 Lab 1 할인율에 따른 할인 가격 16
프로젝트 Lab 2 단어를 추출해 회문인지 검사 16

Chapter 04 일상생활과 비유되는 조건와 반복

01 조건에 따른 선택 if … else 17
02 반복을 제어하는 for문과 while문 17
03 임의의 수인 난수와 반복을 제어하는 break문, continue문 18
프로젝트 Lab 1 1에서 10 사이의 수 맞히기 19
프로젝트 Lab 2 커피 주문받아 주문 가격 표시 20
프로젝트 Lab 3 1개월 달력 출력 21

Chapter 05 항목의 나열인 리스트와 튜플

01 여러 자료 값을 편리하게 처리하는 리스트 22
02 리스트의 부분 참조와 항목의 삽입과 삭제 23
03 항목의 순서나 내용을 수정할 수 없는 튜플 26
프로젝트 Lab 1 스포츠 종목과 팀원 수를 리스트로 적절히 출력 28
프로젝트 Lab 2 햄버거 콤보 번호를 받아 주문 가격 표시 28

Chapter 06 일상에서 활용되는 문자열과 논리 연산

01 키와 값인 쌍의 나열인 딕셔너리 30
02 중복과 순서가 없는 집합 33
03 내장 함수 zip()과 enumerate(), 시퀀스 간의 변환 37
프로젝트 Lab 1 철수와 영희의 가위바위보 게임 39
프로젝트 Lab 2 딕셔너리로 만드는 K-pop 차트 40

Chapter 01 파이썬 언어의 개요와 첫 프로그래밍

01 파이썬 언어와 컴퓨팅 사고력

1. 파이썬 언어란?

◆ 파이썬은 배우기 쉬운 프로그래밍 언어

파이썬은 배우기 쉽고 누구나 무료로 사용할 수 있는 오픈 소스 프로그래밍 언어다. 파이썬은 1991년 네덜란드의 귀도 반 로섬이 개발했으며, 현재는 비영리 단체인 파이썬 소프트웨어 재단이 관리하고 있다.

2. 컴퓨팅 사고력과 파이썬

◆ 프로그래밍의 절차

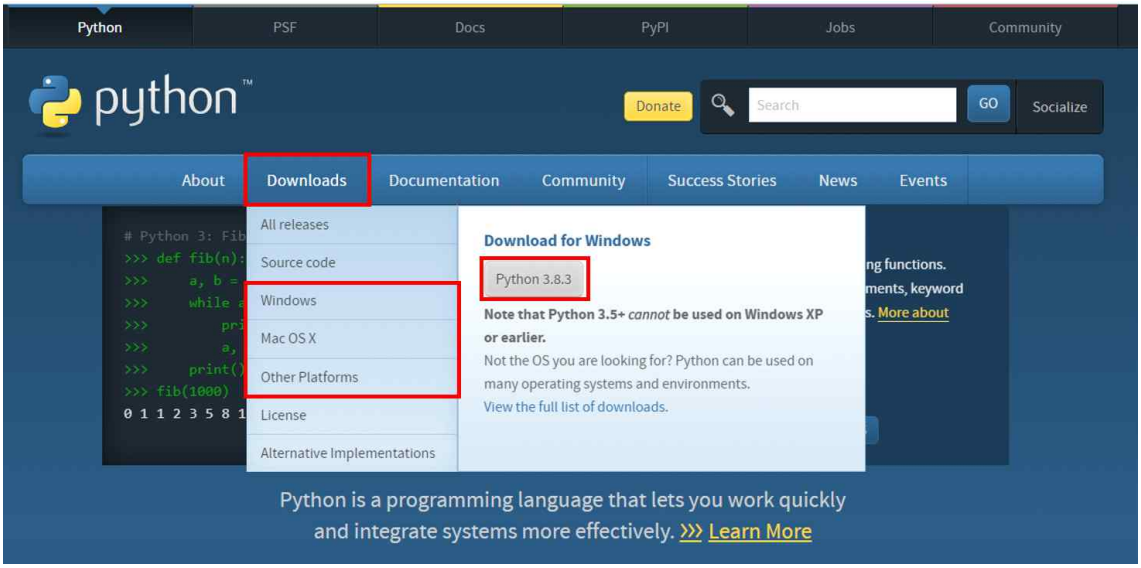
코딩 절차	컴퓨팅 사고력 요소 활용	내용
이해	컴퓨팅 사고력 : 핵심 개념	- 주어진 문제를 이해하고 파악 - 분할: 문제를 좀 더 쉬운 작은 문제들로 분해
설계		- 패턴인식: 분해된 문제들 사이의 유사성 또는 패턴을 탐색 - 추상화: 문제에서 불필요한 부분은 제거하고 주요 핵심 요소를 추려 내는 과정 - 알고리즘: 프로그래밍 언어인 파이썬에 맞는 입력과 출력, 변수 저장 그리고 절차에 따라 구성
구현		- 문제 해결을 위해 파이썬으로 코드 개발, 작성된 코드의 실행과 테스팅 - 디버깅 과정을 거쳐서 코드를 수정하고 필요하면 다시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

02 파이썬 설치와 파이썬 쉘 실행

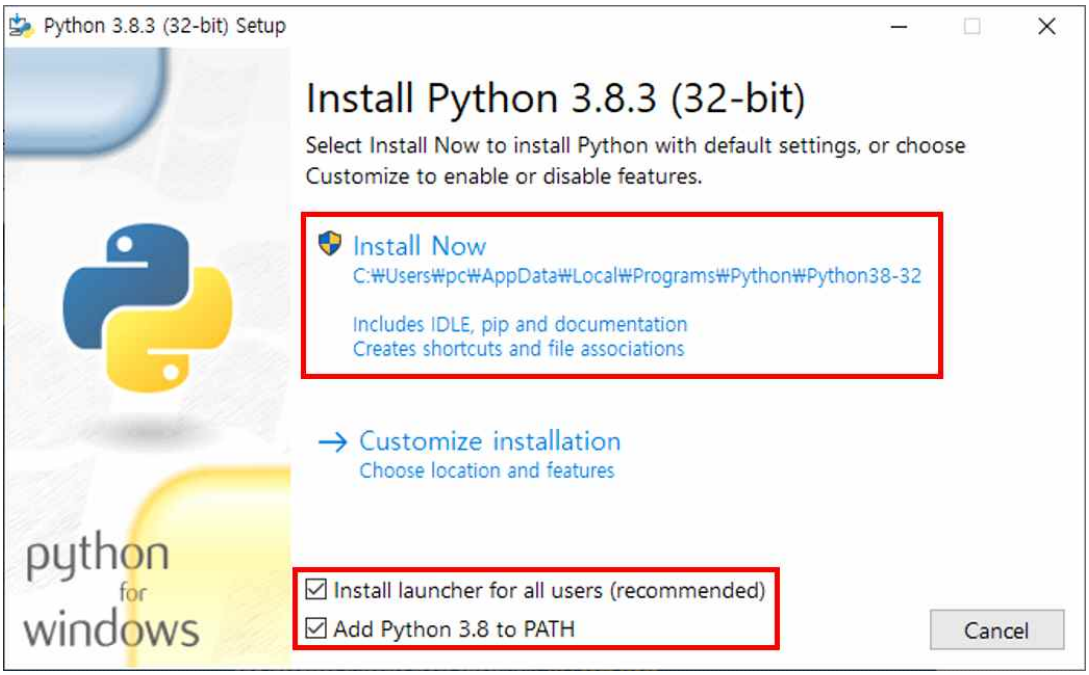
1. 파이썬 개발 도구 설치와 파이썬 쉘의 실행

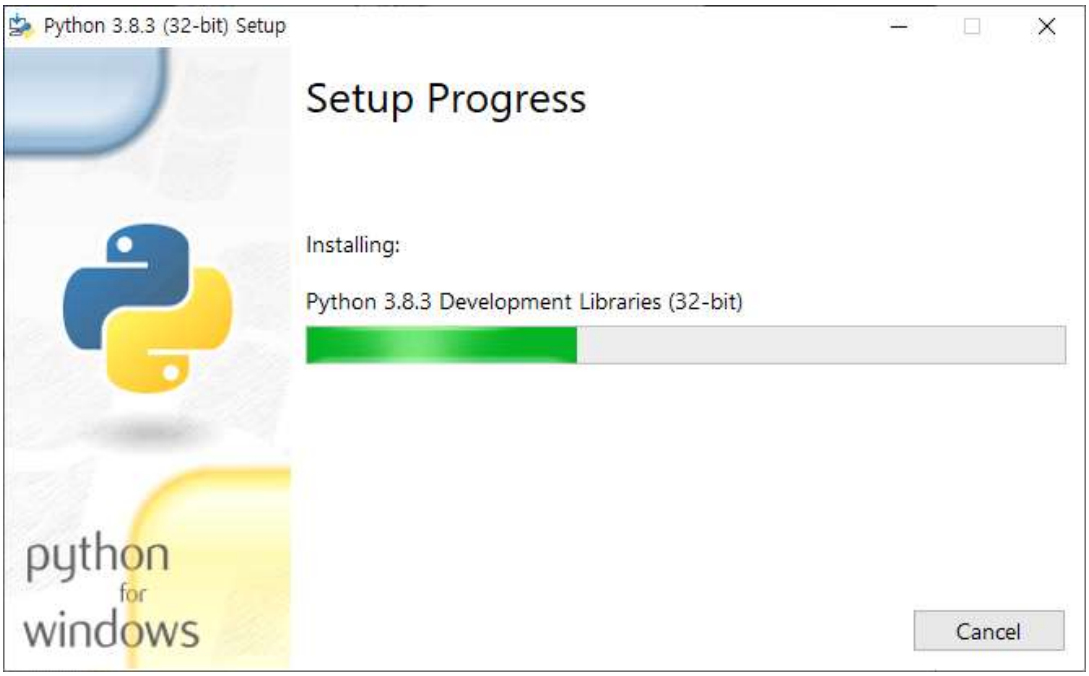
◆ 자신의 운영체제에 적합한 파이썬 설치

윈도 사용자라면 파이썬 홈페이지에서 download를 누르면 나타나는 메뉴에서 Python 3.8.3을 클릭해 설치한다. 다운로드가 완료되면 자동 설치 파일인 python-3.8.3.exe를 실행하자.

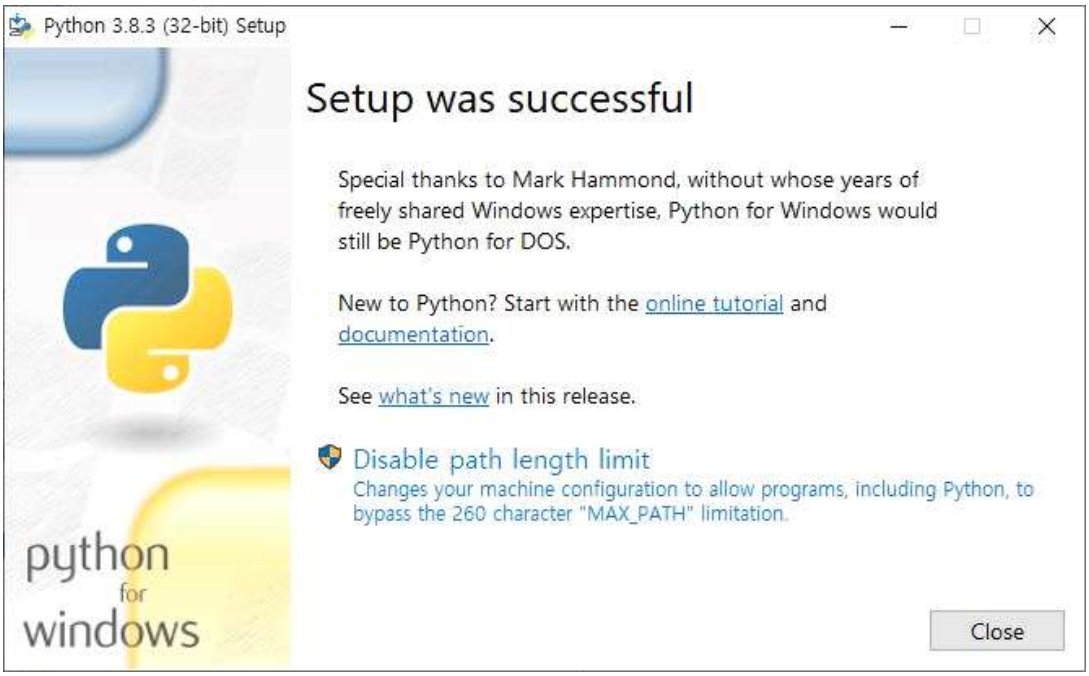


파이썬 설치 첫 화면에서 Install Now를 선택하면 간단하게 설치할 수 있다. 쉬운 사용과 자동 path를 설정하기 위해 하단에 위치한 2개의 체크박스를 모두 선택한 후, 지정된 폴더에 설치하는 Install Now를 선택한다.





정상적으로 설치되면 [Setup was successful] 대화상자가 표시된다. Close 버튼을 눌러 설치를 종료한다.



2. 파이썬 쉘에서 첫 대화형 프로그래밍

◆ 대화형 코딩

파이썬 쉘에 출력하려면 print 다음에 '('Hello World!')'를 입력한다.

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

◆ 파이썬 쉘의 명령어는 첫 칸부터 입력

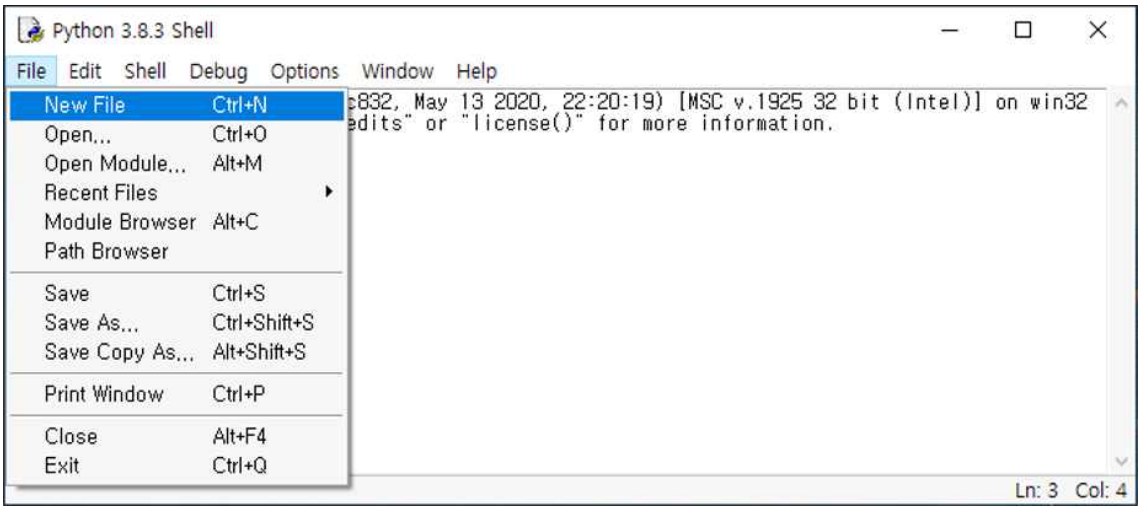
모든 명령어는 첫 칸부터 입력해야 한다. 첫 칸뿐 아니라 그 이상을 공간으로 두고 입력하면 무조건 'SyntaxError: unexpected indent'가 표시된다.

```
>>>  print('Hello World!')
SyntaxError: unexpected indent
```

3. 편집기에서 첫 파일 프로그래밍

◆ 코딩한 파일을 저장할 편집기 생성

파이썬 쉘의 메뉴 [File] - [New File]을 선택해 코딩할 편집기를 나타나게 한다.



◆ 파이썬 편집기 소스 파일 실행

소스 코드가 완성되면 실행 전에 단축키 [Ctrl + S]로 저장한 후 [F5]로 실행한다. 파이썬 편집기에서 작성된 소스 파일의 실행 결과는 다시 파이썬 쉘에 표시된다.

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

```
Hello World!
Hi, python
```

03 제4차 산업혁명 시대, 모두에게 필요한 파이썬

1. 쉽고 강력한 언어

- 간결하고, 인간의 사고 체계와 닮아 사용하기가 쉽다.
- 무료이며, 다양한 자료 구조의 제공으로 생산성이 높다.
- 라이브러리가 독보적이고, 강력하며, 데이터과학과 머신 러닝 등과 같은 다양한 분야에 적용할 수 있다.
- 다른 모듈을 연결해 사용할 수 있는 풀 언어로도 자주 활용된다.

2. 빅테이터 처리와 머신 러닝 등 다양한 분야에 적합한 언어

◆ 교육과 학술, 실무 등 다양한 분야에 사용

대학 프로그래밍 교양 수업부터 스타트업, 대형 글로벌 기업에 이르기까지 다양한 분야에 활용되고 있다.

◆ 인공 지능의 구현과 빅데이터 분석 처리 분야에 사용

파이썬은 인공 지능의 머신 러닝과 딥러닝, 빅데이터 처리를 위한 통계 및 분석 방법의 라이브러리도 풍부하게 제공한다.

◆ 파이썬은 좀 속도가 느리다는 단점

대부분의 인터프리터 언어가 그렇듯이 실행속도가 느리다는 것과 모바일 앱 개발 환경에는 아직 사용하기 힘들다는 것을 들 수 있다.

3. 다양한 종류의 파이썬

아이파이썬, 자이썬, 아이언파이썬, 파이파이

4. 인터프리트 방식의 언어, 파이썬

인터프리터는 한 줄 한 줄의 해석을 담당하고 컴파일러는 컴파일을 담당하는 개발 도구 소프트웨어다.

Chapter 02 파이썬 프로그래밍을 위한 기초 다지기

01 다양한 자료: 문자열과 수

1. 자료의 종류와 문자열 표현

◆ 글자의 모임인 문자열

파이썬에서는 문자 하나 또는 문자가 모인 단어나 문장 또는 단락 등을 문자열이라 한다. 즉, 문자열은 ' 일련의 문자 모임' 이라 할 수 있다. 파이썬에서 작은따옴표나 큰따옴표로 앞뒤를 둘러싸 '문자열' 또는 "문자열"처럼 표현한다.

- 파이썬은 문자 하나도 문자열로 취급하며, 따옴표로 둘러싼 숫자(예: '34', '3.14')도 문자열로 취급한다.
- 즉, 따옴표로 둘러싸면 모두 문자열이다.
- 함수 print()는 문자열이나 숫자 등의 자료를 콘솔에서 출력하는 일을 수행한다.
- 함수 print()는 출력 이후에 다음 줄로 이동해 출력을 준비한다.

2. 문자열 연산자 +, *와 주석

◆ 문자열의 연결 연산자 +와 반복 연산자 *

여러 문자열을 쉽게 연결하는 데에는 "문자열" '연결'과 같이 여러 문자열을 계속 이어 쓰는 방법을 사용한다. 문자열과 문자열 사이 아무것도 없거나 공백이 있어도 상관없으며, 더하기 기호인 +도 사용할 수 있다. 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 발생한다.

◆ 여러 줄의 문자열의 처리에 사용되는 삼중 따옴표

문자열이 길거나 필요에 의해 여러 줄에 걸쳐 문자열을 처리하기 위해서는 삼중 따옴표를 사용할 수도 있다. 삼중 따옴표란, 작은따옴표나 큰따옴표를 연속적으로 3개씩 문자열 앞뒤에 둘러싸는 것을 말한다.

◆ 문법과 상관 없는 주석

코드 내부에 문법과 상관 없는 파일 이름이나 소스 설명 등을 담을 수 있다. 주석은 소스 설명으로, 인터프리터는 주석을 무시한다. 파이썬 주석은 #으로 시작하고 그 줄의 끝까지 유효하다. 주석 #은 한 줄만 가능하므로 여러 줄에 걸친 주석이 필요하다면 줄마다 맨 앞에 #을 넣거나 문자열의 삼중 따옴표를 사용한다.

- 작은따옴표로 둘러싼 문자열의 내부에는 작은따옴표를 문자로 사용할 수 없고, 큰따옴표로 둘러싼 문자열의 내부에는 큰따옴표를 문자로 사용할 수 없다.

3. 정수와 실수의 이해

◆ 수학에서 사용하던 정수와 실수

숫자는 우리가 일상에서 문자열과 함께 많이 사용하는 자료 중 하나다.

- 숫자는 간단히 정수와 실수로 나눈다.

대화형 모드에서는 숫자만 입력해도 결과를 볼 수 있다. 정수 0은 0이 여러 개라도 가능하며, 다른 정수는 0이 앞에 나오면 구문 오류가 발생한다. 실수는 상관없다.

4. 정수와 실수의 다양한 연산

◆ 수의 더하기, 빼기, 곱하기, 나누기 연산자 +-*/

파이썬 쉘은 간단한 계산기로 사용할 수 있다. 수의 연산인 더하기와 빼기, 곱하기와 나누기가 가능하다. 곱하기는 *, 나누기는 /를 사용한다. 이러한 연산자를 모두 산술 연산자라고 부른다. 나누기의 결과는 항상 실수이다. 수학에서 0으로 나누는 계산은 정의가 없듯 파이썬에서 0으로 나누면 ZeroDivisionError가 발생한다.

◆ 수의 몫, 나머지, 지수승 연산자 //, %, **

정수 나눗셈 연산자 //는 나눈 몫이 결과다. 그러므로 나누기 양수에서는 나누기 / 연산에서 소수부 없이 정수만 남긴다. 정수, 실수 모두 가능하다. 나머지 연산자 %는 나머지가 결과다. 연속 별표 2개인 연산자 **는 지수승 연산자다.

```
>>> 8 / 5
1.6
>>> 8 // 5
1
>>> 2.8 / 1.2
2.3333333333333335
>>> 2.8 // 1.2
2.0
>>> -1.5 /  0.2
-7.5
>>> -1.5 // 0.2
-8.0
```

◆ 산술 연산자의 계산 우선순위

연산자	명칭	의미	우선순위
+	더하기	두 피연산자를 더하거나 수의 부호	4, 2
-	빼기	두 피연산자를 빼거나 수의 부호	4, 2
*	곱하기	두 피연산자 곱하기	3
/	나누기	왼쪽을 오른쪽 피연산자로 나누기	3
%	나머지	왼쪽을 오른쪽 피연산자로 나눈 나머지	3
//	몫 나누기	왼쪽을 오른쪽 피연산자로 나눈 결과에서 작거나 같은 양수	3
**	거듭제곱, 지수승	왼쪽을 오른쪽 피연산자로 거듭제곱	1

◆ 표현식 문자열 실행 함수 eval()

함수 eval('expression')은 실행 가능한 연산식 문자열인 expression을 실행한 결과를 반환한다.

02 변수와 키워드, 대입 연산자

1. 자료형

◆ 자료형과 type() 함수

지금까지 알아본 정수와 실수, 문자열 등을 자료형 이라한다.

- 파이썬에서 자료형을 살펴보면 정수는 int, 실수는 float 그리고 문자열은 str로 사용한다.

대화형 모드에서 자료형을 직접 알아보려면 type() 함수를 사용해야한다. 결과는 <class 'int'>와 같이 클래스가 앞에 표시된다.

2. 변수와 대입연산자

◆ 변수의 이해와 대입 연산자 =을 이용한 값의 저장

변수란, 말 그대로 '변하는 자료를 저장하는 메모리 공간'이다. 값을 변수에 저장하기 위해서는 대입 연산자(=)가 필요하다. 오른쪽 값을 왼쪽 변수에 저장하라는 의미다. 대입 연산자의 왼쪽에는 반드시 저장 공간의 변수가 와야 한다. 대입 연산자를 이용하면 a = b = c = 5처럼 여러 변수에 같은 값을 대입 할 수 있다. 변수는 공간은 하나이므로 저장된 값만 기억한다.

◆ 변수 이름을 붙일 때의 규칙

- 문자는 대소문자의 영문자, 숫자 그리고 _로 구성되며, 대소문자는 구별된다.
- 숫자는 맨 앞에 올 수 없다.
- 키워드는 사용할 수 없다.

◆ 다양한 대입 연산자

다양한 대입 연산자	형태	의미	의미
=	a = b	a = b	b의 결괏값을 변수 a에 저장
+=	a += b	a = a + b	a+b의 값을 변수 a에 저장
-=	a -= b	a = a - b	a-b의 값을 변수 a에 저장
*=	a *= b	a = a * b	a*b의 값을 변수 a에 저장
/=	a /= b	a = a / b	a/b의 값을 변수 a에 저장
%=	a %= b	a = a % b	a%b의 값을 변수 a에 저장
//=	a //= b	a = a // b	a//b의 값을 변수 a에 저장
**=	a **= b	a = a ** b	a**b의 값을 변수 a에 저장

03 자료의 표준 입력과 자료 변환 함수

1. 표준 입력과 다양한 변환 함수

◆ 표준 입력 이란?

프로그램 과정에서 쉘이나 콘솔에서 사용자의 입력을 받아 처리하는 방식을 표준 입력 이라 한다.

◆ 함수 input()으로 문자열 표준 입력

함수 input()은 입력되는 표준 입력을 문자열로 읽어 반환하는 함수이고, input()에서 반환되는 입력 문자열을 변수에 저장하려면 대입 연산자 =을 사용해 변수에 저장해야 한다.

◆ 문자열과 정수, 실수 간의 자료변환 함수 str(), int(), float()

함수 str()은 주로 정수와 실수를 문자열로 변환하는 데 사용한다. 함수 int()는 정수 형태의 문자열을 정수형으로 변환 하는데 사용한다. 함수 float()은 소수점이 있는 실수 형태의 문자열을 실수로 변환한다.

2. 16진수, 10진수, 8진수, 2진수와 활용

◆ 16진수, 8진수, 2진수 상수 표현

수의 상수를 표현하는 방법으로 10진법뿐 아니라 16진법, 8진법, 2진법 등이 있다. 16진수는 맨 앞에 0x로 시작하며, 8진수와 2진수는 각 0o와 0b로 시작한다.

◆ 10진수의 변환 함수 bin(), oct(), hex()

10진수를 바로 2진수, 8진수, 16진수로 표현되는 문자열로 변환하려면 각각 함수 bin(), oct(), hex()를 사용해야 한다.

◆ int(정수, 진법기수)를 활용한 여러 진수 상수 형태 문자열의 10진수 변환

여러 진수 형태 문자열을 10진수로 바꾸려면 함수 int('strnum', n)을 사용해야 한다. 여기서 n은 변환하려는 문자열 진수의 숫자를 2, 8, 10, 16등으로 넣는다. int('strnum', 0)을 사용하면 strnum의 맨 앞이 0b, 0o, 0x에 따라 각각의 문자열을 자동으로 각각 2진수, 8진수, 16진수로 인지해 10진수로 변환한다.

프로젝트 Lab 1 표준 입력한 실수와 연산식의 산술 연산 및 결과 출력

표준 입력으로 2개의 실수를 입력받아 더하기, 빼기, 곱하기, 나누기의 연산을 출력한다. 이후 다시 표준 입력으로 하나의 연산식을 한 줄에 입력받아 그 결과를 출력하는 프로그램을 작성해 보자.

```
num1 = float(input('첫 번째 수 입력 >> '))
num2 = float(input('두 번째 수 입력 >> '))
print('합:', num1 + num2)
print('차:', num1 - num2)
print('곱하기:', num1 * num2)
print('나누기:', num1 / num2)
expression = input('연산식 입력(예 3.2 + 4 * 1.5) >> ')
print('연산식:', expression, '결과:', eval(expression))
```

```
첫 번째 수 입력 >> 10.0
두 번째 수 입력 >> 2.5
합: 12.5
차: 7.5
곱하기: 25.0
나누기: 4.0
연산식 입력(예 3.2 + 4 * 1.5) >> 30 // 4
연산식: 30 // 4 결과: 7
```

프로젝트 Lab 2 여러 진수를 표준 입력한 수를 여러 진수로 출력

가장 먼저 변환할 수가 2진수, 8진수, 10진수, 16진수 중에서 무엇인지 입력받는다. 그런 다음, 표준 입력한 수의 2진수, 8진수, 10진수, 16진수를 출력하는 프로그램을 작성하자.

```
base = int(input('입력할 정수의 진수(base)는? '))
invar = input(str(base) + '진수 정수 입력 >> ')
data = int(invar, base)

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

```
입력할 정수의 진수(base)는? 16
16진수 정수 입력 >> 1f
2진수: 0b11111
8진수: 0o37
10진수: 31
16진수: 0x1f
```

Chapter 03 일상에서 활용되는 문자열과 논리 연산

01 문자열 다루기

1. 문자열 str 클래스와 부분 문자열 참조 슬라이싱

◆ 문자열은 클래스 str의 객체

파이썬에서 문자열은 '문자의 나열'로, 텍스트 시퀀스라고도 한다. 문자열의 자료형은 str이다. 'python'과 같은 문자열 상수는 클래스 str의 객체다.

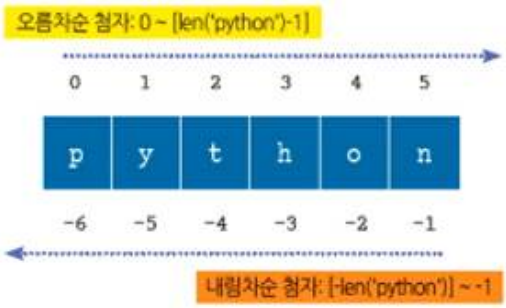
- 작은따옴표: '"큰" 따옴표 표현'처럼 문자열 내부에 큰따옴표 사용 가능
- 큰따옴표: "'작은' 따옴표 표현"처럼 문자열 내부에 작은따옴표 사용 가능
- 삼중 따옴표: '''문자열''' 또는 """문자열"""로 여러 줄의 문자열 표현

◆ 함수 len()으로 문자열의 길이 참조

함수 len(문자열)으로 문자열의 길이를 알 수 있다.

◆ 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.



2. 문자열의 부분 문자열 참조 방식

◆ 문자열 슬라이싱

문자열에서 일부분을 참조하는 방법을 슬라이싱이라고 한다. 물론 기존의 문자열은 절대 수정이 되는 것이 아니라 원 문자열은 변함이 없고 일부분을 반환하는 것이다. 콜론을 사용한 [start:end]로 부분 문자열을 반환하는데, start 첨자에서 end-1 첨자까지의 문자열을 반환한다.

```
>>> 'pyhon'[1:5]
'yhon'
>>> 'pyhon'[2:4]
'ho'
>>> 'pyhon'[0:3]
'pyh'
>>> 'pyhon'[0:6]
'pyhon'
>>> 'pyhon'[0:len('python')]
'pyhon'
>>> 'pyhon'[-5:-1]
'pyho'
>>> 'pyhon'[-4:-1]
'yho'
>>> 'pyhon'[-6:-3]
'py'
>>> 'pyhon'[-6:-1]
'pyho'
>>> 'pyhon'[-len('python'):-1]
'pyho'
>>> 'pyhon'[1:-1]
'yho'
>>> 'pyhon'[-6:4]
'pyho'
```

◆ start와 end를 비우면 '처음부터'와 '끝까지'를 의미

슬라이싱에서 start와 end를 비울 수 있으며, 각각 '처음부터'와 '끝까지'를 의미한다. 그러므로 앞뒤를 모두 비우면 문자열 전체를 반환한다.

```
>>> 'python'[:3]
'pyt'
>>> 'python'[:4]
'pyth'
>>> 'python'[1:]
'ython'
>>> 'python'[3:]
'hon'
>>> 'python'[:]
'python'
```

◆ str[start:end:step]으로 문자 사이 간격을 step으로 조정 가능

부분 반환 문자열에서 문자 사이 간격을 step으로 조정하려면 str[start:end:step]을 사용해야 한다. step을 생략하면 1이다.

- str[:] == str[::] == str[::1]은 모두 전체 문자열을 반환한다.

간격인 step은 음수도 가능하며, 이 경우 start는 end보다 작아야 한다. str[::-1]은 문자열 str을 역순으로 구성된 문자열로 반환한다.

```
>>> 'python'[::1]
'python'
>>> 'python'[1:5:2]
'yh'
>>> 'python'[1:5:3]
'yo'
>>> 'python'[::-1]
'nohtyp'
>>> 'python'[5:0:-1]
'nohty'
>>> 'python'[-1:-7:-1]
'nohtyp'
```

3. 문자 함수와 이스케이프 시퀀스

◆ 문자 함수 ord()와 chr()

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

◆ 이스케이프 시퀀스 문자

하나의 문자를 역슬래시(₩)로 시작하는 조합으로 표현하는 문자를 이스케이프 시퀀스 문자라고 한다.

이스케이프 시퀀스 문자	₩₩	₩'	₩"	₩a	₩b	₩n	₩N{name}
설명	역슬래시	작은따옴표	큰따옴표	벨소리	백스페이스	새 줄	유니코드 이름

₩r	₩f	₩t	₩v	₩uxxxx	₩Uxxxxxxxx	₩ooo	₩xhh
동일한 줄의 맨 앞으로 이동	폼피드	수평 탭	수직 탭	16비트 16진수 코드	32비트 16진수 코드	8진수의 코드 문자	16진수의 코드 문자

◆ 문자열의 최대와 최소

내장 함수 min()과 max()는 인자의 최댓값과 최솟값을 반환하는 함수다. 인자가 문자열 1개이면 문자열을 구성하는 문자에서 코드 값으로 최대와 최소인 문자를 반환한다. 만일 인자가 문자열이 2개 이상이면 문자열 중 최대와 최소인 문자열을 반환한다. 2개 이상의 숫자도 인자가 될 수 있으며 최대와 최소수를 반환한다.

```
>>> min('ipython')
'h'
>>> max('ipython')
'y'
>>> min('3259')
'2'
>>> max('3259')
'9'
>>> min(3, 96.4, 13)
3
>>> max(3, 96.4, 13)
96.4
>>> min('ipython', 'java')
'ipython'
>>> max('ipython', 'java')
'java'
```

02 문자열 관련 메소드

1. 문자열 대체와 부분 문자열 출현 횟수, 문자열 삽입

◆ 문자열 바꿔 반환하는 메소드 replace()

클래스에 소속된 함수를 메소드라 한다. 문자열 클래스 str에 속한 메소드는 매우 다양한데 메소드의 호출은 '문자열 객체명.함수 이름(인자)'와 같이 사용한다. 즉, 문자열 객체가 str이라면 str.replace(a, b)처럼 호출한다. 메소드 str.replace(a, b)는 문자열 str에서 a가 나타나는 모든 부분을 b로 모두 바꾼 문자열을 반환한다.

```
>>> str = '자바는 인기 있는 언어 중 하나다.'
>>> str.replace('자바는', '파이썬은')
'파이썬은 인기 있는 언어 중 하나다.'
>>> str.replace(' ', '')
'자바는인기있는언어중하나다.'
```

메소드 replace(old, new, count)는 문자열 old를 new로 대체하는데, 옵션인 count는 대체 횟수를 지정한다. 옵션인 count가 없으면 모두 바꾸고, 있으면 앞에서부터 지정한 횟수만큼 바꾼다.

```
>>> str = '파이썬 파이썬 파이썬'
>>> str.replace('파이썬', 'Python!')
'Python! Python! Python!'
>>> str.replace('파이썬', 'Python!', 1)
'Python! 파이썬 파이썬'
>>> str.replace('파이썬', 'Python!', 2)
'Python! Python! 파이썬'
```

- 한 번 만들어진 문자열은 수정될 수 없다는 특징이 있으며, 위 문장으로 문자열 str 자체가 수정되는 것이 아니라 replace() 등의 메소드 기능이 작용한 새로운 문자열을 반환한다.

◆ 부분 문자열 출현 횟수를 반환하는 메소드 count()

문자열 str에서 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.

```
>>> num = '12345'
>>> '->'.join(num)
'1->2->3->4->5'
```

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 발생시키지 않고 –1을 반환한다.

```
>>> str = '자바 C'
>>> str.find('자바')
0
>>> str.index('자바')
0
>>> str.find('C++')
-1
>>> str.index('C++')
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    str.index('C++')
ValueError: substring not found
```

3. 문자열 나누기

◆ 문자열을 여러 문자열로 나누는 split() 메소드

문자열 메소드 str.split()는 문자열 str에서 공백을 기준으로 문자열을 나눠준다. 결과는 나눠진 항목들이 리스트라는 형태인 [항목1, 항목2, …]처럼 표시된다. 만일 str.split(',')처럼 괄호 안에 특정한 문자열 값이 있을 경우에는 이 부분 문자열 값을 구분자를 이용해 문자열을 나눠 준다.

```
>>> '사과 배 복숭아 딸기 포도'.split()
['사과', '배', '복숭아', '딸기', '포도']
>>> '데스크톱 1000000, 노트북 1800000, 스마트폰 1200000'.split(', ')
['데스크톱 1000000', '노트북 1800000', '스마트폰 1200000']
```

4. 다양한 문자열 변환 메소드

◆ 폭을 지정하고 중앙에 문자열 배치하는 메소드 center()

```
>>> ' 파이썬 강좌 '.center(30, '*')
'*********** 파이썬 강좌 ***********'
>>> ' 파이썬 강좌 '.center(30)
'            파이썬 강좌            '
>>> ' 파이썬 강좌 '.center(30, '=')
'=========== 파이썬 강좌 ==========='
```

◆ 문자열 앞뒤의 특정 문자들을 제거하는 strip() 메소드

```
>>> '  python  '.lstrip()
'python  '
>>> '  python  '.rstrip()
'  python'
>>> '  python  '.strip()
'python'
>>> '  ***python---  '.strip('* -')
'python'
```

5. 출력을 정형화하는 함수 format()

◆ 문자열의 format() 메소드를 이용해 간결한 출력 처리

문자열 메소드 str.format(인자들)을 사용해 문자열 str 중간중간에 변수나 상수를 함께 출력할 수 있다. 인자는 상수뿐 아니라 변수도 가능하다. {0}, {1}처럼 인자의 순서를 나타내는 정수를 0부터 삽입하면 인자의 순서와 출력 순서를 조정할 수 있다. 10진수 정수는 d, 부동소수는 f로 표기한다.

```
>>> a, b = 10, 4
>>> print('{1} / {0} = {2}'.format(a, b, b / a))
4 / 10 = 0.4
>>> a, b = 10, 3
>>> print('{0:5d} / {1:5d} = {2:10.3f}'.format(a, b, a / b))
   10 /     3 =      3.333
```

6. C 언어의 포맷팅 스타일인 %d와 %f 등으로 출력

◆ 전통적인 정형화 방식인 %d와 %f를 사용한 출력 처리

C의 printf()에서 사용하는 형식 지정자 %d와 같은 스타일도 지원한다.

```
>>> print('%10.2f' % 2.718281)
      2.72
>>> print('%d ** %x = %o' % (3, 2, 3 ** 2))
3 ** 2 = 11
>>> print('%d%%' % 99)
99%
```

03 논리 자료와 다양한 연산

1. 논리 값과 논리 연산

◆ 논리 유형 bool과 함수 bool()

파이썬은 논리 값으로 참과 거짓을 의미하는 True와 False를 키워드로 제공한다. 파이썬에서는 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. 논리 값 True와 False는 int(논리값) 함수에 의해 각각 1과 0으로 변환된다.

2. 관계 연산

◆ 논리 값 True와 False의 산술 연산

파이썬은 논리 값 True와 False를 각각 1과 0으로 산술 연산에 활용할 수 있다.

```
>>> 40 * True, 30 * False
(40, 0)
```

3. 멤버십 연산 in

◆ 문자열에서의 부분 문자열인지 검사

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

4. 비트 논리 연산

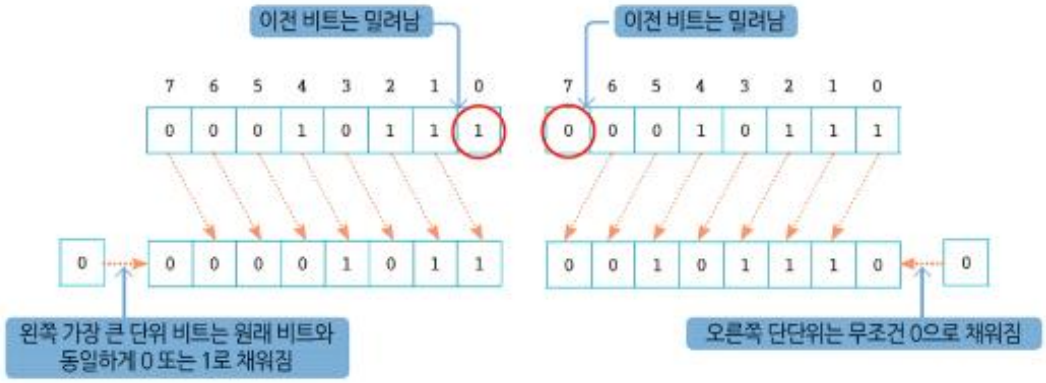
◆ 비트 논리곱 &, 비트 논리합 |, 비트 배타적 논리합 ^

연산식	10진수	2진수표현								설명
		128	64	32	16	8	4	2	1	
a	23	0	0	0	1	0	1	1	1	23의 2진수 00010111
b	57	0	0	1	1	1	0	0	1	57의 2진수 00111001
a & b	17	0	0	0	1	0	0	0	1	비트가 모두 1이면 1
a | b	63	0	0	1	1	1	1	1	1	비트가 하나라도 1이면 1
a ^ b	46	0	0	1	0	1	1	1	0	두 비트가 다르면 1, 같으면 0

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 빈자리는 모두 0으로 채워지며, >>에 의한 빈자리는 원래의 정수 부호에 따라 0이나 양수이면 0, 음수이면 1이 채워진다.



◆ 연산자 우선순위

순위	연산자	설명	부류
1	**	지수승	지수 연산
2	~	비트 보수	단항 연산
3	+x, -x	부호	
4	* / // %	곱, 나누기, 몫, 나머지	산술 연산
5	+ -	더하기, 빼기	
6	>> <<	비트 이동	비트 연산
7	&	비트 and	
8	^	비트 xor	
9	|	비트 or	
10	< <= > >= == != in not in is is not	관계, 소속, id 비교	관계 소속
11	= %= /= //= -= += *= **=	대입, 여러 대입	대입 연산
12	not	논리 not	논리 연산
13	and	논리 and	
14	or	논리 or	

프로젝트 Lab 1 할인율에 따른 할인 가격

다음 표와 같은 할인 조건인 경우에 총 가격을 입력받아, 원 가격, 할인된 가격, 할인율, 할인액을 출력하는 프로그램을 작성하자.

조건	총액 할인율
10,000원 이상, 20,000원 미만	1%
20,000원 이상, 40,000원 미만	2%
40,000원 이상	4%

```
price = int(input('총 가격(원 가격) 입력 >> '))

rate1 = (10000 <= price < 20000) * 1/100
rate2 = (20000 <= price < 40000) * 2/100
rate3 = (40000 <= price) * 4/100
rate = rate1 + rate2 + rate3

discount = price * rate
discPrice = price - discount
print("원 가격:", price, "할인된 가격:", discPrice)
print("할인율:", rate, "할인액:", discount)
```

```
총 가격(원 가격) 입력 >> 9000
원 가격: 9000 할인된 가격: 9000.0
할인율: 0.0 할인액: 0.0

총 가격(원 가격) 입력 >> 30000
원 가격: 30000 할인된 가격: 29400.0
할인율: 0.02 할인액: 600.0

총 가격(원 가격) 입력 >> 45000
원 가격: 45000 할인된 가격: 43200.0
할인율: 0.04 할인액: 1800.0
```

프로젝트 Lab 2 단어를 추출해 회문인지 검사

표준 입력으로 3개의 단어를 콤마로 구분해 입력받자. 3개의 단어를 추출해 각 단어와 역순의 단어를 출력한다. 그리고 그 단어가 회문인지 판단하는 논리 값도 출력하는 프로그램을 작성하자.

- 회문은 거꾸로 읽어도 제대로 읽는 것과 같은 문장이나 낱말을 말한다. 예를 들면 level, 기러기 등이다.

```
str = input('콤마로 구분된 단어 3개 입력 >> ')
str = str.replace(' ', '')
w1, w2, w3 = str.split(',')

print('단어: {}, 역순: {}, 회문: {}'.format(w1, w1[::-1], (w1 == w1[::-1])))
print('단어: {}, 역순: {}, 회문: {}'.format(w2, w2[::-1], (w2 == w2[::-1])))
print('단어: {}, 역순: {}, 회문: {}'.format(w3, w3[::-1], (w3 == w3[::-1])))
```

```
콤마로 구분된 단어 3개 입력 >> 기러기, 파이썬, level
단어: 기러기, 역순: 기러기, 회문: True
단어: 파이썬, 역순: 썬이파, 회문: False
단어: level, 역순: level, 회문: True
```

Chapter 04 일상생활과 비유되는 조건과 반복

01 조건에 따른 선택 if … else

1. 조건의 논리 값에 따른 선택 if

◆ 조건에 따른 선택을 결정하는 if문

조건에 따라 해야 할 일을 처리해야 하는 경우, if문을 사용한다. if문은 조건인 논리 표현식의 결과가 True이면 이후에 구성 된 블록 구문을 실행한다. 그러나 결과가 False이면 실행하지 않는다.

- if문에서 논리 표현식 이후에는 반드시 콜론이 있어야 한다.
- 콜론 이후 다음 줄부터 시작되는 블록은 반드시 들여쓰기 해야 한다.

2. 조건에 따라 하나를 선택하는 if … else

◆ 논리 표현식 결과인 True와 False에 따라 나뉘는 if … else문

조건 if의 논리 표현식 결과는 True 또는 False이며, if … else문에서 else: 블록은 논리 표현식 결과가 False일 때 실행된다.

3. 여러 조건 중에서 하나를 선택하는 구문 if … elif

◆ 다중 택일 결정 구조인 if … elif

조건의 여러 경로 중에서 하나를 선택하는 if … elif문은 논리 표현식 1이 True이면 문장의 블록을 실행하고 종료된다. 논리 표현식 1이 False여야 elif 논리 표현식 2로 진행되며, 그 이후도 마찬가지다. elif는 필요한 만큼 늘릴 수 있으며 마지막 else는 선택적으로 생략 할 수 있다.

- 블록은 단 하나만 선택돼 실행

4. 중첩된 조건

if문의 내부에 다시 if문을 사용할 수 있다.

02 반복을 제어하는 for문과 while문

1. 시퀀스의 내부 값으로 반복을 실행하는 for문

◆ 일상생활과 같은 반복의 실행

반복문을 사용하면 중복을 줄여 프로그램이 간결해지며, 프로그램 절차를 효과적으로 수행할 수 있다. 파이썬에서 제공하는 반복 방법은 크게 두 가지다. while문은 <반복조건>에 따라 반복을 결정하며, for문은 항목의 나열인 <시퀀스>의 구성 요소인 모든 항목이 순서대로 변수에 저장돼 반복을 수행한다.

- 콜론과 반복 몸체인 블록 구성이 반드시 필요하다.

◆ 내장함수 range()를 사용한 for문

반복문 for문의 시퀀스에 내장 함수 range()를 활용하면 매우 간결하다. 내장 함수 range(5)는 정수 0에서 4까지 5개의 항목인 정수로 구성되는 시퀀스를 만든다. range(start, stop, step)에서 start는 시퀀스의 시작, stop은 시퀀스의 끝, step은 증가 값이고, 시퀀스의 끝은 실질적으로 stop-1이다.

- 쉘에서 range() 함수를 list() 내부에 적으면 곧바로 항목을 볼 수 있다.

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 문장1, 문장2를 실행한 후 다시 논리 표현식을 검사해 실행한다.
- 논리 표현식이 False이면 반복 몸체를 실행하지 않고, 선택 사항인 else: 이후의 블록을 실행한 후 반복을 종료한다.

3. 중첩된 반복

변수 여러 개로 반복문을 중첩해 사용할 수 있다.

03 임의의 수인 난수와 반복을 제어하는 break문, continue문

1. 임의의 수인 난수 발생과 반복에 활용

◆ 임의의 수를 발생하는 난수

파이썬에서는 모듈 random의 함수 randint(시작, 끝)를 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.

2. 반복을 제어하는 break문과 continue문

◆ 반복을 종료하는 break문

반복 while의 논리 표현식이 True라면 무한히 반복된다. 이를 무한 반복이라고 한다. for나 while 반복 내부에서 문장 break는 else: 블록을 실행하지 않고 반복을 무조건 종료시킨다.

◆ 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.

프로젝트 Lab 1 1에서 10 사이의 수 맞히기

컴퓨터가 정한 1에서 10 사이의 정수인 정답을 사용자가 맞히는 프로그램을 작성해 보자. 사용자가 정답을 맞출 때까지 계속 힌트를 준다. 입력한 수보다 '작은 수' 또는 '큰 수'라는 정보를 제공해 사용자가 정답을 맞힐 확률을 높인다.

```
import random
answer = random.randint(1, 10)

indata = int(input('1에서 10 사이의 수를 맞히세요 >> '))
while True:
    if indata == answer:
        print('축하한다. {}: 정답이다.'.format(indata))
        break;
    elif indata < answer:
        str = '{}보다 더 큰 수로 다시 입력 >>'.format(indata)
    else:
        str = '{}보다 더 작은 수로 다시 입력 >>'.format(indata)
    indata = int(input(str))

print(" 종료 ".center(30, '*'))
```

```
1에서 10 사이의 수를 맞히세요 >> 7
7보다 더 작은 수로 다시 입력 >>3
3보다 더 큰 수로 다시 입력 >>5
축하한다. 5: 정답이다.
************ 종료 ************
```

프로젝트 Lab 2 여러 진수를 표준 입력한 수를 여러 진수로 출력

요즘은 식당이나 커피 전문점에서 셀프 주문기를 자주 본다. 주문을 위해 커피 종류와 수량을 입력받아 총 주문 가격을 출력하는 프로그램을 작성해 보자. 커피의 종류는 네 가지이며, 0을 입력하면 주문을 종료한다. 1에서 4까지의 커피를 주문하면 바로 수량을 입력받도록 한다. 한 종류의 커피와 수량을 입력받으면 그때까지의 주문 가격도 표시하며, 주문이 종료되면 총 주문 가격을 출력한다. 반복 while True:로 무한 반복하다가 주문이 종료되면 break로 종료하도록 프로그램을 작성해 보자.

- 주문 종류와 주문 종료를 모두 정수 형태로 입력

```
menu = '''Coffee menu!
    1. 아메리카노     2000
    2. 카페라테       2500
    3. 카푸치노       3000
    4. 캐러멜마키아또 4000
    0. 주문 종료
종류 ? '''

print('환영합니다. 음료를 선택하세요')
total = 0
while True:
    order = int(input(menu))
    if order == 0:
        print()
        print(' 주문 종료 '.center(18, '*'))
        break
    else:
        cnt = int(input('수량 ? '))
        if order == 1:
            total += cnt * 2000
            print('₩n%s %d개 주문' % ('아메리카노', cnt))
        elif order == 2:
            total += cnt * 2500
            print('₩n%s %d개 주문' % ('카페라테', cnt))
        elif order == 3:
            total += cnt * 3000
            print('₩n%s %d개 주문' % ('카푸치노', cnt))
        elif order == 4:
            total += cnt * 4000
            print('₩n%s %d개 주문' % ('캐러멜마키아또', cnt))
        else:
            print('모르겠어요.')
        print('현재 주문 가격: %d원' % total)
        print()

print('총 주문 가격: %d원' % total)
print(' 안녕! '.center(20, '='))
```

```
환영합니다. 음료를 선택하세요
Coffee menu!
    1. 아메리카노     2000
    2. 카페라테       2500
    3. 카푸치노       3000
    4. 캐러멜마키아또 4000
    0. 주문 종료
종류 ? 2
수량 ? 3

카페라테 3개 주문
현재 주문 가격: 7500원

Coffee menu!
    1. 아메리카노     2000
    2. 카페라테       2500
    3. 카푸치노       3000
    4. 캐러멜마키아또 4000
    0. 주문 종료
종류 ? 4
수량 ? 2

캐러멜마키아또 2개 주문
현재 주문 가격: 15500원

Coffee menu!
    1. 아메리카노     2000
    2. 카페라테       2500
    3. 카푸치노       3000
    4. 캐러멜마키아또 4000
    0. 주문 종료
종류 ? 0

***** 주문 종료 ******
총 주문 가격: 15500원
======= 안녕! ========
```

프로젝트 Lab 3 1개월 달력 출력

주어진 조건으로 1개월 달력을 출력하는 프로그램을 작성해 보자. 주변의 달력을 보면서 매월 무엇이 바뀌는지 살펴보자. 매월 1일부터의 요일이 바뀌거나 마지막 날짜가 바뀔 수 있다. 표준 입력으로 한 달의 최대 일수와 첫 날 1일의 시작 요일을 입력받아 그 달 한 달의 달력을 출력하는 프로그램을 작성해 보자.

- 일반적으로 달력 표시는 일요일부터 시작하며 1일부터 말일까지 출력한다.

```
dates = int(input('한 달 최대 일수를 입력 >> '))
day = int(input('첫 날 1일의 시작 요일을 입력(0=일, 1=월, ... 6=토) >> '))
day %= 7

print('₩n', end= ' ')
for i in '일월화수목금토':
    print('%s' %i, end= ' ')
else:
    print('₩n')

cnt = 0
if day != 0:
    print('    ' * day, end = '')
    cnt += day

for i in range(1, dates + 1):
    print('%2d' % i, end= ' ')
    cnt += 1
    if cnt % 7 == 0:
        print()
else:
    print()
```

```
한 달 최대 일수를 입력 >> 31
첫 날 1일의 시작 요일을 입력(0=일, 1=월, ... 6=토) >> 4

 일 월 화 수 목 금 토

            1  2  3
 4  5  6  7  8  9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30 31
```

Chapter 05 항목의 나열인 리스트와 튜플

01 여러 자료 값을 편리하게 처리하는 리스트

1. 리스트의 개념과 생성

◆ 관련된 나열 항목을 관리하는 리스트

리스트는 항목의 나열인 시퀀스다. 즉, 리스트는 콤마로 구분된 항목(또는 원소라고도 부름) 들의 리스트로 표현되며, 각 항목은 모두 같은 자료형일 필요는 없다. 즉, 정수, 실수, 문자열, 리스트 등이 모두 가능하다. 항목 순서는 의미가 있으며, 항목 자료 값은 중복돼도 상관없다.

- 리스트는 대괄호 사이에 항목을 기술한다.

◆ 빈 리스트의 생성과 항목 추가

다음과 같이 빈 대괄호로 빈 리스트를 만들 수 있고, 인자가 없는 내장 함수 list()로도 빈 리스트를 생성할 수 있다. 리스트의 가장 뒤에 항목을 추가하려면 리스트의 메소드 append(삽입할 항목)를 사용한다.

```
>>> pl1 = []
>>> print(pl1)
[]
>>> pl2 = list()
>>> pl2.append('C++')
>>> pl2.append('java')
>>> print(pl2)
['C++', 'java']
```

2. 리스트의 항목 참조

◆ 문자열 시퀀스와 같이 첨자로 리스트의 항목 참조

```
>>> 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
```

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.

```
>>> top = ['BTS', '봄빨간사춘기', 'BTS', '블랙핑크', '태연', 'BTS']
>>> top.count('BTS')
3
>>> top.index('봄빨간사춘기')
1
>>> top.index('BTS')
0
```

◆ 리스트의 첨자로 항목 수정

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

4. 리스트 내부에 다시 리스트를 포함하는 중첩 리스트

◆ 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']
```

02 리스트의 부분 참조와 항목의 삽입과 삭제

1. 리스트의 부분 참조인 슬라이싱

◆ 첨자 3개로 리스트 일부분을 참조하는 슬라이싱

슬라이스는 리스트[start:stop:step]와 같이 사용하며, 첨자 start에서 첨자 stop-1까지 step 간격의 요소로 구성된 부분 리스트를 반환한다.

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

2. 리스트의 부분 수정

◆ 리스트의 슬라이스로 리스트의 일부분을 수정

리스트의 일부분을 다른 리스트로 수정하려면 슬라이스 방식에 대입해야 한다.

```
>>> sports = ['풋살', '족구', '비치사커', '야구', '농구', '배구']
>>> sports[0:3] = ['축구']
>>> print(sports)
['축구', '야구', '농구', '배구']
>>> sports[1:3] = sports[4:6]
>>> print(sports)
['축구', '배구']
```

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.

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

◆ 리스트 메소드 remove(항목)과 pop(첨자), pop()로 항목 삭제

메소드 remove(값)은 리스트에서 지정된 값의 항목을 삭제한다. pop(첨자)는 지정된 첨자의 항목을 삭제하고 반환하며, 인자가 없는 pop()은 마지막 항목을 삭제하고 삭제된 값을 반환한다.

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

◆ 문장 del에 의한 항목이나 변수의 삭제

문장 del은 뒤에 위치한 변수나 항목을 삭제한다. 문장 del은 변수 자체를 메모리에서 제거한다.

```
>>> kpop = ['BTS', '장범준', '블랙핑크', '잔나비']
>>> del kpop[0]
>>> print(kpop)
['장범준', '블랙핑크', '잔나비']
>>> del kpop[0:2]
>>> print(kpop)
['잔나비']
>>> del kpop
>>> print(kpop)
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    print(kpop)
NameError: name 'kpop' is not defined
```

◆ 리스트의 모든 항목을 제거해 빈 리스트로 만드는 메소드 clear()

메소드 clear()는 리스트의 모든 항목이 삭제된다. 이후 리스트는 빈 리스트가 된다.

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

4. 리스트의 추가, 연결과 반복

◆ 리스트에 리스트를 추가하는 메소드 extent()

메소드 리스트.extend(list)는 리스트에 인자인 list를 가장 뒤에 추가한다.

```
>>> kpop = ['BTS', '장범준', '블랙핑크', '잔나비']
>>> kpop.clear()
>>> print(kpop)
[]
>>> day = ['월', '화', '수']
>>> day2 = ['목', '금', '토', '일']
>>> day.extend(day2)
>>> print(day)
['월', '화', '수', '목', '금', '토', '일']
```

5. 리스트의 항목 순서와 정렬

◆ 리스트 항목 순서를 뒤집는 메소드 reverse()

리스트 메소드 reverse()는 항목 순서를 반대로 뒤집는다.

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

◆ 리스트 항목 순서를 정렬하는 메소드 sort()

리스트 메소드 sort()는 리스트 항목의 순서를 오름차순으로 정렬한다. sort(reverse=True)로 호출하면 역순인 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']
```

◆ 리스트 항목의 순서를 정렬한 리스트를 반환하는 내장 함수 sorted()

내장 함수 sorted(리스트)는 리스트의 항목 순서를 오름차순으로 정렬한 새로운 리스트를 반환한다. sorted(리스트, reverse=True)로 호출하면 역순인 내림차순으로 정렬된 리스트를 반환한다.

```
>>> 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']
```

6. 리스트 컴프리헨션

◆ 조건을 만족하는 항목으로 리스트를 간결히 생성하는 컴프리헨션

리스트 컴프리헨션은 리스트를 만드는 간결한 방법을 제공한다. 리스트 컴프리헨션은 리스트를 만들므로 대괄호 […]로 감싸고, 구성 항목인 i를 가장 앞에 배치하며, for문이 오는데 range(2, 11, 2) 뒤에 콜론이 빠진다. 컴프리헨션은 함축, 축약, 내포, 내장 등으로도 불린다.

- 리스트 = [항목연산식 for 항목 in 시퀀스 if 조건식]

```
>>> even = [i for i in range(2, 11, 2)]
>>> print(even)
[2, 4, 6, 8, 10]
>>> odd = [i for i in range(10) if i%2 == 1]
>>> print(odd)
[1, 3, 5, 7, 9]
```

7. 리스트 대입과 복사

◆ 리스트 대입에 의한 동일 리스트의 공유

리스트에서 대입 연산자 = 은 얕은 복사라고 해서 대입되는 변수가 동일한 시퀀스를 가리킨다.

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

◆ 리스트의 깊은 복사에 의한 대입으로 새로운 리스트의 생성

리스트에서 완전히 새로운 리스트를 만들어 복사하려면 슬라이스 [:]나 copy() 또는 list() 함수를 이용해야 한다. 이러한 복사를 깊은 복사라 한다.

```
>>> f1 = ['사과', '귤', '복숭아', '파인애플']
>>> f2 = f1[:]
>>> f2.pop(1)
'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']
```

◆ 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
>>> print(f1 is f2)
True
>>> f3 = f1[:]
>>> print(f1 is f3)
False
```

03 항목의 순서나 내용을 수정할 수 없는 튜플

1. 괄호로 정의하는 시퀀스 튜플

◆ 수정할 수 없는 항목의 나열인 튜플

튜플은 리스트와 달리 항목의 순서나 내용의 수정이 불가능하다. 튜플도 모두 콤마로 구분된 항목들의 리스트로 표현되며, 각각의 항목은 정수, 실수, 문자열, 리스트, 튜플 등 제한이 없다.

- 튜플은 괄호 (…) 사이에 항목을 기술한다. 괄호는 생략할 수 있다.

◆ 튜플의 생성

빈 튜플은 ()로 만든다. 빈 튜플은 함수 tuple()로도 생성할 수 있다. 항목이 하나인 튜플을 표현할 때는 마지막에 콤마를 반드시 붙여야 한다.

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

2. 튜플 연결과 반복, 정렬과 삭제

◆ 튜플 연결 + 연산자와 반복 * 연산자

리스트와 같이 +와 *는 각각 튜플을 연결하고 항목이 횟수만큼 반복된 튜플을 반환한다.

```
>>> 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')
```

◆ 튜플 항목의 순서를 정렬한 리스트를 반환하는 내장 함수 sorted()

내장 함수 sorted(튜플)은 튜플 항목의 순서를 오름차순으로 정렬한 새로운 리스트를 반환한다. 함수 sorted(튜플, reverse=True)로 호출하면 역순인 내림차순으로 정렬된 리스트를 반환한다. 절대 튜플 자체가 수정되지는 않는다.

```
>>> 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')
```

◆ 문장 del에 의한 튜플 변수 자체의 제거

문장 del에서 변수 kpop을 지정하면 변수 자체가 사라진다.

프로젝트 Lab 1 스포츠 종목과 팀원 수를 리스트로 적절히 출력

다음과 같이 스포츠 종목과 팀원 수로 구성된 리스트로 적절히 출력하는 프로그램을 작성하자.

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

또한 위 두 리스트 sports와 num으로 다음 중첩된 리스트를 만들어 종목과 팀원 수를 적절히 출력하는 코드도 포함시키자.

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

마지막으로 두 리스트 sports와 num을 갖고 리스트 컴프리헨션으로 다음 구조의 중첩된 리스트도 만들어 종목과 팀원 수를 적절히 출력하는 코드도 포함시켜 보자.

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

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

print(sports)
print(num)

for i in range(len(sports)):
    print('%s: %d명  ' % (sports[i], num[i]), end = ' ')
print(); print()

sponum = [sports, num]
print(sponum)

for i in range(len(sponum[0])):
    print('%s: %d명  ' % (sponum[0][i], sponum[1][i]), end = ' ')
print(); print()

psponum = [[sports[i], num[i]] for i in range(len(sports))]
print(psponum)

for one in psponum:
    print('%s: %d명  ' % (one[0], one[1]), end = ' ')
print()
```

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

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

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

프로젝트 Lab 2 햄버거 콤보 번호를 받아 주문 가격 표시

이번에는 햄버거 주문을 구현해 보자. 주문을 위해 콤보의 종류와 수량을 입력받아 주문한 내역과 가격 그리고 총 주문 가격을 출력하는 프로그램을 작성해 보자. 주문 종류는 오른쪽과 같으며, 0을 입력하면 주문을 종료한다. 콤보 종류와 수량을 한 번에 입력받으면 주문 내역과 총 가격을 표시하고, 주문이 종료되면 총 주문 가격을 출력한다.

```
menu = ('주문 종료', '올인원팩', '투게더팩', '트리오팩', '패밀리팩')
price = (0, 6000, 7000, 8000, 10000)
```

주문 리스트와 가격을 리스트나 튜플로 사용하면 훨씬 간결하다. 주문에 사용하는 번호와 menu의 항목 첨자 번호를 같게 하면 편리하다.

```
주문할 콤보 번호와 수량을 계속 입력하세요!
        0 주문 종료
        1 올인원팩  6000 원
        2 투게더팩  7000 원
        3 트리오팩  8000 원
        4 패밀리팩 10000 원
 >> 1 1
```

```
menu = ('주문 종료', '올인원팩', '투게더팩', '트리오팩', '패밀리팩')
price = (0, 6000, 7000, 8000, 10000)

msg = '주문할 콤보 번호와 수량을 계속 입력하세요!'
for i in range(len(menu)):
    msg += '₩n₩t %d %s' % (i, menu[i])
    if i != 0:
        msg += ' %5d 원' % (price[i])
msg += '₩n >> '

more = True
total = 0
while more:
    instr = input(msg)
    if instr.count(' ') > 0:
        order, cnt = instr.split()
        cnt = int(cnt)
    else:
        order = instr
    order = int(order)
    if order == 0:
        print()
        print(' 주문 종료 '.center(20, '*'))
        more = False
    elif 1 <= order <= 4:
        print('%s, %d개 주문' % (menu[order], cnt))
        sub = price[order] * cnt
        total += sub
        print('%s 주문 가격 %d, 총 가격 %d' % (menu[order], sub, total))
    else:
        print('모르겠어요. 다시 주문하세요!')
else:
    print('총 주문 가격 %d 원' % total)
    print('주문을 마치겠습니다.')
    print(' 안녕! '.center(20, '='))
```

```
주문할 콤보 번호와 수량을 계속 입력하세요!
        0 주문 종료
        1 올인원팩  6000 원
        2 투게더팩  7000 원
        3 트리오팩  8000 원
        4 패밀리팩 10000 원
 >> 1 1
올인원팩, 1개 주문
올인원팩 주문 가격 6000, 총 가격 6000
주문할 콤보 번호와 수량을 계속 입력하세요!
        0 주문 종료
        1 올인원팩  6000 원
        2 투게더팩  7000 원
        3 트리오팩  8000 원
        4 패밀리팩 10000 원
 >> 3 2
트리오팩, 2개 주문
트리오팩 주문 가격 16000, 총 가격 22000
주문할 콤보 번호와 수량을 계속 입력하세요!
        0 주문 종료
        1 올인원팩  6000 원
        2 투게더팩  7000 원
        3 트리오팩  8000 원
        4 패밀리팩 10000 원
 >> 0

****** 주문 종료 *******
총 주문 가격 22000 원
주문을 마치겠습니다.
======== 안녕! ========
```

Chapter 06 키와 값의 나열인 딕셔너리와 중복을 불허하는 집합

01 키와 값인 쌍의 나열인 딕셔너리

1. 딕셔너리의 개념과 생성

◆ 키와 값의 쌍을 항목으로 관리하는 딕셔너리

딕셔너리는 키와 값의 쌍인 항목을 나열한 시퀀스다. 즉, 딕셔너리는 콤마로 구분된 항목들의 리스트로 표현된다. 각각의 항목은 키:값과 같이 키와 값을 콜론으로 구분하고 전체는 중괄호 {…}로 묶는다.

- 딕셔너리의 항목 순서는 의미가 없으며, 키와 중복 될 수 없다.
- 키는 수정될 수 없지만, 값은 수정될 수 있다.
- 값은 키로 참조된다.

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

◆ 빈 딕셔너리의 생성과 항목 추가

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

```
>>> lect = {}
>>> print(lect)
{}

>>> lect = dict()
>>> lect['강좌명'] = '파이썬 기초'
>>> print(lect)
{'강좌명': '파이썬 기초'}
```

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.

```
>>> day = dict([]) # 빈 딕셔너리
>>> day = dict(()) # 빈 딕셔너리
```

함수 dict()의 리스트나 튜플 내부에서 일련의 키-값 쌍으로 [키, 값] 리스트 형식과 (키, 값) 튜플 형식을 모두 사용할 수 있다.

```
>>> day = dict([['월', 'monday'], ['화', 'tuesday'], ['수', 'wednesday']])
>>> day = dict([('월', 'monday'), ('화', 'tuesday'), ('수', 'wednesday')])
>>> print(day)
{'월': 'monday', '화': 'tuesday', '수': 'wednesday'}
```

◆ 키가 문자열이면 키 = 값 항목의 나열로도 딕셔너리 생성

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

```
>>> day = dict(월='monday', 화='tuesday', 수='wednesday')
>>> print(day)
{'월': 'monday', '화': 'tuesday', '수': 'wednesday'}
```

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. 새로운 키로 대입하면 항상 새로운 키-값의 항목이 삽입된다. 이미 있는 키로 항목을 참조하면 값이 반환된다.

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

◆ 튜플은 키로 가능하지만 리스트는 키로 사용 불가능

수정 불가능한 튜플은 딕셔너리의 키로 사용될 수 있다.

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

4. 딕셔너리 항목의 순회

◆ 딕셔너리 메소드 keys()

딕셔너리 메소드 keys()는 키로만 구성된 리스트를 반환한다.

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

◆ 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.

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

◆ 딕셔너리 메소드 values()

딕셔너리 메소드 values()는 값으로 구성된 리스트를 반환한다.

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

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.

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

◆ 키로 삭제하는 딕셔너리 메소드 pop(키[, 키가_없을_때_반환 값])

딕셔너리 메소드 pop(키)는 키인 항목을 삭제하고, 삭제되는 키의 해당 값을 반환한다.

```
>>> city = {'대한민국':'부산', '뉴질랜드':'웰링톤', '캐나다':'몬트리올'}
>>> print(city.pop('뉴질랜드'))
웰링톤
>>> city
{'대한민국': '부산', '캐나다': '몬트리올'}
>>> print(city.pop('중국', '없네요'))
없네요
>>> city
{'대한민국': '부산', '캐나다': '몬트리올'}
```

◆ 임의의 항목을 삭제하는 딕셔너리 메소드 popitem()

딕셔너리 메소드인 popitem()은 임의의(키, 값)의 튜플을 반환하고 삭제한다.

```
>>> city = {'대한민국':'부산', '뉴질랜드':'웰링톤'}
>>> print(city.popitem())
('뉴질랜드', '웰링톤')
>>> city
{'대한민국': '부산'}
>>> print(city.popitem())
('대한민국', '부산')
>>> city
{}
```

◆ 문장 del로 딕셔너리 항목 삭제

딕셔너리를 문장 del에 이어 키로 지정하면 해당 항목이 삭제된다.

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

6. 딕셔너리 항목 전체 삭제와 변수 제거

◆ 모든 항목을 제거하는 딕셔너리 메소드 clear()

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

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

◆ 문장 del로 딕셔너리 변수 자체 제거

딕셔너리를 문장 del에 이어 키로 저장하면 변수 자체가 메모리에서 제거 된다.

```
>>> city = {'대한민국':'부산', '뉴질랜드':'웰링톤'}
>>> del city
>>> city
Traceback (most recent call last):
  File "<pyshell#13>", line 1, in <module>
    city
NameError: name 'city' is not defined
```

7. 딕셔너리 결합과 키의 멤버십 검사 연산자 in

◆ 딕셔너리를 결합하는 메소드 update()

딕셔너리의 메소드 update()는 인자인 다른 딕셔너리를 합병한다. 인자 딕셔너리에 원 딕셔너리와 동일한 키가 있다면 인자 익셔너리의 값으로 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}
```

◆ 문장 in으로 쉽게 검사

문장 in으로 딕셔너리에 키가 존재하는지 간단히 검사할 수 있다. 값의 존재 여부는 확인할 수 없으므로 값으로 조회하면 항상 False다. 물론 not in도 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'}
```

02 중복과 순서가 없는 집합

1. 수학에서 배운 집합을 처리하는 자료형

◆ 원소는 유일하고 순서는 의미 없는 집합

파이썬에서 집합은 수학과 같이 원소를 콤마로 구분하며 중괄호로 둘러싸 표현한다.

- 원소는 불변 값으로 중복될 수 없으며 서로 다른 값이어야 한다.
- 즉, 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. 집합의 원소는 중복을 허용하지 않으므로 멤버십 검사와 중복 제거에 주로 사용될 수 있다.

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()에서 인자는 리스트와 튜플, 문자열처럼 반복적이면 가능하다. 그러나 리스트나 튜플의 항목은 변할 수 없는 것이어야 한다. 항목이 리스트나 딕셔너리처럼 가변적인 것은 허용되지 않는다.

◆ 함수 set() 호출로 공집합 만들기

내장 함수 set()으로 만들어지는 다음 집합 a는 공집합이며, 집합의 자료형 이름은 클래스 set이다. 공집합을 출력해보면 set()인 것을 알 수 있다.

```
>>> s = set()
>>> type(s)
<class 'set'>
>>> s
set()
```

◆ 리스트나 튜플을 인자로 사용하는 함수 set()

함수 set()에서는 인자로 리스트 또는 튜플 자체를 사용할 수 있으며, 결과는 시퀀스 항목에서 중복을 제거한 원소로 구성된다.

```
>>> set([1, 2, 3])
{1, 2, 3}
>>> set((1, 2, 2))
{1, 2}
>>> set(['a', 'b'])
{'a', 'b'}
```

◆ 문자열을 인자로 사용하는 함수 set()

함수 set()의 인자로 문자열이 사용되면 각각의 문자가 원소인 집합이 생성된다. 단, 집합이므로 순서는 의미가 없다.

```
>>> set('abc')
{'c', 'a', 'b'}
```

◆ 수정 가능한 리스트와 딕셔너리는 집합의 원소로 사용 불가

함수 set()의 인자에서 리스트나 튜플의 항목으로 수정될 수 있는 리스트나 딕셔너리는 허용되지 않는다.

3. 중괄호로 직접 원소를 나열해 집합 생성

◆ {원소1, 원소 2, …}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.

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

◆ {원소1, 원소2, …}에서 리스트나 딕셔너리는 원소로 사용 불가

리스트나 딕셔너리와 같이 가변적인 것은 원소로 사용할 수 없다.

4. 집합의 원소 추가와 삭제

◆ 집합 메소드 add(원소)로 추가

이미 만들어진 집합에 원소의 추가는 메소드 add(원소)를 사용한다.

```
>>> odd = {1, 3, 5}
>>> odd.add(7)
>>> odd.add(9)
>>> print(odd)
{1, 3, 5, 7, 9}
```

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

원소의 삭제는 메소드 remove()를 사용한다. 메소드 remove(원소) 호출에서는 삭제하려는 원소가 없으며녀 오류가 발생한다. 메소드 discard(원소)로도 원소를 삭제할 수 있으며, 원소가 없어도 오류가 발생하지 않는다. 임의의 원소를 삭제하려면 pop()을 사용해야 한다.

```
>>> odd = {1, 3, 5}
>>> odd.remove(3)
>>> print(odd)
{1, 5}
>>> odd.remove(9)
Traceback (most recent call last):
  File "<pyshell#42>", 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}
```

◆ 메소드 clear()로 집합의 모든 원소 삭제

집합의 모든 원소를 삭제하려면 메소드 clear()를 사용해야 한다.

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

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. 메소드 union()은 합집합을 반환하며 a 자체가 수정되지 않는다. 메소드 a.update(b)도 합집합과 같은 효과가 있으며, a와 b의 합집합 결과가 호출하는 집합 a에 반영 돼 수정되는 차이가 있다.

```
>>> 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 연산자와 &와 메소드 intersection()

양쪽 모든 집합에 속하는 원소로 구성되는 교집합은 연산자 &와 메소드 intersection()을 사용한다. a.intersection(b)는 집합 a, b 모두에 영향을 미치지 않는다. 그러나 a.intersection_update(b)는 a를 교집합으로 수정한다.

```
>>> 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}
```

◆ 차집합 연산자 - 와 메소드 difference()

피연산자인 왼쪽 집합에는 있지만 오른쪽에는 없는 원소로 구성되는 차집합은 연산자 -와 메소드 difference()를 사용한다. 피연산자의 순서에 따라 결과가 달라지므로 교환 법칙은 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 메소드 symmetric_difference()

한쪽 집합에는 소속되지만 교집합이 아닌 원소로 구성되는 여집합은 대칭 차집합이라고도 부르며, 연산자 ^와 메소드 symmetric_difference()를 사용한다.

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

6. 함수 len()과 소속 연산 in

◆ 집합 원소 수 함수 len()

함수 len()으로 집합에 들어 있는 원소의 개수를 확인할 수 있다.

```
>>> p1 = {'fortran', 'basic', 'cobol', 'c'}
>>> p2 = {'python', 'kotlin'}
>>> len(p1)
4
>>> len(p2)
2
```

◆ 멤버십 연산자 in

소속 연산자 in은 집합에서 유용하게 사용될 수 있다. 특정 원소가 집합에 있는지 확인하기 위해 in을 사용한다.

```
>>> 'fortran' in p1
True
>>> 'fortran' in p2
False
>>> 'python' not in p1
True
```

03 내장 함수 zip()과 enumerate(), 시퀀스 간의 변환

1. 내장 함수 zip()

◆ 리스트나 튜플 항목으로 조합된 항목을 생성하는 내장 함수 zip()

내장 함수 zip()을 이용하면 몇개의 리스트나 튜플의 항목으로 조합된 튜플을 만들 수 있다. zip()은 동일한 수로 이뤄진 여러 개의 튜플 항목 시퀀스를 각각의 리스트로 묶어 주는 역할을 하는 함수다.

함수 zip()의 결과는 자료형 zip이다. 자료형 zip은 간단히 리스트나 튜플로 변환 할 수 있다. zip()의 결과를 내장함수 list()의 인자로 사용하면 튜플인 리스트가 생성된다. zip()을 tuple()의 인자로 사용하면 항목이 튜플인 튜플이 생성된다. zip()에서 인자의 수는 2개 이상 올 수 있다. zip()의 인자들의 길이가 같지 않더라도 짧은 쪽의 인자에 맞는 리스트를 구성하고 긴 것은 무시한다.

```
>>> a = ['FTP', 'telnet', 'SMTP', 'DNS']
>>> b = (20, 23, 25, 53)
>>> z = zip(a, b)
>>> type(z)
<class 'zip'>
>>> list(zip(a, b))
[('FTP', 20), ('telnet', 23), ('SMTP', 25), ('DNS', 53)]
>>> tuple(zip(a, b))
(('FTP', 20), ('telnet', 23), ('SMTP', 25), ('DNS', 53))
>>> list(zip('ABCD', a, b))
[('A', 'FTP', 20), ('B', 'telnet', 23), ('C', 'SMTP', 25), ('D', 'DNS', 53)]
>>> tuple(zip('abcd', 'XY'))
(('a', 'X'), ('b', 'Y'))
```

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

내장 함수 zip()을 이용하면, 기존 리스트나 튜플의 조합으로 딕셔너리를 간단히 만들 수 있다. 내장 함수 dict()에서 zip()의 인자 2개를 사용하면, 앞은 키, 뒤는 값의 조합이 구성돼 딕셔너리가 생성된다.

```
>>> a = ['FTP', 'telnet', 'SMTP', 'DNS']
>>> b = (20, 23, 25, 53)
>>> dict(zip(a, b))
{'FTP': 20, 'telnet': 23, 'SMTP': 25, 'DNS': 53}
>>> dict(zip('ABCD', a))
{'A': 'FTP', 'B': 'telnet', 'C': 'SMTP', 'D': 'DNS'}
```

2. 내장 함수 enumerate()

◆ 첨자를 자동으로 만들어 첨자와 값과의 쌍인 튜플을 만들어 주는 내장 함수 enumerate()

내장 함수 enumerate(시퀀스)는 0부터 시작하는 첨자와 항목 값의 튜플 리스트를 생성한다. 키워드 인자 start를 사용해 enumerate(시퀀스, start = 1)로 호출하면 시작 첨자를 1로 지정할 수 있다. 키워드 start는 생략할 수 있다. 문자열 리스트도 사용할 수 있다.

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

함수 enumerate(시퀀스)는 for 반복의 시퀀스에 사용하는 것이 좋다. 변수 tp로 지정하면 tp[0]는 첨자, tp[1]은 값이 된다. 물론 format() 메소드에서 *tp로 지정하면 자동으로 나뉘어 앞부분의 {}에는 첨자, 뒤의 {}에는 값이 출력된다.

```
>>> subj = ['국어', '영어', '수학']
>>> 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
lst[2]: 수학
lst[2]: 수학
```

3. 시퀀스 간의 변환

◆ 튜플과 시퀀스 간의 변환

내장 함수 list()와 tuple(), set(), dict() 등을 사용하면 문자열을 비롯한 시퀀스를 간단하게 변환할 수 있다.

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

>>> singer = ['BTS', '봄빨간사춘기', 'BTS', '블랙핑크']
>>> print(singer)
['BTS', '봄빨간사춘기', 'BTS', '블랙핑크']
>>> print(tuple(singer))
('BTS', '봄빨간사춘기', 'BTS', '블랙핑크')
```

◆ 리스트와 집합 간의 변환

set(리스트) 함수를 사용해 리스트를 집합으로 변환한다. 리스트에서 중복된 항목은 집합으로 변환되면서 1개만 원소로 삽입된다. 순서도 무의미해진다.

```
>>> singer = ['BTS', '봄빨간사춘기', 'BTS', '블랙핑크']
>>> print(singer)
['BTS', '봄빨간사춘기', 'BTS', '블랙핑크']
>>> print(set(singer))
{'BTS', '블랙핑크', '봄빨간사춘기'}
```

◆ 딕셔너리를 다른 시퀀스로 변환하면 항목이 키로만 구성

딕셔너리를 list() 함수로 변환하면 항목이 키로만 구성된 리스트가 반환된다. 딕셔너리를 tuple() 함수로 튜플로 변환하면 항목이 키로만 구성된 튜플이 반환된다. 집합도 마찬가지로 순서는 의미가 없다.

```
>>> game = dict(일월='소나무', 이월='매화', 삼월='벚꽃', 사월='등나무')
>>> lgame = list(game)
>>> print(lgame)
['일월', '이월', '삼월', '사월']
>>> print(tuple(game))
('일월', '이월', '삼월', '사월')
>>> print(set(game))
{'이월', '일월', '삼월', '사월'}
```

프로젝트 Lab 1 철수와 영희의 가위바위보 게임

딕셔너리와 리스트 튜플 등을 활용해 철수와 영희의 가위바위보 게임을 구현해 보자. 철수와 영희에게 난수를 사용해 가위바위보 중 하나를 선정해 승부를 판정한다. 게임은 10회 연속하며 각각의 승부 내용을 출력한다. 구현하려는 가위바위보 게임에서 매우 유용한 구조가 딕셔너리다. 다음 딕셔너리 dcs는 가위바위보 게임의 승부를 결정하는 주요 구조로, 키:값의 쌍을 '가위':'보오'처럼 키로 이기는 패와 값으로 지는 패를 선택한다. 그러므로 항목은 3개가 나온다. 다만 한글 형식화 출력에 정렬 문제가 있어 보를 두 글자인 '보오'로 표현하자.

```
dcs = {'가위':'보오', '바위':'가위', '보오':'바위'}
```

철수와 영희가 서로 다른 것을 내고 철수가 낸 것을 키로 검색해 'dcs[철수 낸 것] == 영희 낸 것'이라면 철수가 승자가 된다. 이와 반대로 조건이 아니면 영희가 승리한 것으로 판정하면 된다. 매우 간단하지만 처음에는 익숙하지 않을 수 있다. 또한 모듈 random의 함수 choice()는 임의로 여러 가지 중 하나를 선택하는 데 활용되는 함수다. 다음과 같이 튜플이나 리스트인 rsp를 인자로 choice(rsp)를 호출하면 임의로 '가위', '바위', '보오' 중의 하나를 선택해 반환한다.

```
from random import choice
dcs = {'가위':'보오', '바위':'가위', '보오':'바위'}
tit = ['비김', '철수', '영희', '승자']
rsp = ('가위', '바위', '보오')

print('*'*17)
print('{:4} {:4} {:4}'.format(tit[1], tit[2], tit[3]))
print('*'*17)

for _ in range(10):
    cs = choice(rsp)
    yh = choice(rsp)

    print('{:4} {:4}'.format(cs, yh), end = ' ')

    if cs == yh:
        index = 0
    elif dcs[cs] == yh:
        index = 1
    else:
        index = 2
    print('{:4}'.format(tit[index]))
```

```
*****************
철수   영희   승자
*****************
바위   바위   비김
가위   가위   비김
보오   가위   영희
바위   가위   철수
가위   보오   철수
바위   바위   비김
가위   바위   영희
보오   가위   영희
바위   보오   영희
바위   보오   영희
```

프로젝트 Lab 2 딕셔너리로 만드는 K-pop 차트

딕셔너리로 가상의 K-pop 차트를 만들어 출력하는 프로그램을 작성해 보자. 우선 가수의 모음인 리스트와 노래의 모음인 리스트를 만든 후, 함수 zip()과 enumerate()를 사용한다. 함수 zip()으로 가수와 노래를 조합하고, 다시 이 조합된 리스트를 함수 enumerate()로 순위를 키로 하는 딕셔너리를 만든다.

```
{1: ('BTS', '작은 것들을 위한 시'),
 2: ('볼빨간사춘기', '나만 봄'),
 3: ('BTS', '소우주'),
 4: ('블랙핑크', 'Kill This Love'),
 5: ('태연', '사계')}
```

최종으로 만든 kpchart를 위와 같이 보기 좋게 출력하려면 모듈 pprint의 함수 pprint()을 사용한다.

```
import pprint
pprint.pprint(kpchart)


singer = ['BTS', '볼빨간사춘기', 'BTS', '블랙핑크', '태연', 'BTS']
song = ['작은 것들을 위한 시', '나만 봄', '소우주', 'Kill This Love', '사계']

kpop = list(zip(singer, song))
print(kpop)

kpchart = dict(enumerate(kpop, start = 1))

print(kpchart)
print()

import pprint
pprint.pprint(kpchart)

[('BTS', '작은 것들을 위한 시'), ('볼빨간사춘기', '나만 봄'), ('BTS', '소우주'), ('
블랙핑크', 'Kill This Love'), ('태연', '사계')]
{1: ('BTS', '작은 것들을 위한 시'), 2: ('볼빨간사춘기', '나만 봄'), 3: ('BTS', '소우
주'), 4: ('블랙핑크', 'Kill This Love'), 5: ('태연', '사계')}

{1: ('BTS', '작은 것들을 위한 시'),
 2: ('볼빨간사춘기', '나만 봄'),
 3: ('BTS', '소우주'),
 4: ('블랙핑크', 'Kill This Love'),
 5: ('태연', '사계')}
```