동양미래대학교
샤에디 ◆ 20191754 ◆ 컴퓨터정보공학과

2020 학년도 2 학기
교과목 포트폴리오

Python

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.

1.1 파이썬 언어란?

파이썬(Python)은 1990 년 암스테르담의 귀도 반 로섬(Guido Van Rossum)이 개발한 인터프리터 언어이다. 귀도는 파이썬이라는 이름을 자신이 좋아하는 코미디 쇼인 "몬티 파이썬의 날아다니는 서커스(Monty Python's Flying Circus)"에서 따왔다고 한다.

※ 인터프리터 언어란 한 줄씩 소스 코드를 해석해서 그때그때 실행해 결과를 바로 확인할 수 있는 언어이다.

파이썬의 사전적 의미는 고대 신화에 나오는 파르나소스 산의 동굴에 살던 큰 뱀을 뜻하며, 아폴로 신이 델파이에서 파이썬을 퇴치했다는 이야기가 전해지고 있다. 대부분의 파이썬 책 표지와 아이콘이 뱀 모양으로 그려져 있는 이유가 여기에 있다.

파이썬은 컴퓨터 프로그래밍 교육을 위해 많이 사용하지만, 기업의 실무를 위해서도 많이 사용하는 언어이다. 그 대표적인 예가 바로 구글이다. 필자는 구글에서 만든 소프트웨어의 50%이상이 파이썬으로 작성되었다는 이야기를 들었다. 이외에도 많이 알려진 예를 몇 가지 들자면 온라인 사진 공유 서비스 인스타그램(Instagram), 파일 동기화 서비스 드롭박스(Dropbox)등이 있다.

또한 파이썬 프로그램은 공동 작업과 유지 보수가 매우 쉽고 편하다. 그 때문에 이미 다른 언어로 작성된 많은 프로그램과 모듈이 파이썬으로 재구성되고 있다. 국내에서도 그 가치를 인정받아 사용자 층이 더욱 넓어지고 있고, 파이썬을 사용해 프로그램을 개발하는 업체들 또한 늘어 가고 있는 추세이다.

1.2 파이썬의 특징

필자는 파이썬을 무척 좋아한다. 모든 프로그래밍 언어에는 각기 장점이 있지만 파이썬에는 다른 언어에서는 쉽게 찾아볼 수 없는 파이썬만의 독특한 매력이 있다. 파이썬의 특징을 알면 왜 파이썬을 공부해야 하는지, 과연 시간을 투자할 만한 가치가 있는지 분명하게 판단할 수 있을 것이다.

파이썬은 인간다운 언어이다.

- 프로그래밍이란 인간이 생각하는 것을 컴퓨터에 지시하는 행위라고 할 수 있다. 앞으로 살펴볼 파이썬 문법에서도 보게 되겠지만, 파이썬은 사람이 생각하는 방식을 그대로 표현할 수 있는 언어이다. 따라서 프로그래머는 굳이 컴퓨터의 사고 체계에 맞추어서 프로그래밍을 하려고 애쓸 필요가 없다. 이제 곧 어떤 프로그램을 구상하자마자 머릿속에서 생각한 대로 술술 써 내려가는 여러분의 모습에 놀라게 될 것이다.

- 다음 소스 코드를 보면 이 말이 쉽게 이해될 것이다.

```
if 4 in [1,2,3,4]: print("4가 있습니다")
```

- 위 예제는 다음처럼 읽을 수 있다.

```
만약 4가 1, 2, 3, 4 중에 있으면 "4가 있습니다"를 출력한다.
```

- 프로그램을 모르더라도 직관적으로 무엇을 뜻하는지 알 수 있지 않는가? 마치 영어 문장을 읽는 듯한 착각에 빠져든다.

파이썬은 문법이 쉬워 빠르게 배울 수 있다.

- 어려운 문법과 수많은 규칙에 둘러싸인 언어에서 탈피하고 싶지 않은가? 파이썬은 문법 자체가 아주 쉽고 간결하며 사람의 사고 체계와 매우 닮아 있다. 배우기 쉬운 언어, 활용하기 쉬운 언어가 가장 좋은 언어가 아닐까? 유명한 프로그래머인 에릭 레이먼드(Eric Raymond)는 파이썬을 공부한 지 단 하루 만에 자신이 원하는 프로그램을 작성할 수 있었다고 한다.

※ 프로그래밍 경험이 조금이라도 있다면 파이썬의 자료형, 함수, 클래스 만드는 법, 라이브러리 및 내장 함수 사용 방법 등을 익히는 데 1 주일이면 충분하리라 생각한다.사용 방법 등을 익히는 데 1 주일이면 충분하리라 생각한다.

파이썬은 무료이지만 강력하다.

- 오픈 소스인 파이썬은 당연히 무료이다. 사용료 걱정없이 언제 어디서든 파이썬을 다운로드하여 사용할 수 있다.

※ 오픈 소스(Open Source)란 저작권자가 소스 코드를 공개하여 누구나 별다른 제한 없이 자유롭게 사용 · 복제 · 배포 · 수정할 수 있는 소프트웨어이다.

- 또한 프로그래머는 만들고자 하는 프로그램의 대부분을 파이썬으로 만들 수 있다. 물론 시스템 프로그래밍이나 하드웨어 제어와 같은 매우 복잡하고 반복 연산이 많은 프로그램은 파이썬과 어울리지 않는다. 하지만 파이썬은 이러한 약점을 극복할 수 있게끔 다른 언어로 만든 프로그램을 파이썬 프로그램에 포함시킬 수 있다.
- 파이썬과 C는 찰떡궁합이란 말이 있다. 즉 프로그램의 전반적인 뼈대는 파이썬으로 만들고, 빠른 실행 속도가 필요한 부분은 C로 만들어서 파이썬 프로그램 안에 포함시키는 것이다(정말 놀라우리만치 영악한 언어가 아닌가). 사실 파이썬 라이브러리 중에는 순수 파이썬만으로 제작된 것도 많지만 C로 만든 것도 많다. C로 만든 것은 대부분 속도가 빠르다.

※ 파이썬 라이브러리는 파이썬 프로그램을 작성할 때 불러와 사용할 수 있는 미리 만들어 놓은 파이썬 파일 모음이다.

1.3 파이썬으로 할 수 있는 일

파이썬으로 할 수 있는 일은 아주 많다. 대부분의 프로그래밍 언어가 하는 일을 파이썬은 쉽고 깔끔하게 처리한다. 파이썬으로 할 수 있는 일들을 나열하자면 끝도 없겠지만 대표적인 몇 가지 예를 들어 보겠다.

시스템 유틸리티 제작

파이썬은 운영체제(윈도우, 리눅스 등)의 시스템 명령어를 사용할 수 있는 각종 도구를 갖추고 있기 때문에 이를 바탕으로 갖가지 시스템 유틸리티를 만드는 데 유리하다. 실제로 여러분은 시스템에서 사용 중인 서로 다른 유틸리티성 프로그램을 하나로 뭉쳐서 큰 힘을 발휘하게 하는 프로그램들을 무수히 만들어낼 수 있다.

GUI 프로그래밍

GUI(Graphic User Interface) 프로그래밍이란 쉽게 말해 화면에 또 다른 윈도우 창을 만들고 그 창에 프로그램을 동작시킬 수 있는 메뉴나 버튼, 그림 등을 추가하는 것이다. 파이썬은 GUI 프로그래밍을 위한 도구들이 잘 갖추어져 있어 GUI 프로그램을 만들기 쉽다. 대표적인 예로 파이썬 프로그램과 함께 설치되는 Tkinter(티케이인터)가 있다. Tkinter를 사용하면 단 5줄의 소스 코드만으로 윈도우 창을 띄울 수 있다.

웹 프로그래밍

일반적으로 익스플로러나 크롬, 파이어폭스 같은 브라우저로 인터넷을 사용하는데, 누구나 한 번쯤 웹 서핑을 하면서 게시판이나 방명록에 글을 남겨 본 적이 있을 것이다. 그러한 게시판이나 방명록을 바로 웹 프로그램이라고 한다. 파이썬은 웹 프로그램을 만들기에 매우 적합한 도구이며, 실제로 파이썬으로 제작한 웹 사이트는 셀 수 없을 정도로 많다.

데이터베이스 프로그래밍

파이썬은 사이베이스(Sybase), 인포믹스(Infomix), 오라클(Oracle), 마이에스큐엘(MySQL), 포스트그레스큐엘(PostgreSQL) 등의 데이터베이스에 접근하기 위한 도구를 제공한다.

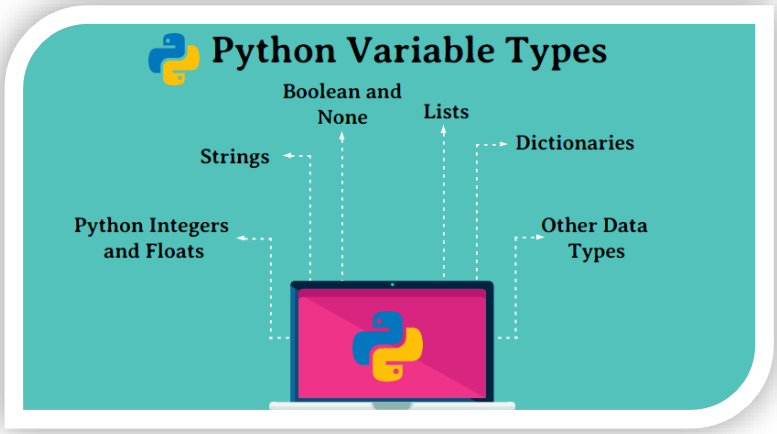
C/C++와의 결합

파이썬은 접착(glue) 언어라고도 부르는데, 그 이유는 다른 언어와 잘 어울려 결합해서 사용할 수 있기 때문이다. 나 로 만든 프로그램을 파이썬에서 사용할 수 있으며, 파이썬으로 만든 프로그램 역시 나 에서 사용할 수 있다.

데이터 분석, 사물 인터넷

파이썬으로 만든 판다스(Pandas) 모듈을 사용하면 데이터 분석을 더 쉽고 효과적으로 할 수 있다. 데이터 분석을 할 때 아직까지는 데이터 분석에 특화된 'R'이라는 언어를 많이 사용하고 있지만, 판다스가 등장한 이후로 파이썬을 사용하는 경우가 점점 증가하고 있다.

2.1 파이썬의 다양한 자료



I. 숫자형

숫자형(Number)이란 숫자 형태로 이루어진 자료형으로, 우리가 이미 잘 알고 있는 것이다. 우리가 흔히 사용하는 것을 생각해 보자. 123 같은 정수, 12.34 같은 실수, 드물게 사용하긴 하지만 8 진수나 16 진수 같은 것도 있다.

정수형

정수형(Integer)이란 말 그대로 정수를 뜻하는 자료형을 말한다. Python 은 서명된 정수를 가질 수 있다. 그것은 모든 길이의 값을 가질 수 있는데, 유일한 제한은 사용 가능한 메모리의 양이다.

```
>>> a = 123
>>> a = -178
>>> a = 0
```

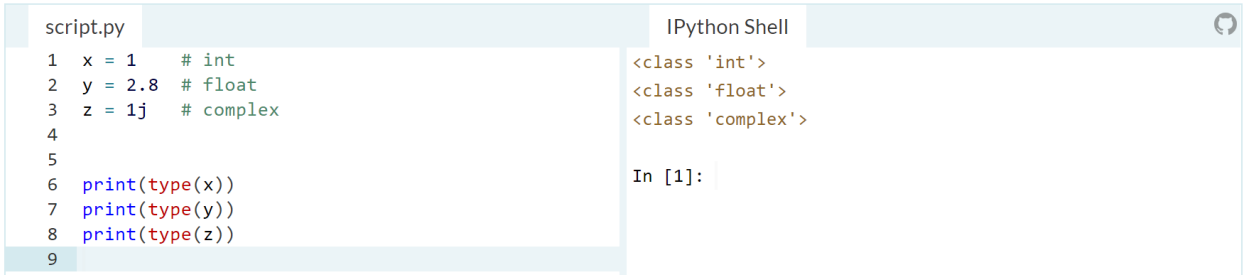
실수형

파이썬에서 실수형(Floating-point)은 소수점이 포함된 숫자를 말한다. 다음은 실수를 변수 a 에 대입하는 예이다. 실수 값은 소수점 15 자리까지만 정확하다. 그 후에는, 파이썬이 그 번호를 반올림한다.

```
>>> a = 1.2
>>> a = -3.45
```

허수 (복잡한 숫자)

허수(imaginary number)들은 표시 문자 "j"와 쓰여진다.



II. 문자열 자료형

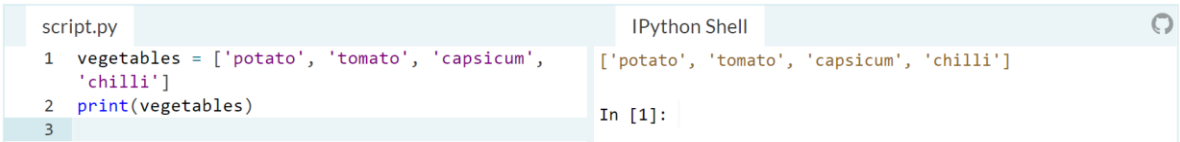
문자열은 단순히 문자의 연속이다. Python 에서는 따옴표 안에 있는 모든 것이 문자열로 간주되며, 다음과 같이 문자열 주위에 단일 따옴표 또는 큰따옴표를 사용할 수 있다.

```
"This is a string."
'This is also a string.'
```

III. 리스트 자료형

리스트는 특정 순서에 따른 항목의 모음입니다. 알파벳 문자, 0~9 의 숫자, 또는 가족 모두의 이름이 포함된 목록을 만들 수 있다.

원하는 것은 무엇이든 리스트에 넣을 수 있고, 리스트에 있는 항목은 어떤 특정한 방식으로든 연관될 필요가 없다. 리스트에는 보통 둘 이상의 요소가 포함되어 있기 때문에, 리스트의 이름을 letters, digits 또는 names 과 같이 복수형으로 만드는 것이 좋다.



IV. 튜플 자료형

리스트는 프로그램의 수명 동안 변경될 수 있는 항목 세트를 저장하는 데 효과적이다. 웹 사이트의 사용자 목록이나 게임의 캐릭터 목록과 함께 작업할 때 목록을 수정할 수 있는 기능은 특히 중요하다. 그러나 변경할 수 없는 항목의 목록을 만들고 싶을 때가 있다. 튜플은 우리가 단지 그것만 할 수 있도록 해준다. 파이썬은 불변(immutable)으로 변할 수 없는 값을 말하며 불변 리스트는 튜플이라고 한다.

항목을 쉼표로 구분하는 괄호() 안에서 정의한다.

```
box = (200, 50)
```

V. 딕셔너리 자료형

Python 의 딕셔너리는 키-값 쌍의 모음입니다. 각 키는 값에 연결되어 있으며, 키를 사용하여 해당 키와 관련된 값에 액세스할 수 있다. 키의 값은 숫자, 문자열, 리스트 또는 다른 딕셔너리일 수 있다. 사실 파이썬에서 만들 수 있는 어떤 오브젝트라도 사전의 값으로 사용할 수 있다.

```
fruits = {'value': tomato, 'key': red}
```

VI. 집합 자료형

집합(set)은 파이썬 2.3 부터 지원하기 시작한 자료형으로, 집합에 관련된 것을 쉽게 처리하기 위해 만든 자료형이다.

집합 자료형은 다음과 같이 set 키워드를 사용해 만들 수 있다.

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

VII. 불 자료형

불(bool) 자료형이란 참(True)과 거짓(False)을 나타내는 자료형이다. 불 자료형은 다음 2 가지 값만을 가질 수 있다.

- True - 참
- False – 거짓

2.2 파이썬의 문자열

문자열은 파이썬의 가장 인기 있는 면 중 하나이다. 모든 문자를 작은따옴표나 큰따옴표로 둘러싸면 간단히 문자열을 만들 수 있다.

```
"Life is too short, You need Python"
"a"
"123"
```

여러 줄인 문자열을 변수에 대입

1. 줄을 바꾸기 위한 이스케이프 코드 \n 삽입하기

```
>>> multiline = "Life is too short\nYou need python"
>>> print(multiline)
```

2. 연속된 작은따옴표 3 개(''') 또는 큰따옴표 3 개(""") 사용하기

```
>>> multiline='''
... Life is too short
... You need python
... '''
>>> print(multiline)
```

Output:

```
Life is too short
You need python
```

두 경우 모두 결과는 동일하다. 위 예에서도 확인할 수 있듯이 문자열이 여러 줄인 경우 이스케이프 코드를 쓰는 것보다 따옴표를 연속해서 쓰는 것이 훨씬 깔끔하다.

문자열 인덱싱

Python 의 다른 문자와 마찬가지로, 문자열에 사용되는 문자도 인텍스 번호와 같이, 각 문자가 1 바이트의 메모리를 차지하며 인덱스 면에서 주소를 가진다. 예를 들어, 내가 'python'이라는 단어로 구성된 줄을 가지고 있다면, 'python'의 각 문자는 별도의 인덱스와 메모리를 가지고 있다. ‘python'이라는 문자열에서 p 를 액세스하고 싶다고 가정해봅시다. 그러면 간단하게 다음과 같이 쓴다.

```
string = 'python'
print(string[0])
```

인덱싱은 항상 0 부터 시작되기 때문에 이 경우 첫 글자는 0 인덱스에 저장된 'p'가 된다. 위 코드의 출력은 다음과 같다.

```
p
```

문자열 슬라이싱

슬라이싱 방법을 통해 슬라이싱할 문자의 인덱스 번호를 선택하여 문자열을 자를 수도 있다. 아래는 나머지 문자 인덱스 번호에서 '0' 인덱스를 생략하여 p 를 python 에서 잘라 보는 예제이다.

```
#slicing p from python
string = 'python'
print(string[1:6])
```

Output:

```
ython
```

문자열의 양수 및 음수 인덱싱

0 부터 시작하는 숫자를 통해 문자열의 인덱싱을 양수로 할 수 있다는 것을 배웠다. 그러나 문자열은 역순으로도 음수를 사용하여 인덱싱을 할 수 있다. 예를 들어, 'python' 단어의 마지막 문자를 보려면 '-1' 인덱스를 입력하여 'python'에서 n 을 참고할 수 있다.

```
>>> string[-1:]
‘n’
>>> string[1:-1]
‘ython’
```

0	1	2	3	4	5
P	Y	T	H	O	N
-6	-5	-4	-3	-2	-1

슬라이싱을 통해 문자열 업데이트

슬라이싱 방법을 사용하여 현재 문자열을 다른 문자로 업데이트할 수 있다.

```
#updating a string through slicing
string = 'Python Tricks'
print("New String: " + string[0:6] + " Love")
```

Output:

```
Python Love
```

문자열 삭제

문자열 삭제를 할 수 있지만 문자열에서 문자를 삭제할 수는 없다. del 기능을 사용하여 전체 문자열을 삭제할 수는 있다는 점을 살펴보는 예제이다.

```
string = 'python'
del string
print(string)
# del 문을 사용하여 문자열을 삭제했으므로 출력이 null 임
```

Output:

```
Traceback (most recent call last):
  File "<stdin>", line 4, in <module>
    print(string)
NameError: name 'string' is not defined
```

파이썬의 문자열 연산

파이썬에는 다양한 데이터 유형과 변수를 사용하여 할 수 있는 많은 문자열 연산이 있다.

I. 문자열 연결(concatenate)

문자열에 연결하거나 '더하기'를 해서 하나의 새로운 문자열을 만들 때, 그 과정을 'concatenation' 또는 '연결하기'라고 한다. 더하기(+) 기호는 다음과 같은 두 문자열을 연결하거나 결합하는 데 사용된다.

```
string1 = 'I love'
string2 = ' Python Programming'
adding = string1 + string2
print(adding)
```

Output:

```
I love Python Programming
```

또한 * 기호를 사용하여 문자열을 연결할 수도 있다. 동일한 문자열을 여러 번 출력하려면 문자열을 특정 숫자로 곱해야 한다. 예를 들면 다음과 같다.

```
# Adding & multiplying two strings together
string1 = 'I love'
string2 = ' Python Programming '
adding = string1 + string2
multiplying = string1 * 3
print(adding)
print(multiplying)
```

Output:

```
I love Python Programming
I loveI loveI love
```

II. 문자열에 반복 사용

문자열에 있는 캐릭터들을 쉽게 통과하기 위해 for 문을 사용할 수 있다. 예를 들어, 문자열에 있는 문자 수를 세려면 다음을 수행하면 된다.

```
# Counting the number of characters of a string using a for loop
string = 'I love Python'
count = 0
for letter in string:
    count = count + 1
print(count)
```

Output:

```
13
```

III. 문자열 내에서 서브문자열 테스트

`in` 과 `not in` 연산을 사용하여 문자열 내에 하위 문자열이 존재하는지 여부를 쉽게 알 수 있다. 만일 하위 문자열이 문자열 내에 존재한다면 결과는 `bool` 형태로 True 가 된다. 마찬가지로, 하위 문자열이 문자열 내에 존재하지 않는 경우 `bool` 값은 False 가 된다. 예를 들면 다음과 같다.

```
# Checking whether a substring is present or not inside a string
if 'p' in 'python':
    print(True)
elif 'p' not in 'python':
    print(False)
```

Output:

```
True
```

문자열의 내장 함수

문자열 자료형은 자체적으로 함수를 가지고 있다. 이들 함수를 다른 말로 문자열 내장 함수라 한다. 이 내장 함수를 사용하려면 문자열 변수 이름 뒤에 '.'를 붙인 다음에 함수 이름을 써주면 된다. 다음 문자열의 내장 함수에 대해서 알아보는 예제이다.

- **문자 개수 세기(count)**

```
>>> a = "hobby"
>>> a.count('b')
2
```

문자열 중 문자 b 의 개수를 돌려준다.

- **위치 알려주기 1(find)**

```
>>> a = "Python is the best choice"
>>> a.find('b')
14
>>> a.find('k')
-1
```

문자열 중 문자 b 가 처음으로 나온 위치를 반환한다. 만약 찾는 문자나 문자열이 존재하지 않는다면 -1 을 반환한다. (파이썬은 숫자를 0 부터 세기 때문에 b 의 위치는 15 가 아닌 14 가 된다.)

- **위치 알려주기 2(index)**

```
>>> a = "Life is too short"
>>> a.index('t')
8
>>> a.index('k')
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
ValueError: substring not found
```

문자열 중 문자 t 가 맨 처음으로 나온 위치를 반환한다. 만약 찾는 문자나 문자열이 존재하지 않는다면 오류를 발생시킨다. 앞의 find 함수와 다른 점은 문자열 안에 존재하지 않는 문자를 찾으면 오류가 발생한다는 점이다.

- **문자열 삽입(join)**

```
>>> ",".join('abcd')
'a,b,c,d'
```

abcd 문자열의 각각의 문자 사이에 ','를 삽입한다.

join 함수는 문자열뿐만 아니라 앞으로 배울 리스트나 튜플도 입력으로 사용할 수 있다(리스트와 튜플은 곧 배울 내용이니 여기에서는 잠시 눈으로만 살펴보자). join 함수의 입력으로 리스트를 사용하는 예는 다음과 같다.

```
>>> ",".join(['a', 'b', 'c', 'd'])
'a,b,c,d'
```

- **소문자를 대문자로 바꾸기(upper)**

```
>>> a = "hi"
>>> a.upper()
'HI'
```

upper 함수는 소문자를 대문자로 바꾸어 준다. 만약 문자열이 이미 대문자라면 아무 변화도 일어나지 않을 것이다.

- **대문자를 소문자로 바꾸기(lower)**

```
>>> a = "HI"
>>> a.lower()
'hi'
```

lower 함수는 대문자를 소문자로 바꾸어 준다.

- **왼쪽 공백 지우기(lstrip)**

```
>>> a = " hi "
>>> a.lstrip()
'hi '
```

문자열 중 가장 왼쪽에 있는 한 칸 이상의 연속된 공백들을 모두 지운다. lstrip 에서 l 은 left 를 의미한다.

- **오른쪽 공백 지우기(rstrip)**

```
>>> a = " hi "
>>> a.rstrip()
' hi'
```

문자열 중 가장 오른쪽에 있는 한 칸 이상의 연속된 공백을 모두 지운다. rstrip 에서 r 는 right 를 의미한다.

- **양쪽 공백 지우기(strip)**

```
>>> a = " hi "
>>> a.strip()
'hi'
```

문자열 양쪽에 있는 한 칸 이상의 연속된 공백을 모두 지운다.

- **문자열 바꾸기(replace)**

```
>>> a = "Life is too short"
>>> a.replace("Life", "Your leg")
'Your leg is too short'
```

replace(바뀌게 될 문자열, 바꿀 문자열)처럼 사용해서 문자열 안의 특정한 값을 다른 값으로 치환해 준다.

- **문자열 나누기(split)**

```
>>> a = "Life is too short"
>>> a.split()
['Life', 'is', 'too', 'short']
>>> b = "a:b:c:d"
>>> b.split(':')
['a', 'b', 'c', 'd']
```

split 함수는 `a.split()`처럼 괄호 안에 아무 값도 넣어 주지 않으면 공백(스페이스, 탭, 엔터 등)을 기준으로 문자열을 나누어 준다. 만약 `b.split(':')`처럼 괄호 안에 특정 값이 있을 경우에는 괄호 안의 값을 구분자로 해서 문자열을 나누어 준다.
이렇게 나눈 값은 리스트에 하나씩 들어가게 된다. `['Life', 'is', 'too', 'short']`나 `['a', 'b', 'c', 'd']`가 리스트인데 02-3 에서 자세히 알아볼 것이니 여기에서는 너무 신경 쓰지 않아도 된다.

문자열 포매팅

프로그래밍하는 동안 파이썬에서는 문자열 내 문자 배열로 인해 구문 오류가 발생할 수 있다. 관련 문제를 접하고 오류를 해결하려면 문자열 포매팅이 필요하다. 예를 들어, 작은 따옴표 또는 큰 따옴표를 문자열에 두 번 사용하면 오류가 발생할 수 있다.

```
string = "He asked me, "What's your name?""
print(string)
```

큰 따옴표나 작은따옴표를 함께 사용하는 구문 오류를 해결하기 위해 여러 가지 방법이 있다. 작은 따옴표와 큰따옴표를 3 개(''' 또는 """) 따옴표로 묶거나 이스케이프 시퀀스 (\)를 사용하여 문제를 해결할 수 있다.

```
# Escape single quotes
print('He asked me, "What\'s your name?"')

# Escaping double quotes
print("He asked me, \"What's your name?\"")

# Using triple quotes
print('''He asked me, "What's your name?"''')
```

Output:

```
He asked me, "What's your name?"
He asked me, "What's your name?"
He asked me, "What's your name?"
```

2.3 파이썬의 리스트

리스트는 다른 데이터 유형의 시퀀스를 저장하는 데 사용된다. 파이썬은 리스트에 시퀀스를 저장할 수 있는 6 가지 데이터의 유형을 포함하며 리스트는 가장 보편적인 자료형이다.

리스트는 여러 데이터 유형의 값이나 항목의 모음으로 정의될 수 있다. 목록의 항목은 쉼표(,)로 구분되며, 모든 항목은 대괄호 [] 안에 캡슐화된다.

리스트를 다음과 같이 정의할 수 있다:

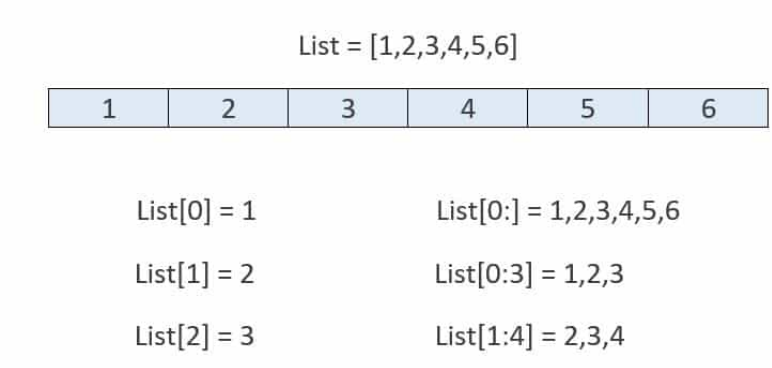
```
names = ["Hira", 302, "Paris"]
numbers = [1, 2, 3, 4, 5, 6]
random = [3, "John"]

print(names)
print(numbers)
print(random)
```

Output:

```
['Hira', 302, 'Paris']
[1, 2, 3, 4, 5, 6]
[3, 'John']
```

인덱싱은 문자열 처리와 동일한 방식으로 처리된다. 슬라이싱 연산자 []를 사용하여 리스트 구성 요소에 액세스할 수 있다. 지수는 0 에서 시작해서 -1 에서 끝난다. 리스트의 첫 번째 구성요소는 0 번째 인덱스에 배치되고, 리스트의 두 번째 구성요소는 1 번째 인텍스에 배치되는 등으로 확인할 수 있다.



파이썬은 우리에게 다른 언어와 달리 마이너스 인덱싱도 사용할 수 있는 유연성을 준다. 오른쪽부터 마이너스 인덱스를 센다. 리스트의 마지막 구성 요소(가장 오른쪽)는 인덱스-1 을 가지며, 다음 요소는 인덱스-2(오른쪽부터 왼쪽까지)에 위치하며, 왼쪽 구성 요소의 대부분을 찾을 때까지 계속된다.

목록에 항목 업데이트

리스트는 변형이 가능하며, 이는 문자열이나 튜플과 달리 그 요소들이 바뀔 수 있다는 것을 의미한다. 아이템이나 다양한 아이템을 변경하기 위해서는 할당 연산자(=)를 이용하면 된다. 예:

```
string = ['p','y','t','h','o','n']
```

```
del string[2]
print(string)
```

Output:
```
['p', 'y', 'h', 'o', 'n']
```

또한 파이썬은 리스트에서 어떤 구성요소를 제거해야 할지 모르는 경우 `remove()`나 `pop()` 메소드를 제공한다. `pop()` 메소드는 인덱스가 제공되지 않을 경우 리스트의 마지막 요소를 제거한다.

```
string = ['p','y','t','h','o','n']
string.remove('p')
string.pop()
print(string)
```

Output:
```
['y', 't', 'h', 'o']
```

리스트에 요소 추가

`add()` 메소드를 사용하여 기존 목록에 새 요소를 추가할 수 있다. `add()` 메소드를 사용하는 유일한 조건은 리스트 끝에만 새 요소를 추가할 수 있다는 것이다.

```
fruits = ['apple', 'banana', 'orange', 'kiwi']
fruits.append('grapes')
print(fruits)
```

Output:
```
['apple', 'banana', 'orange', 'kiwi', 'grapes']
```

리스트 반복

for loop 을 사용하여 리스트를 반복할 수 있다.

```
numbers = [1,2,3,4,5,6,7,8]
for n in numbers:
print(n)
```

Output:
```
1
2
3
4
```

```
5
6
7
8
```

2.4 파이썬의 튜플

튜플을 사용하여 불변하는 파이썬 객체의 시퀀스가 저장된다. 튜플(tuple)은 몇 가지 점을 제외하곤 리스트와 거의 비슷하며 리스트와 다른 점은 다음과 같다.

- 리스트는 [ ]으로 둘러싸지만 튜플은 ( )으로 둘러싼다.
- 리스트는 그 값의 생성, 삭제, 수정이 가능하지만 튜플은 그 값을 바꿀 수 없다.

```
tuple1 = (230, "James", 123)
tuple2 = ("red", "green", "blue")
```

튜플에 for loop 사용

for loop 을 사용하여 튜플을 반복할 수 있다.

```
tuple= ("apple", "orange", "lemon")
for n in tuple:
  print(n)
```

Output:

```
apple
orange
lemon
```

튜플에 항목 추가

튜플은 리스트과 달리 만들어진 튜플에 값을 추가할 수 없다는 불변의 특징을 가지고 있다. 즉, 리스트의 항목 값은 변화가 가능하고 튜플의 항목 값은 변화가 불가능하다. 예를 들면:

```
tuple1= ("apple", "orange", "lemon")
tuple1[3] = "banana"
# 오류
```

```
print(tuple1)
```

Output:

```
Traceback (most recent call last):
    File "/tmp/sessions/4ee7c8963287ca22/main.py", line 2, in  tuple1[3] =
    "banana"
TypeError: 'tuple' object does not support item assignment
```

따라서 프로그램이 실행되는 동안 그 값이 항상 변하지 않기를 바란다거나 값이 바뀔까 걱정하고 싶지 않다면 튜플을 사용해야 한다.

튜플 삭제

튜플 내의 항목은 삭제할 수 없지만 전체 튜플은 삭제할 수 있다. 예를 들면 다음과 같다.

```
tuple1= ("apple", "orange", "lemon")
del tuple1
print(tuple1)
```

Output:

```
Traceback (most recent call last):
    File "/tmp/sessions/550f21853aa12970/main.py", line 3, in
    print(tuple1)
NameError: name 'tuple1' is not defined
```

튜플의 음수 인덱싱

리스트와 마찬가지로 튜플도 오른쪽(-1)부터 왼쪽(-1)까지 마이너스 인덱싱이 있다. 예:

```
tuple1= ("apple", "orange", "lemon")
print(tuple1[-1])
```

Output:

```
lemon
```

튜플 값 변경

튜플이 불변의 것이라는 것을 알고 있듯이, 튜플 내부의 값을 바꿀 수 없다는 것을

의미한다. 그러나 튜플을 리스트로 변환한 다음 그 리스트의 값을 변경하고 다시 그 특정 리스트를 튜플로 변경할 수 있다. 예:

```
tuple1= ("apple", "orange", "lemon")
tuple2 = list(tuple1)
tuple2[1] = "banana"
tuple1 = tuple(tuple2)
print(tuple1)
```

Output:

```
('apple', 'banana', 'lemon')
```

2.5 파이썬의 딕셔너리

딕셔너리이란 무엇인가?

딕셔너리는 순서가 없고, 변경이 가능하며, 인덱스화된 모음입니다. 딕셔너리는 파이썬에서 중괄호를 사용하여 만들어지며, 키-값 쌍으로 구성되어 있다. 다음과 같이 간단한 딕셔너리를 만드는 예제이다.

```
book= { "pages": "277", "name": "Gone Girl", "year": 2007 }
print(book)
```

Output:

```
{'pages': '277', 'name': 'Gone Girl', 'year': 2007}
```

딕셔너리 내 요소 액세스

딕셔너리의 '값'을 접근하기 위해 항상 '키' 이름을 사용함으로써 딕셔너리 내부의 요소를 접근할 수 있다.

```
book= {
  "pages": "277",
  "name": "Gone Girl",
  "year": 2007
}
x = book['name']
print(x)
```

Output:
```
Gone Girl
```

사전에서 요소 변경

특정 항목의 키를 참조하여 특정 항목의 값을 변경할 수 있다. 예:

```
book= {
"pages": "277",
"name": "Gone Girl",
"year": 2007
}

book['year'] = '2010'
print(book)
```

Output:
```
{'name': 'Gone Girl', 'pages': '277', 'year': '2010'}
```

반복을 사용하여 키 및 값 출력

간단한 for loop 을 사용하여 다음과 같은 방법으로 모든 키를 출력할 수 있다.

```
book= {
  "pages": "277",
  "name": "Gone Girl",
  "year": 2007
}
for n in book:
  print(n)
```

Output:
```
pages
name
year
pages
name
year
```

일반적으로 loop 을 사용하면 키만 출력되는데 값을 별도로 출력하고 싶을 때 이름 끝에 `values()` 메소드를 사용하여 다음과 같이 값을 출력할 수 있다.

```
book= {
```

```
  "pages": "277",
  "name": "Gone Girl",
  "year": 2007
}
for n in book.values():
  print(n)
```

Output:

```
2007
277
Gone Girl
```

for loop 을 사용하여 키-값 쌍으로 키와 값을 모두 출력하려면 다음과 같이 items() 메소드로 n, m 변수에 저장하여 출력할 수 있다.

```
book= {
  "pages": "277",
  "name": "Gone Girl",
  "year": 2007
}
for n, m in book.items():
  print(n,m)
```

Output:

```
name Gone Girl
pages 277
year 2007
```

사전에서 항목 제거

`pop()` 함수를 사용하여 딕셔너리의 항목도 삭제할 수 있다. 예를 들어 'book' 내의 'name'이라는 항목을 삭제하려면 `pop()` 괄호 안에 '키'의 이름을 넣으면 된다.

```
book= {
  "pages": "277",
  "name": "Gone Girl",
  "year": 2007
}
book.pop('name')
print(book)
```

Output:

```
{'pages': '277', 'year': 2007}
```

del 함수를 사용하여 딕셔너리 또는 전체 딕셔너리의 항목을 삭제할 수도 있다.

```
book= { "pages": "277",
"name": "Gone Girl",
"year": 2007 }
del book['pages']
print(book)
```

Output:

```
{'name': 'Gone Girl', 'year': 2007}
```

마찬가지로 델 키워드를 사용하여 전체 사전도 삭제할 수 있다.

```
book= { "pages": "277",
"name": "Gone Girl",
"year": 2007 }
del book
print(book)
```

Output:

```
Traceback (most recent call last):
    File "/tmp/sessions/11ab166edc475687/main.py", line 7, in
      print(book)
NameError: name 'book' is not defined
```

2.6 파이썬의 집합

집합 자료형은 다음과 같이 set 키워드를 사용해 만들 수 있다.

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

위와 같이 set()의 괄호 안에 리스트를 입력하여 만들거나 다음과 같이 문자열을 입력하여 만들 수도 있다.

```
>>> s2 = set("Hello")
>>> s2
{'e', 'H', 'l', 'o'}
```

비어 있는 집합 자료형은 s = set()로 만들수 있다.

위에서 보면 set("Hello")의 결과가 좀 다르다. 분명 "Hello" 문자열로 set 자료형을 만들었는데 생성된 자료형에는 l 문자가 하나 빠져 있고 순서도 뒤죽박죽이다. 그 이유는 set 에 다음과 같은 2 가지 큰 특징이 있기 때문이다.

- 중복을 허용하지 않는다.
- 순서가 없다(Unordered).

리스트나 튜플은 순서가 있기(ordered) 때문에 인덱싱을 통해 자료형의 값을 얻을 수 있지만 set 자료형은 순서가 없기(unordered) 때문에 인덱싱으로 값을 얻을 수 없다.

교집합, 합집합, 차집합 구하기

다음과 같이 2 개의 set 자료형이다.

```
>>> s1 = set([1, 2, 3, 4, 5, 6])
>>> s2 = set([4, 5, 6, 7, 8, 9])
```

I. 교집합

s1 과 s2 의 교집합을 구해 보자.

```
>>> s1 & s2
{4, 5, 6}
```

"&" 기호를 이용하면 교집합을 간단히 구할 수 있다.

또는 다음과 같이 intersection 함수를 사용해도 동일한 결과를 돌려준다.

```
>>> s1.intersection(s2)
{4, 5, 6}
```

s2.intersection(s1)을 사용해도 결과는 같다.

II. 합집합

합집합은 다음과 같이 구할 수 있다. 이때 4, 5, 6 처럼 중복해서 포함된 값은 한 개씩만 표현된다.

```
>>> s1 | s2
{1, 2, 3, 4, 5, 6, 7, 8, 9}
```

"|" 기호를 사용한 방법이다.

```
>>> s1.union(s2)
{1, 2, 3, 4, 5, 6, 7, 8, 9}
```

또는 union 함수를 사용하면 된다. 교집합에서 사용한 intersection 함수와 마찬가지로 s2.union(s1)을 사용해도 동일한 결과를 돌려준다.

III. 차집합

차집합은 다음과 같이 구할 수 있다.

```
>>> s1 - s2
{1, 2, 3}
>>> s2 - s1
{8, 9, 7}
```

빼기(-) 기호를 사용한 방법이다.

```
>>> s1.difference(s2)
{1, 2, 3}
>>> s2.difference(s1)
{8, 9, 7}
```

difference 함수를 사용해도 차집합을 구할 수 있다.

집합의 관련 함수

- **값 1 개 추가 add()**

  이미 만들어진 set 자료형에 값을 추가할 수 있다. 1 개의 값만 추가(add)할 경우에는 다음과 같이 한다.

  ```
  >>> s1 = set([1, 2, 3])
  >>> s1.add(4)
  >>> s1
  {1, 2, 3, 4}
  ```

- **값 여러 개 추가 update()**

  여러 개의 값을 한꺼번에 추가(update)할 때는 다음과 같이 하면 된다.

  ```
  >>> s1 = set([1, 2, 3])
  >>> s1.update([4, 5, 6])
  >>> s1
  {1, 2, 3, 4, 5, 6}
  ```

- **특정 값 제거하기 remove()**

  특정 값을 제거하고 싶을 때는 다음과 같이 하면 된다.

```
>>> s1 = set([1, 2, 3])
>>> s1.remove(2)
>>> s1
{1, 3}
```

2.7 파이썬의 불

불 데이터 유형은 True 또는 False 이다. Python 에서 불 변수는 True 및 False 키워드로 정의된다.

```
>>> a = True
>>> type(a)
<class 'bool'>

>>> b = False
>>> type(b)
<class 'bool'>
```

노트: True 와 False 키워드는 대문자를 포함해야 함을 기억해야 한다. 소문자 true 를 사용하면 오류가 반환된다.

불 산술 연산자

파이썬의 일반적인 불 연산자는 다음과 같다.

- or
- and
- not
- == (equivalent)
- != (not equivalent)

아래 코드 섹션에서는 두 변수에 True 와 False 라는 불 값이 할당된다. 그런 다음 이 불 값을 불 연산자와 결합하고 조작한다.

```
>>> A = True
>>> B = False
>>> A or B
```

```
True
>>> A and B
False
>>> not A
False
>>> not B
True
>>> A == B
False
>>> A != B
True
```

and, or, not 불 연산자는 괄호 안에 결합하여 복합 불 식을 만들 수 있다.

```
>>> C = False
>>> A or (C and B)
True
>>> (A and B) or C
False
```

불 함수 bool()

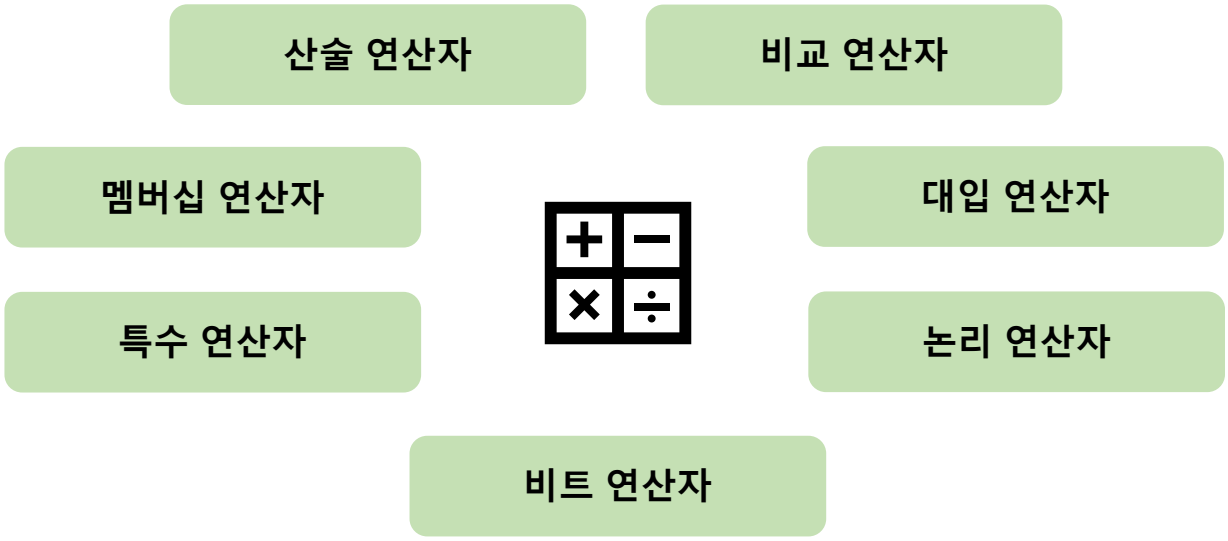
bool 내장 함수를 사용하면 자료형의 참과 거짓을 식별할 수 있다.

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

3.1 기본 연산자 개요

파이썬은 로직을 기반으로 구축되기 때문에 기본적인 연산을 지원한다.

파이선의 연산자는 7 가지 범주로 나뉜다.



3.2 산술 연산자

이름 그대로 산술 연산자는 덧셈, 뺄셈, 나눗셈, 모듈로, 거듭제곱, 곱셈과 같은 간단한 계산을 하는 데 쓰인다.

연산자	의미	예제
+	더하기	x + y = 30
–	빼기	x – y = -10
*	곱하기	x * y = 200
/	나누기	x / y = 2.0
//	나누기(floor) – 정수 몫	x // y = 3
%	나누고 나머지 반환	x % y = 0
**	누승 제곱	x ** y = 1000

```
#Taking two numbers a and b
a = 3
b = 5
# Addition
addition = a + b
# Subtraction
subtraction = a - b
# Division
division = a / b
# Multiplication
multiplication = a * b
# Exponential Power
exponential = a**b
# Modulo
modulo = a % b
# Printing the variables
print(addition)
print(subtraction)
print(multiplication)
print(division)
print(exponential)
print(modulo)
```

Output:

```
8
-2
15
0.6
243
3
```

3.3 비교 연산자

비교 연산자는 참 또는 거짓으로 출력 특성을 확인하여 값을 비교하고 결과를 반환하는 데 사용한다.

연산자	연산 사용	의미	문자열 관계 연산
>	a > b	크다(greater than)	사전 순서(lexicographically)에서 뒤에(코드 값이 크다)
>=	a >= b	크거나 같다 (greater than or equal to)	사전 순서에서 뒤에(코드 값이 크다) 있거나 동일
<	a < b	작다(less than)	사전 순서에서 앞에(코드 값이 작다)
<=	a <= b	작거나 같다 (less than or equal to)	사전 순서에서 앞에(코드 값이 작다) 있거나 동일
==	a == b	같다(equal to)	사전 순서에서 동일
!=	a != b	다르다(not equal to)	사전 순서와 다름

```
# Comparsion operators
x = 10
y = 4
# When x > y
print(x > y)
# When x < y
print(x < y)
# When x == y
print(x == y)
# When x != y
print(x != y)
# When x >= y
print(x >= y)
# When x <= y
print(x <= y)
```

Output:

```
True
False
False
True
True
False
```

3.4 대입 연산자

할당 연산자는 변수에 값을 할당하는 데 사용된다.

연산자	설명	예제/의미
=	y 의 결과값을 변수 x 에 저장	x = y
+=	x + y 의 결과값을 변수 x 에 저장	x += y x = x + y
-=	x - y 의 결과값을 변수 x 에 저장	x -= y x = x - y
*=	x * y 의 결과값을 변수 x 에 저장	x *= y x = x * y
/=	x / y 의 결과값을 변수 x 에 저장	x /= y x = x / y
//=	x // y 의 결과값을 변수 x 에 저장	x //= y x = x // y
%=	x % y 의 결과값을 변수 x 에 저장	x %= y x = x % y
**=	x ** y 의 결과값을 변수 x 에 저장	x **= y x = x ** y

```
>>> var = 5
>>> var += 2
>>> print(var)
7
>>> var -= 3
>>> print(var)
4
>>> var *= 5
>>> print(var)
20
>>> var //= 3
>>> print(var)
6
>>> var **= 2
>>> print(var)
36
```

```
>>> var %= 7
>>> print(var)
1
>>> var /= 5
>>> print(var)
0.2
```

3.5 논리 연산자

논리 연산자는 피연산자 간에 AND (&), NOT (^) 및 OR (|) 기능을 수행한다.

연산자	설명	예제
and 또는 &	두 값이 모두 참이면 참으로 변환됨	x and y
or 또는 |	피연산자 중 하나 또는 둘 다 참일 경우 참으로 변환됨	x or y
not 또는 ^	피연산자가 거짓일 경우 참으로 변환됨	not x

```
>>> True and True, True and False, False and False,
(True, False, True)
>>> True or True, True or False, False or False,
(True, True, False)
>>> not True, not False, not False and True
(False, True, True)
```

3.6 비트 연산자

비트 연산자는 비트별로 연산을 수행한다.

연산자	설명	예제
&	a 와 b 의 비트를 AND 연산	x & y
|	a 와 b 의 비트를 OR 연산	x | y
^	a 와 b 의 비트를 XOR 연산(배타적 OR, Exclusive OR)	x ^ y
~	x 의 비트를 뒤집음	~x
<<	x 의 비트를 y 번 왼쪽으로 이동시킴	x << y
>>	x 의 비트를 y 번 오른쪽으로 이동시킴	x >> y

```
>>> bin(0b1101 & 0b1001)    # 비트 AND
'0b1001'
>>> 13 & 9                  # 비트 AND
```

```
9
>>> bin(0b1101 | 0b1001)    # 비트 OR
'0b1101'
>>> 13 | 9                  # 비트 OR
13
>>> bin(0b1101 ^ 0b1001)    # 비트 XOR
'0b100'
>>> 13 ^ 9                  # 비트 XOR
4
>>> bin(~0b1101)            # 비트 NOT
'-0b1110'
>>> ~13                     # 비트 NOT
-14
>>> 0b0011 << 2    # 비트를 왼쪽으로 2 번 이동
12
>>> bin(12)
'0b1100'
>>> 0b1100 >> 2    # 비트를 오른쪽으로 2 번 이동
3
>>> bin(3)
'0b11'
```

3.7 특수 연산자

ID 연산자: 동일한 메모리 위치에 두 피연산자가 있는지 여부를 확인하는 데 사용된다.

연산자	설명	예제
is	두 피연산자가 동일한 경우 참으로 변환	x is y
is not	두 피연산자가 동일하지 않은 경우 변환	x is not y

```
x1 = 5
y1 = 5
x2 = 'Hello'
y2 = 'Hello'
x3 = [1,2,3]
y3 = [1,2,3]
print(x1 is not y1)
print(x2 is y2)
print(x3 is y3)
```

Output:

```
False
True
False
```

3.8 멤버십 연산자

멤버십 연산자는 값이나 피연산자가 어떤 시퀀스에 존재하는지 여부를 확인해 주는 연산자이다.

연산자	설명	예제
in	시퀀스에서 피연산자를 찾을 경우 참으로 전환됨	x in y
not in	시퀀스에서 피연산자를 찾을 수 없는 경우 참으로 전환됨	x not in y

```
x = 'Hello world'
y = {1:'a',2:'b'}

print('H' in x)
print('hello' not in x)
print(1 in y)
print('a' in y)
```

Output:

```
True
True
True
False
```

4.1 조건문, 반복문, 제어문 개요

조건문 (conditional statement)

특정 조건이 충족되는 경우에만 코드를 실행하고자 할 때 의사결정(decision making)이 필요하다. 따라서 파이썬에서 의사결정을 수행하고 싶을 때 조건문을 사용한다.

- `if statement`
- `if-else statement`
- `else-if statement`

반복문 (iteration statement)

반복문은 매번 다른 값으로 동일한 코드를 반복해서 실행해야 할 때 사용된다.

반복문을 사용하면 조건이 거짓일 때까지 동일한 코드가 실행된다.

- `while loop`
- `for loop`
- `nested loop`

제어문 (control statement)

제어문은 반복문(loop)의 정상적인 행동 또는 흐름을 바꿔 주는 문이다 . 파이썬에는 사용할 수 있는 제어문의 유형이 3 가지가 있다.

- `break`
- `continue`

4.2 파이썬의 조건문

if 문

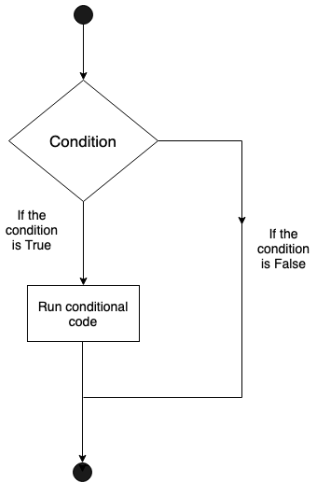
구문:

```
if expression:
    statement
```

expression 이 True 를 반환하면 블록 코드(조건문)가 실행되고, 문장이 거짓이면 expression 이 False 로 반환되며, 프로그램은 조건문을 실행하지 않고 종료된다.

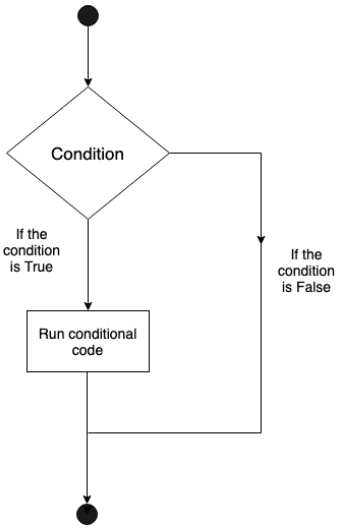
```
>>> switch = True
>>> if switch:
...     print("The light is on!")
```

위에 코드는 switch 가 참일 때만 아래 문장이 실행되고 출력이 있다. switch 가 거짓이면 애래 문장이 실행되지 않는다.



반복문을 만들 때는 if expression: 바로 아래 문장부터 if 문에 속하는 모든 문장에 들여쓰기(indentation)를 해주어야 한다.

if-else 문



구문:

```
if expression:
    statement
else:
    statement
```

이 문장은 조건이 참인지 확인하고 조건 기준에 맞지 않으면 else 문장을 실행한다.

```
a = 200
b = 33
if b > a:
    print("b is greater than a")
else:
    print("b is not greater than a")
# Output:
# b is not greater than a
```

else-if 문

구문:

```
if expression1:
    statement
elif expression2:
    statement
elif expression3:
    statement
else:
    statement
```

else-if 문은 if-else 문과 유사하지만 우리가 원하는 만큼의 코드를 여러 개의 'elif' 블록 안에 실행할 수 있는 자유를 준다.

```
>>> name = 'Joe'
>>> if name == 'Fred':
```

```
...     print('Hello Fred')
... elif name == 'Xander':
...     print('Hello Xander')
... elif name == 'Joe':
...     print('Hello Joe')
... elif name == 'Arnold':
...     print('Hello Arnold')
... else:
...     print("I don't know who you are!")
...
Hello Joe
```

4.3 파이썬의 반복문

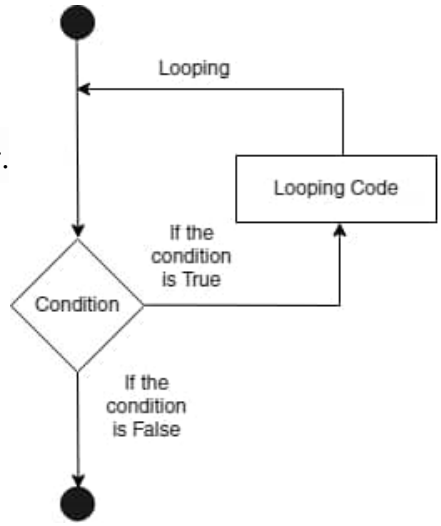
왜 반복문이 필요한가?

우리가 프로그램을 반복해야 하는 주된 이유는 복잡한 문제를 더 단순하게 만들기 위해서 하는 것이다. 우리가 동일한 코드를 여러 번 실행해야 할 때 각 질의를 다시 작성하면 매우 번거로운 일이다.

반복문을 사용하면 시간을 절약할 수 있고 매번 동일한 기능을 수행해야 할 때 동일한 코드를 쓸 필요가 없으며 반복문을 사용하면 코드가 깨끗해 보인다는 것이 주요 장점이다.

while 문

파이썬의 while 문은 주어진 조건이 참인 동안 코드를 계속 실행하는데, 조건이 거짓이 되면, while 문이 정지된다. While 문은 몇 번 반복해야 할지 확실하지 않을 때 많이 사용된다.



구문:

```
while expression:
    statements
```

위의 구문에서는 while 문은 표현을 확인하고 시작되는데, 표현이 참이면 아래 문장을 실행한다. 반복을 한 후 다시 표현으로 돌아가 표현이 False 가 될 때까지 문장을 계속 실행한다.

script.py

```
1  n = 1
2  while n < 10:
3      print(n)
4      n += 1
```

IPython Shell

```
1
2
3
4
5
6
7
8
9
```

While 문 + else

파이썬은 우리에게 while loop 과 함께 else 문장을 사용할 수 있게 해 준다. 만약 그 동안 표현이 False 로 판명된다면, 우리는 프로그램이 더 이상 실행되지 않는 것을 사용자에게 알려주기 위해 else 문을 사용한다.

script.py

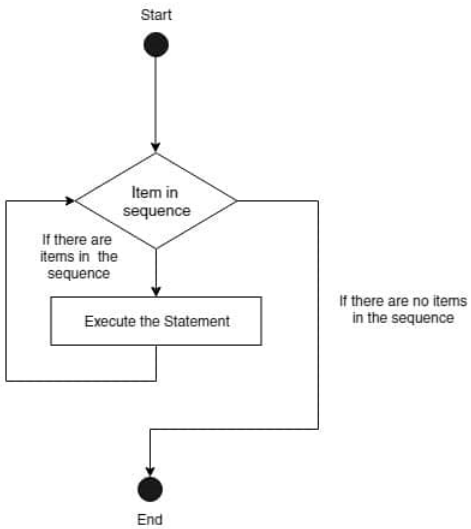
```
1  n = 1
2  while n < 10:
3      print(n)
4      n += 1
5  else:
6      print("The program has ended since n is
   greater than 10")
```

IPython Shell

```
1
2
3
4
5
6
7
8
9
The program has ended since n is greater than 10
```

for 문

for 문은 주어진 횟수 동안 명령어 그룹을 반복하는 문이다. 반복 횟수를 알 때 항상 for loop 을 사용하는 것이 좋은 방법이다.



구문:

```
for variable in sequence:
    statement 1
    statement 2 ...
```

script.py

```
1  fruits = ['apple','grapes', 'oranges',
   'bananas']
2  for fruit in fruits:
3      print(fruit)
```

IPython Shell

```
apple
grapes
oranges
bananas
```

for 문의 range() 함수 사용

우리는 for 문을 사용할 때 range() 함수를 사용할 수 있으며, 그 함수에서는 매개변수로 반복할 횟수를 정의하면 된다.

script.py

```
1  #print natural numbers upto 10
2  for n in range(11):
3      print(n)
```

IPython Shell

```
0
1
2
3
4
5
6
7
8
9
10
```

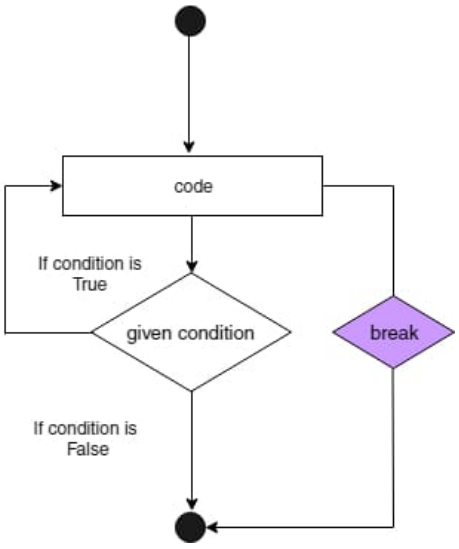
range() 함수

↪ range(*start, stop, step*)

start	**시작할 위치를 지정하는 정수 번호. 기본값은 0. (생략 가능)**
stop	중지할 위치(포함되지 않음)를 지정하는 정수. (필수)
step	증분을 지정하는 정수. 기본값은 1. (생략 가능)

4.4 파이썬의 제어문

Break 문



break 문은 반복문을 종료하거나 폐기하는 데 사용된다. 반복문이 깨지면, ‘break’ 문 후의 다음 문장은 계속 실행된다. 이유는 중첩된 반복문(즉, 반복문 내의 반복문)에서 빨리 빠져나가기 위한 문이다. break 문은 while 문 또는 for 문과 함께 사용된다.

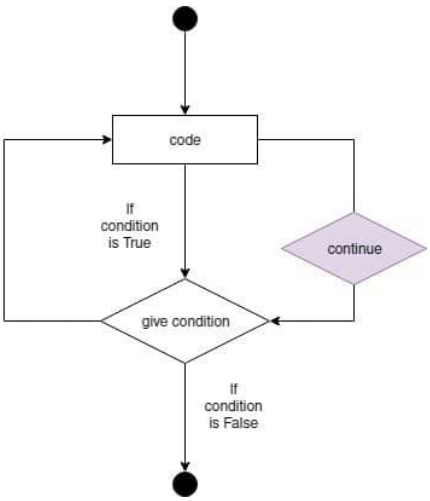
```
country_name = "\nPlease enter the name of the country you love: "
while True:
    country = input(country_name)
    if country == 'quit':
        break
else:
    print("My favorite country is: " + country.title())
```

Output:

```
Please enter the name of the country you love: France
My favorite country is: France
Please enter the name of the country you love: quit
# 종료
```

Continue 문

파이썬의 continue 문은 프로그램이 실행되는 동안 중단이 발생한 후에도 프로그램을 계속 실행해 주는 문이다. Continue 문은 코드가 반복 내에서 진행되며 특정 조건을 건너뛰고 이동할 수 있도록 한다.




```
for letter in 'Py thon':
    if letter == ' ':
        continue
    print ('Letters:', letter)
```

Output:

```
Letters: P
Letters: y
Letters: t
Letters: h
Letters: o
Letters: n
```

{ 마무리말 }

This is just my thought on python. Honestly, after learning this much about python language, I can say that I really like python! Python is so convenient and simple. It's amazing how things that are done with other programming languages can be so simple with python. It doesn't even take many lines when we code a complex program. It's one of the reason why coding with python can significantly reduce programmers' coding time. I did a lot of research about python before taking this class and after taking this class, I didn't regret it at all. I believe that even after I finish this semester or graduate from university, python will still have a place in me, for me to continue learning it and nurture strong concept towards the language.

I hope that one day I will master this programming language. ^^