PORTFOLIO

동양미래대학교

파이썬 프로그래밍

-중간고사 대체과제-

과목명	파이썬프로그래밍
담당교수	강환수 교수님
학과	컴퓨터정보공학과
학년	2학년
학번	20190683
이름	이나윤
제출일	2020.05.31.일 까지

PORTFOLIO

목차

Ⅰ. 파이썬 언어의 개요와 개발환경

1. 파이썬 언어의 개요

◆ 파이썬은 배우기 쉬운 프로그래밍 언어

- 파이썬은 배우기 쉽고 누구나 무료로 사용할 수 있는 오픈 소스(open source)[1] 프로그래밍(programming language)[2] 언어다. 파이썬은 1991년 네덜란드의 귀도 반 로섬이 개발했으며, 현재는 비영리 단체인 파이썬 소프트웨어 재단(PSF)이 관리하고 있다.

- 파이썬의 사전적 의미는 '비단뱀'으로, 그리스 신화에서 유래됐으며, 파이썬 로고가 비단뱀인 것은 바로 이 때문이다. 그런데 실제로는 개발자인 귀도가 애청하던 영국의 코미디 프로그램 'Monty Python's Flying Circus'에서 따온 것이라고 한다.

◆ 현재 파이썬의 인기는 최고다

- 파이썬은 현재 미국과 우리나라의 대학 등 전 세계적으로 가장 많이 가르치는 프로그래밍 언어 중 하나다. 특히 비전공자의 컴퓨팅 사고력(computational thinking)[3]을 키우기 위한 프로그래밍 언어로도 많이 활용되고 있다. 파이썬은 배우기 쉽고 간결하며, 개발속도가 빠르고 강력하기 때문이다. 또한 파이썬은 라이브러리(library)[4]가 풍부하고 다양한 개발 환경을 제공하고 있어 개발자가 쉽고 빠르게 소프트 웨어를 개발하는 데 도움을 준다.

- 파이썬은 프로그래밍 교육 분야에서뿐 아니라 실무에서도 사용이 급증하고 있으며, 이를 증명이라도 하듯 프로그래밍에 대해 질문하고 답하는 사이트로 유명한 스택오버플로(stackoverflow.com)에서 '가장 빠르게 성장하는 프로그래밍 언어'로 선택되기도 했다.

---

1 소프트웨어를 만든 프로그램 소스를 모두 공개한 것을 말한다. 파이썬의 공개된 소스는 깃허브 사이트에서 볼 수 있다.

2 배틀 그라운드와 같은 컴퓨터 게임이나 스마트폰의 다양한 앱 프로그램을 만드는 언어를 '프로그래밍 언어'라고 한다.

3 컴퓨팅의 기본 개념과 원리를 기반으로 문제를 효율적으로 해결하는 사고 능력으로, 누구나가 배워야 할 기본 역량이다.

4 도서관에서 책을 대여하듯 프로그램 작성 시 자주 사용하는 부분을 가져다 쓸 수 있는 부분 프로그램인 모듈(module)을 말한다. 라이브러리는 대개 함수(function)나 클래스(class)의 모임이다.

2. 컴퓨터 사고력과 파이썬

◆ 지금은 지능 · 정보화 혁명 시대인 제 4차 산업혁명 시대

- 우리는 정보와 통신 기술의 발달로 디지털 및 정보화 혁명이 일어난 제3차 산업혁명 시대를 거쳐 스마트폰의 개발과 인공지능(AI: Artificial Intelligence), 사물 인터넷(IoT: Internet of Things), 빅 데이터(big data) 등의 발전으로 제 4차 산업혁명 시대에 살고 있다. 즉, 제 4차 산업혁명이란 '모든 사물이 연결된 초연결(hyper connected) 사회에서 생산되는 빅데이터를 기존 산업과 융합해 인공 지능, 클라우드 등의 첨단 기술로 처리하는 정보 · 지능화 혁명 시대'라고 할 수 있다.

◆ 문제 해결 능력과 창의 · 융합 사고 능력에 필요한 컴퓨팅 사고력

- 제 4차 산업혁명 시대는 모든 사물이 연결된 초연결 사회이며, 모든 산업은 컴퓨터 과학 · 공학 기술을 기반으로 융합될 것이다. 결국 모든 산업 분야에서 컴퓨터 과학 기술을 필요로 하게 된다는 것이다. 그러므로 미래의 인재는 컴퓨터 과학 원리와 개념을 활용해 자신의 영역과 융합할 수 있는 역량을 갖춰야 한다. 이러한 능력을 컴퓨팅 사고력(CT : Computational Thinking)이라고 하는데, 이는 '컴퓨터 과학의 기본 개념과 원리 및 컴퓨팅 시스템을 활용해 실생활 및 다양한 학문 분야의 문제를 이해하고 창의적 해법을 구현해 적용할 수 있는 능력'으로 정의할 수 있다.

- 컴퓨팅 사고력은 컴퓨터 분야의 문제 해결은 물론, 일상생활에서의 문제 해결에 효율적으로 사용될 수 있는 방법을 제공할 뿐 아니라 창의성을 높이는 데에도 도움이 된다. 따라서 컴퓨팅 사고력 교육은 제 4차 산업혁명 시대의 인재 교육에서 가장 중요하다. 컴퓨팅 사고력은 컴퓨터 전공자에게만 해당하는 것이 아니라 모든 사람에게 필요한 기본 역량이다.
다시 말해, 우리가 어릴 때부터 글을 읽고, 쓰고, 더하기, 빼기 등의 간단한 셈을 하는 것과 마찬가지로 모든 사람에게 컴퓨팅 사고력이 필요하다.

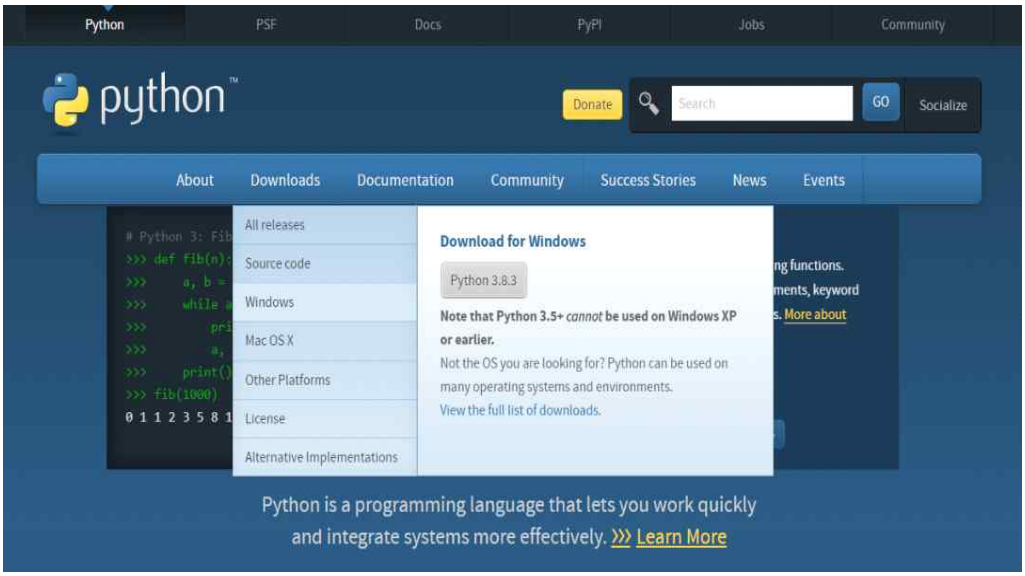
◆ 컴퓨터 사고력 구성 요소

- 영국의 컴퓨팅 교육에서는 컴퓨팅 사고력을 분해, 패턴 인식, 추상화, 알고리즘 설계를 구성 요소로 제안하고 있다.
  - 분해(decomposion): 데이터, 프로세스 또는 문제를 작고 관리 가능한 부분으로 나눔
  - 패턴 인식(pattetn recognition): 데이터의 패턴, 추세 및 정규성 관찰
  - 추상화(abstraction): 패턴을 생성하는 일반 원칙을 규정
  - 알고리즘 설계(algorithm design): 이 문제와 유사한 문제 해결을 위한 단계별 지침을 개발

3. 개발 환경 구성

◆ 자신의 운영체제에 적합한 파이썬 설치

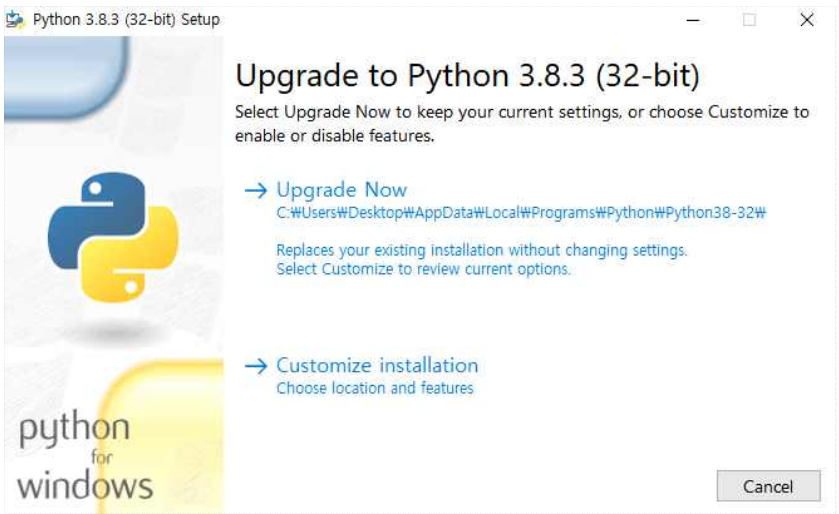
- 윈도우 사용자라면 파이썬 홈페이지에서 **download**를 누르면 나타나는 메뉴에서 **Python3.o.o**을 클릭해 설치한다.
  파이썬 다운로드 페이지(www.python.org/downloads)로 이동한 후 **Download Python3.o.o**을 눌러 설치할 수도 있다.
  실행 버튼을 눌러 바로 실행하거나 **저장**을 눌러 설치 파일을 저장한 후에 설치할 수도 있다.
  다운로드가 완료되면 자동 설치 파일인 **python-3.o.o.exe**를 실행하자.



▲ **파이썬 개발 도구의 다운로드와 실행**

- 파이썬 설치 첫 화면에서 **Install Now**를 선택하면 간단하게 설치할 수 있다. 쉬운 사용과 자동 path를 설정하기 위해 하단에 위치한 2개의 체크박스를 모두 선택한 후, 지정된 폴더에 설치하는 **Install Now**를 선택한다.
  파이썬 개발 도구가 설치되는 폴더는 윈도우 로그인 계정이름의 하부임을 기억하자.

■ **기본 파이썬 설치 폴더 :** C:₩Users₩Desktop₩AppData₩Local₩Programs₩Python₩Python37-32

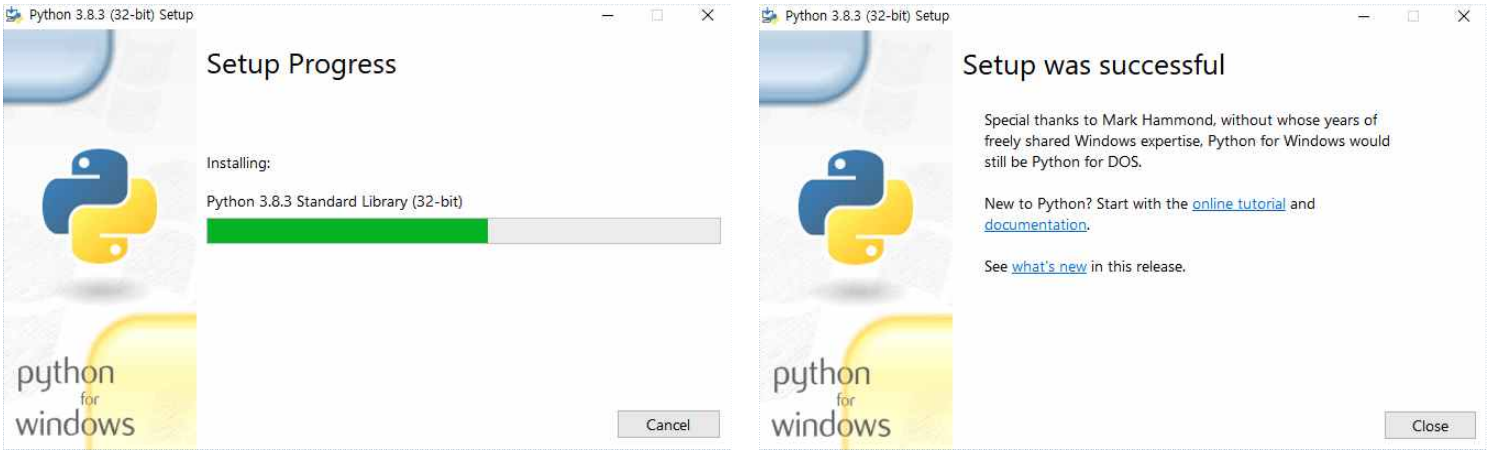


▲ **파이썬 설치 시작 화면**

- **Install Now**를 이용해 설치(docs.python.org/3/using/windows.html)할 때는 다음 사항에 유의하자.

  - 표준 라이브러리와 파이썬 쉘인 IDLE 및 pip, 매뉴얼 문서(documentation) 등이 설치
  - 체크박스 **Add Python 3.7 to PATH** 옵션을 선택하면 파이썬 설치 폴더가 PATH에 설정

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



▲ **파이썬 설치 과정과 종료 화면**

◆ 설치한 파이썬 쉘의 실행과 종료

- 설치된 파이썬 메뉴를 찾아 실행해 보자. 시스템의 [설치 프로그램 메뉴]를 표시한 후, 실행하는 방법은 두가지가 있다.

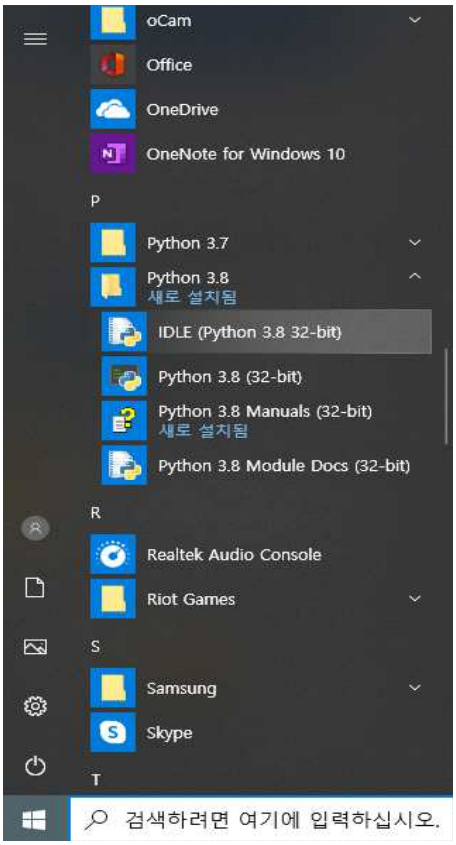
  - 윈도 화면의 하단에 위치한 **[작업 표시줄]**의 가장 왼쪽에 있는 **[윈도 시작]** 버튼 선택
  - 키보드의 **windows로고 키** 선택

- '파이썬 쉘 IDLE[5]'이 정상적으로 실행되면 오른쪽에 『**Python 3.o.o Shell**』 창이 표시된다. 메뉴 **[Help] - [About IDLE]**을 선택하면 IDLE이 파이썬의 통합 개발 및 학습 환경을 의미한다는 것을 알 수 있으며, 여러 정보가 담긴 창이 표시된다. 실행된 『**파이썬 쉘 IDLE**』은 메뉴 **[File] - [Exit]** 메뉴 또는 단축키 **Ctrl + Q**로 종료할 수 있다.

- 『**파이썬 쉘 IDLE**』의 가장 상단에는 버전 정보와 help, copyright 등을 이용할 수 있다는 두세 줄의 메시지가 표시된다. 메시지 이후 첫 칸에 프롬프트라 부르는 3개의 부호 기호인 >>>가 표시되며, 키보드 입력을 기다리는 커서가 깜빡인다. 프롬프트는 개발자로부터 쉘의 명령이나 파이썬 문장을 기다린다는 의미를 가진 기호다. 『**파이썬 쉘 IDLE**』과 같은 모습의 창을 **'대화형 창'** 또는 **'REPL(read – eval – print – loop) 창'**이라고도 한다.

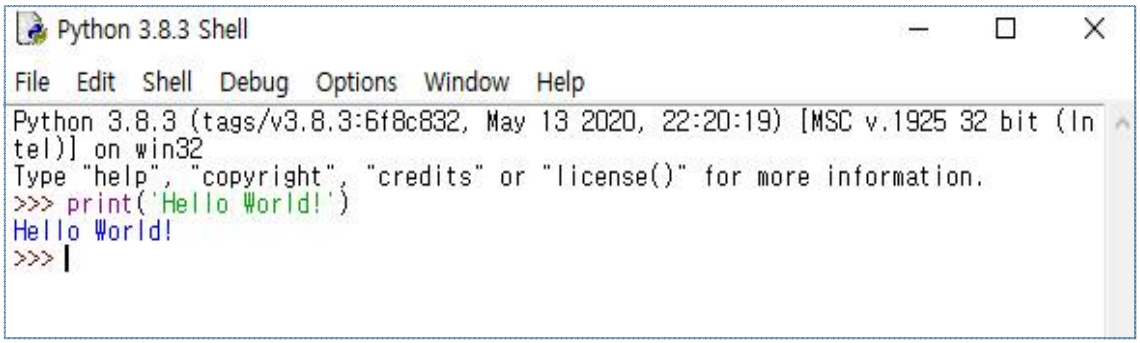
5 일반적인 통합 개발 환경을 IDLE, '아이 디 이'라고 발음하는데, IDLE는 간단히 '아이들'이라고 발음한다.

▼ **파이썬의 실행 메뉴와『파이썬 쉘 IDLE』**



◆ 글자 Hello World!를 출력하는 첫 대화형 코딩

- 파이썬의 첫 코딩으로 글자 Hello World!를 쉘에 출력하는 프로그램을 작성해보자.
  - 'Hello World!'와 같이 출력하고자 하는 글자의 앞뒤에 작은따옴표(또는 홑따옴표)를 붙인다.
  - 작은따옴표를 붙인 글자를 문자열 또는 스트링(string)이라 한다.
  - print('Hello World!')를 입력한 후 Enter를 눌러 결과를 알아보자.



- 모든 명령어는 첫 칸부터 입력해야 한다. 첫 칸뿐 아니라 그 이상을 **공간(space)으로 두고 입력하면 무조건 'SyntaxError: unexpected indent'가 표시**된다. 맨 앞부분의 메시지가 모두 빨간색으로 표시되고 SyntaxError라는 메시지가 나타나는 것으로 보아 문제가 있는 듯하다.

  맞다. **무조건 첫 칸부터 입력해야 한다.**

Chapter 02 문자열 연산자

◆ 문자열의 연결 연산자 + 와 반복 연산자 +

- 더하기 기호인 +는 문자열에서 문자열을 연결(concatenation)하는 역할을 한다.
- 별표 *는 문자열에서 문자열을 지정된 수만큼 반복하는 연산을 수행한다. 반복 횟수인 정수가 하나 있어야 하므로 **'파이썬' * '언어'**처럼 두 문자열로만 반복연산자를 사용하면 오류가 발생한다.

8 lines (8 sloc) | 328 Bytes — Raw | Blame | History

```
>>> print("원의 원주율 " + '3.11592')
원의 원주율 3.141592
>>> print("python" 'programming ' + 'language')
python programming language
>>> print('파이썬 언어는' + " 강력하다")
파이썬 언어는 강력하다
>>> print('파이썬 ' + "언어! " + 3 * "방가")
파이썬 언어! 방가 방가 방가
```

▲ **문자열 연결과 반복 연산자 +, *의 활용 예제 2 – 1 코드**

◆ 정수와 실수의 다양한 연산

◆ 수의 더하기, 빼기, 곱하기, 나누기 연산자 + - * /

- 파이썬 쉘은 간단한 계산기처럼 사용할 수 있으며, 나누기 결과는 항상 실수다.
- 수의 연산인 더하기, 빼기, 곱하기와 나누기가 가능하며 다만 곱하기는 * 나누기는 /를 사용한다. 이러한 연산자를 모두 산술 연산자라 부른다.

◆ 수의 몫, 나머지, 지수승 연산자 //, %, **

- 이러한 연산자를 사용한 수는 정수와 실수 모두 가능하다.

4 lines (3 sloc) | 101 Bytes — Raw | Blame | History

```
print(4 ** (1/2))
print( ((3**2 + 4**2)) ** (1/2) )
print( ((2-3.1)**2 + (5-4.8)**2) ** (1/2) )
```

▲ **두 점 사이의 거리 계산 연산자 활용 예제 2 – 3코드**

Chapter 02 변수와 키워드, 대입 연산자

◆ 자료형과 type( ) 함수

- 지금까지 알아본 정수와 실수, 문자열 등을 자료형(data type 또는 자료 유형)이라 한다.
  - 파이썬에서 자료형을 살펴보면 **정수는 int**, **실수는 float** 그리고 **문자열은 str**로 사용한다.

◆ 변수의 이해와 대입 연산자 =을 이용한 값의 저장

- 변수(variable)란, 말 그대로 **'변하는 자료를 저장하는 메모리 공간'**이다. 변수는 자료를 담을 수 있는 그릇이며, 그릇에 이름인 태그가 붙어 있다고 생각하면 이해하기 쉽다.
- 값을 변수에 저장하기 위해서는 대입 연산자(=)가 필요하다. =의 의미는 왼쪽 화살표 (←)라고 생각하자. 즉, 오른쪽 값을 왼쪽 변수에 저장하라는 의미이다. 대입 연산자의 왼쪽에는 반드시 저장 공간인 변수가 와야한다.
  - 변수 이름은 저장 값에 의미 있는 이름을 붙이자.
- 변수뿐 아니라 나중에 배울 함수와 클래스 이름 등 프로그래머가 이름을 짓는 단어를 식별자(identifiers)라 하며 다음 조건을 만족해야 한다.
  - 문자는 대소문자의 영문자(a, b, c, A, B ···), 숫자(0, 1, 2, ···) 그리고 _로 구성되며, 대소문자는 구별된다.
  - 숫자는 맨 앞에 올 수 없다. 그러므로 영문자로 시작하자.
  - import, True, False 등과 같은 키워드는 사용할 수 없다.

6 lines (6 sloc) | 259 Bytes — Raw | Blame | History

```
1  celsius = 37
2  fahrenhite = celsius * 9/5 + 32 # 화씨로 변환
3  print('섭씨: ', celsius, ',', '화씨: ', fahrenhite)
4  celsius += 3 # 5도 증가
5  fahrenhite = celsius * 9/5 +32 # 화씨로 변환
6  print('섭씨: ', celsius, ',', '화씨: ', fahrenhite)
```

▲ **섭씨 온도를 화씨 온도로 변환하는 변수, 대입연산자 활용 예제 2 – 8 코드**

Chapter 02 자료의 표준 입력과 자료 변환 함수

◆ 표준 입력이란?

- 프로그램 작성 과정에서 사용자의 입력을 받아 처리해야 하는 경우 즉, **프로그램 과정에서 쉘이나 콘솔에서 사용자의 입력을 받아 처리하는 방식**을 표준 입력 (standard input)이라 한다.
  - **함수 input( )은 입력되는 표준 입력을 문자열로 읽어 반환하는 함수**이다. input( )에서 반환되는 입력 문자열을 변수에 저장하려면 대입 연산자 =을 사용해 변수에 저장해야 한다.
  - **문자열과 정수, 실수 간의 자료 변환 함수 str( ), int( ), float( )**

3 lines (3 sloc) | 101 Bytes — Raw | Blame | History

```
1  univ = input('대학은? ')
2  name = input('이름은? ')
3  print('대학: ', univ, '이름: ', name)
```

▲ **학교와 이름을 입력받아 출력한 표준 입력 활용 예제 2 – 11 코드**

◆ 16진수, 10진수, 8진수, 2진수의 활용

- 수의 상수를 표현하는 방법으로 10진법뿐 아니라 16진법, 8진법, 2진법 등이 있다. **16진수(hexadecimal)는 맨 앞에 0x(숫자 0과 알파벳 x)**로 시작하며, **8진수(octal)와 2진수(binary)는 각각 0o(숫자 0과 알파벳 o)와 0b(숫자 0과 알파벳 b)**로 시작한다. 각각의 **진수를 표시하는 알파벳은 대소문자 모두 가능**하다.
  - 10진수를 바로 16진수, 8진수, 2진수로 표현되는 문자열로 변환하려면 각각 함수 **bin( ), oct( ), hex( )**을 사용해야 한다.

6 lines (5 sloc) | 245 Bytes — Raw | Blame | History

```
1  data = int(input('정수 입력 >> '))
2
3  print('2진수: ', bin(data)) # 2진수로 출력
4  print('8진수: ', oct(data)) # 8진수로 출력
5  print('10진수: ', data) # 10진수로 출력
6  print('16진수: ', hex(data)) # 16진수로 출력
```

▲ **10진수 정수를 입력받아 2,8,10,16진수를 출력하는 활용 예제 2 – 15 코드**

◆ 문자열 str 클래스와 부분 문자열 참조 슬라이싱

◆ 문자열은 클래스 str의 객체

- 파이썬에서 문자열은 '문자의 나열'로 텍스트 시퀀스(text sequence)라고도 한다. 문자열의 자료형은 class str이다. 'python'과 같은 문자열 상수는 클래스 str의 객체다. 이미 학습한 문자열 표시 방법을 정리하면 다음과 같다.

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

◆ 함수 len( )으로 문자열의 길이 참조

- 함수 len(문자열)으로 문자열의 길이를 알 수 있다.

6 lines (6 sloc) | 212 Bytes — Raw | Blame | History

```
1  str = 'Hello python!'
2  n = len(str)
3  print('문자열', str, '길이', n)
4  print('첫 문자', str[0], str[-n] )
5  print('가운데 문자', str[n//2], str[-n//2] )
6  print('마지막 문자', str[n-1], str[-1] )
```

▲ **첨자로 문자열의 문자 참조의 활용 예제 3 - 1 코드**

◆ 문자열 부분 문자열 참조 방식

◆ 0과 양수를 이용한 문자열 슬라이싱

- 문자열에서 일부분을 참조하는 방법을 슬라이싱(slicing)이라고 한다. 문자열에서 잘라 일부분을 참고한다고 생각하면 된다. 물론 기존의 문자열은 절대 수정이 되는 것이 아니라 **원 문자열은 변함이 없고 일부분을 반환**한다.

  - **문자열 슬라이싱 str[start:end]: 문자열 str에서 start 첨자에서 end-1 첨자까지의 문자열을 반환**

◆ 음수를 이용한 문자열 슬라이싱

- 슬라이스 [start:end]에서는 음수도 사용할 수 있으며, start 첨자에서 end-1 첨자까지의 문자열을 반환한다.

8 lines (8 sloc) | 365 Bytes — Raw | Blame | History

```
1  str = '일요일 거러기'
2  print(str[:3], str[4:]) # 양수 이용
3  print(str[:-4], str[-3:]) # 음수 이용
4  print(str[:], str[::], str[::1]) # 모든 문자열 참조
5  print(str[::2]) # 한 문자씩 건너뛰기
6  print(str[::3]) # 두 문자씩 건너뛰기
7  print(str[::-2]) # 역순으로 한 문자씩 건너뛰기
8  print(str[::-1]) # 역순으로 출력
```

▲ **다양한 문자열 슬라이싱 활용 예제 3 - 3 코드**

문자열 바꿔 반환하는 메소드 replace( )

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

부분 문자열 출현 횟수를 반환하는 메소드 count( )

```
>>> str = '단순한 것이 복잡한 것보다 낫다.'
>>> str.count('복잡')
1
```

문자와 문자 사이에 문자열을 삽입하는 메소드 join( )

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

문자열을 찾는 메소드 find( )와 index( )

- find( )와 index( ) 모두 클래스 str에서 부문 문자열 sub의 맨 처음에 위치한 첨자를 반환시킨다. **메소드 index( )**는 찾는 문자열이 없을 경우, **ValueError를 발생**시키지만 **find( )**는 오류가 발생시키지 않고 **–1을 반환** 시킨다.

```
>>> str = '자바 C 파이썬 코틀린'
>>> str.find('자바')
0
>>> str.index('자바')
0
```

문자열을 여러 문자열로 나누는 split( ) 메소드

```
>>> '사과 배 복숭아 딸기 포도'.split( )
['사과', '배', '복숭아', '딸기', '포도']
```

문자열의 format( ) 메소드를 이용해 간결한 출력 처리

```
>>> '3 + 4 = 7'
'3 + 4 = 7'
>>> str = '{ } + { } = { }'.format(3, 4, 3 + 4)
>>> print(str)
3 + 4 = 7
```

메소드 format( )의 다양한 형식 지정

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

Chapter 03 논리 자료와 다양한 연산

◆ 논리곱과 논리합 연산자 and와 or

- 논리곱인 **and와 & 연산자는 두 항이 모두 참이어야 True**다. 하나라도 거짓이면 False다.

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

- 논리곱인 **or와 | 연산자는 두 항이 모두 거짓이어야 False**다. 하나라도 참이면 True다.

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

◆ 배타적 논리합 연산자 ^와 not 연산자

- 배타적 논리합(exclusive or) 연산자인 ^은 **두 항이 다르면 True, 같으면 False**이다.

```
>>> True ^ True
False
>>> True ^ Flase
True
```

- 연산자 not은 뒤에 위치한 논리 값을 바꾼다.

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

Chapter 04 조건에 따른 선택 if … else

◆ 조건의 논리 값에 따른 선택 if

◆ 조건에 따른 선택을 결정하는 if문

- 조건에 따라 해야 할 일(문장들)을 처리해야 하는 경우, if문을 사용한다. if문은 조건인 논리 표현식의 결과가 True이면 이후에 구성된 블록 구문인 문장들을 실행한다. 그러나 결과가 False이면 실행하지 않는다.
  - if문에서 논**리 표현식 이후에는 반드시 콜론**이 있어야 한다.
  - 콜론 이후 다음 줄부터 시작되는 블록은 반드시 **들여쓰기**를 해야 한다. 그렇지 않으면 오류가 발생한다.

```
weather = '화창'
if weather == '화창' :
    print('어제 산 신발을 신고 가야지!')
```

3 lines (3 sloc) | 124 Bytes — Raw | Blame | History

```
height = 152 # 탑승을 체크할 키를 대입
if 140 <= height:
    print('롤러코스터 T-Express, 즐기세요!')
```

▲ **키가 140이상이면 놀이 기구 타기 if문 활용 예제 4 - 1 코드**

◆ 조건에 따라 하나를 선택하는 if … else

◆ 논리 표현식 결과인 True와 False에 따라 나뉘는 if … else문

- if문에서 논리 표현식 결과가 **True**이면 **논리 표현식 콜론 이후 블록**을 실행한다.

8 lines (7 sloc) | 267 Bytes — Raw | Blame | History

```
from time import localtime
hour = localtime().tm_hour
mnt = localtime().tm_min

if hour < 10:
    print ('지금 시각 : %d시 %d분, 조조 할인 된다. '%(hour,mnt))
else:
    print ('지금 시각 : %d시 %d분, 조조 할인 안 된다. '%(hour,mnt))
```

- 논리 표현식의 결과가 **False**이면 **else: 이후의 블록**을 실행한다.

▲ **영화 조조 할인 판정 if … else 활용 예제 4 - 3 코드**

◆ 여러 조건 중에서 하나를 선택하는 구문 if … elif

◆ 다중 택일 결정 구조인 if … elif

- if … elif문은 조건의 여러 경로 중에서 하나를 선택하는 구문이다. 논리 표현식 1이 True이면 문장 1, 문장 2의 블록을 실행하고 종료된다. 논리 표현식 1이 False여야 elif논리 표현식 2로 진행되며, 그 이후도 마찬가지다. elif는 필요한 만큼 늘릴 수 있으며, 마지막 else는 선택적으로 생략할 수 있다.

9 lines (9 sloc) | 429 Bytes — Raw | Blame | History

```
1  PM = float(input('미세먼지(10마이크로그램)의 농도는?'))
2  if 151 <= PM:
3      print('미세먼지 농도: {:2f}, 등급: {}'.format(PM, '매우 나쁨'))
4  elif 81 <= PM:
5      print('미세먼지 농도: {:2f}, 등급: {}'.format(PM, '나쁨'))
6  elif 31 <= PM:
7      print('미세먼지 농도: {:2f}, 등급: {}'.format(PM, '보통'))
8  else :
9      print('미세먼지 농도: {:2f}, 등급: {}'.format(PM, '좋음'))
```

▲ **미세먼지 예보 if … elif 활용 예제 4 - 5 코드**

Chapter 04 반복을 제어하는 for문과 while문

◆ 시퀀스의 내부 값으로 반복을 실행하는 for 문

◆ 정해져 있는 시퀀스의 항목 값으로 반복을 실행하는 for문

- **여러 자료 값이 순서대로 구성된 시퀀스에서 자료 값의 개수만큼 반복적인 작업을 수행하기 위한 구문**이 for문이다. 다음 for문에서 시퀀스 이후의 콜론과 반복 몸체인 블록 구조, **실행 흐름**을 이해하자.

1. 여러 개의 값을 갖는 시퀀스에서 변수에 하나의 값을 순서대로 할당 → 2. 할당된 변숫값을 갖고 블록의 문장들을 순차적으로 실행 → 3. 반복 몸체인 문장에선 변수를 사용할 수 있다. → 4. 시퀀스의 그 다음 값을 변수에 할당해 다시 반복 블록을 실행 → 5. 이러한 과정을 시퀀스의 마지막 항목까지 수행 → 6. 시퀀스의 마지막 항목까지 실행한 후 선택 사항인 else: 블록을 실행하고 반복을 종료한다.

6 lines (6 sloc) | 127 Bytes — Raw | Blame | History

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

▲ **for구문으로 수의 나열에서 합과 평균 구하기 활용 예제 4 - 7 코드**

◆ 횟수를 정하지 않은 반복에 적합한 while 반복

◆ 반복 구조가 간단한 while 반복

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

15 lines (14 sloc) | 384 Bytes — Raw | Blame | History

```
1  MAXNUM = 4
2  MAXHEIGHT = 130
3
4  more = True
5  cnt = 0
6  while more:
7      height = float(input("키는? "))
8      if height < MAXHEIGHT:
9          cnt+=1
10         print('들어가세요.','%d명' %cnt)
11     else : print('커서 못 들어갑니다.')
12     if cnt == MAXNUM :
13         more = False
14     else :
15         print('%d명 모두 탔습니다. 다음 번에 이용하세요.'%cnt)
```

▲ **while문을 이용한 어린이를 위한 놀이 기구 탑승 검사 활용 예제 4 - 10 코드**

Chapter 04 임의의 수인 난수와 반복을 제어하는 break, continue문

◆ 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. 여기서는 정수는 시작과 끝을 모두 포함한다.

```
>>> import random
>>> random.randint(1, 3)
3
```

모듈 random에 속한 함수 randint를 호출하는 문장이다.

16 lines (16 sloc) | 398 Bytes — Raw | Blame | History

```
1  winnumber = 11, 17, 28, 30, 33, 35
2  print(' 모의 로또 당첨 번호 '.center(28, '='))
3  print(winnumber)
4  print()
5  print(' 내 번호 확인 '.center(30, '-'))
6  cnt = 0
7  import random
8  for i in range(6):
9      n = random.randint(1, 45)
10     if n in winnumber:
11         print(n, 'O ', end = ' ')
12         cnt += 1
13     else:
14         print(n, 'X ', end = ' ')
15 print()
16 print(cnt, '개 맞음')
```

▲ 난수를 이용해 1에서 45까지의 6개 수를 맞히는 로또 활용 예제 4 - 13 코드

◆ 반복을 제어하는 break문과 continue문

◆ 반복을 종료하는 break문

- 반복 while의 논리 표현식이 True라면 무한히 반복된다. 이를 무한 반복(indefinite loop)이라고 한다.
  for나 while 반복 내부에서 문장 break는 else: 블록을 실행시키지 않고 반복을 무조건 종료한다.

무한 반복

```
while True :
    반복 문장들
```

→

반복을 종료

```
while True :
    . . .
    break
    . . .
```

5 lines (5 sloc) | 110 Bytes — Raw | Blame | History

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

▲ **1을 입력하면 계속하고 0을 입력하면 반복 종료인 break 활용 예제 4 - 14 코드**

◆ continue 이후의 반복 몸체를 실행하지 않고 다음 반복을 계속 실행

- 반복 for와 while문 내부에서 continue문장은 이후의 반복 몸체를 실행하지 않고 다음 반복을 위해 논리 조건을 수행한다.

for에서 continue

```
for 변수 in 시퀀스 :
    . . .
        continue
    . . .
```

while에서 continue

```
while 논리 표현식 :
    . . .
        continue
    . . .
```

11 lines (9 sloc) | 319 Bytes — Raw | Blame | History

```
1  days = ['monday', 'tuesday' 'wednesday']
2
3  while True:
4      user = input('월, 화, 수 중 하나 영어 단어 입력 >> ')
5      if user not in days:
6          print('잘못 입력했어요!')
7          continue
8      print('입력: %s, 철자가 맞습니다.' % user)
9      break
10
11 print(' 종료 '.center(15, '*'))
```

▲ **continue문을 이용한 월, 화, 수 중 영어 철자 하나 검사 활용 예제 4 - 16 코드**

Chapter 05 여러 자료 값을 편리하게 처리하는 리스트

◆ 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(square brackets) [ ] 사이에 항목을 기술한다.

[항목 1, 항목 2, 항목 3, ...] #리스트

```
menu = ['coffee', 'beverage', 'ade']
coffee = ['에스프레소', '아메리카노', '카페라떼', '카페모카']
```

9 lines (7 sloc) 198 Bytes Raw Blame History

```
1  coffee = ['에스프레소', '아메리카노', '카페라테', '카페모카']
2  print(coffee)
3  print(type(coffee))
4
5  num = 0
6  for s in coffee:
7      num += 1
8      print('%d. %s' % (num, s))
9
```

▲ 다양한 커피 종류가 저장된 리스트 활용 예제 5 - 1 코드

◆ 빈 리스트의 생성과 항목 추가

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

```
>>> p1 = [ ] #빈 리스트
>>> print(p1)
[ ]
```

- 인자가 없는 내장 함수 list( )로도 빈 리스트를 생성할 수 있다. 리스트의 가장 뒤에 항목을 추가하려면 리스트의 메소드 append(삽입할 항목)를 사용한다.

```
>>> p1 = list( )
>>> p1.append('C++')
>>> p1.append('java')
>>> print(p1)
['C++', 'java']
```

6 lines (6 sloc) | 152 Bytes — Raw | Blame | History

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

▲ **append( )를 이용해 리스트로 편의점에서 구입할 품목 만들기 활용 예제 5 - 3 코드**

◆ 리스트의 첨자로 항목 수정

- 리스트의 첨자를 이용한 항목을 대입 연산자의 오른쪽에 위치시켜 리스트의 항목을 수정할 수 있다. 첨자는 유효한 것이어야 하므로 적어도 하나 이상의 항목이 있는 경우에 수정할 수 있다.

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

12 lines (11 sloc) | 307 Bytes — Raw | Blame | History

```
food = ['짜장면', '짬뽕', '우동', '울면']
print(food)
# 탕수육 주문 추가
food.append('탕수육')
print(food)
# 짬뽕을 굴짬뽕으로 주문 변경
food[1] = '굴짬뽕'
print(food)

# 우동을 물만두로 주문 변경
food[food.index('우동')] = '물만두'
print(food)
```

▲ **리스트 추가 삭제로 중국집에서 음식 주문하기 활용 예제 5 - 6 코드**

Chapter 05 리스트의 부분 참조와 항목의 삽입과 삭제

◆ 첨자 3개로 리스트 일부분을 참조하는 슬라이싱

리스트[start : stop : step]

: 첨자 start에서 첨자 stop-1까지 step 간격으로 부분 리스트 반환

- 0에서 시작하는 오름차순(수가 차례로 늘어나는 것).
- 마지막 요소 –1에서 시작하는 내림차순(수가 차례로 줄어가는 것)의 첨자도 가능하며, 다소 복잡하지만 두 순차의 첨자를 함께 사용 가능

14 lines (14 sloc) | 393 Bytes — Raw | Blame | History

```
1  wlist = ['밥', '삶', '길', '죽', '꿈', '차', '떡', '복', '말']
2  print('wlist[:] = ', wlist[:])
3  print('wlist[::] = ', wlist[::])
4  print('wlist[::-1] = ', wlist[::-1])
5  # 오름차순
6  print(wlist[::3])
7  print(wlist[1::3])
8  print(wlist[2::3])
9  # 내림차순
10 print(wlist[::-2])
11 print(wlist[-1:-8:-3])
12 # 오름과 내림차순의 혼재
13 print(wlist[1:-1:])
14 print(wlist[-2:-9:-3])
```

▲ **한 글자의 한국 단어로 이해하는 리스트 슬라이싱 활용 예제 5 - 8 코드**

◆ 리스트의 추가, 연결과 반복

◆ 리스트에 리스트를 추가하는 메소드 extend( )

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

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

◆ 리스트를 연결하는 +, * 연산자

- 더하기 연산자 +는 리스트와 리스트를 연결해주며, 리스트와 정수와 *로 곱하면 항목이 지정된 정수만큼 반복된 리스트를 반환한다.

```
>>> korean = ['불고기','설렁탕']
>>> chinese = ['탕수육', '기스면']
>>> food = korean + chinese
>>> print(food)
['불고기', '설렁탕', '탕수육', '기스면']
```

```
>>> days = ['월', '화']
>>> print(days * 3)
['월', '화', '월', '화', '월', '화']
```

Chapter 05 항목의 순서나 내용을 수정할 수 없는 튜플

◆ 괄호로 정의하는 시퀀스 튜플

◆ 수정할 수 없는 항목의 나열인 튜플

- 튜플은 문자열, 리스트와 같은 항목의 나열인 시퀀스다. 튜플은 리스트와 달리 **항목의 순서나 내용의 수정이 불가능**하다. 튜플도 모두 콤마로 구분된 항목들의 **리스트로 표현**되며, 각각의 항목은 정수, 실수, 문자열, 리스트, 튜플 등 **제한이 없다**.
  - 튜플은 괄호 (…) 사이에 항목을 기술한다. 괄호는 생략할 수 있다

(항목 1, 항목 2, 항목 3, …) #튜플

13 lines (10 sloc) | 389 Bytes — Raw | Blame | History

```
1  singer = ('BTS', '봄빨간사춘기', 'BTS', '블랙핑크', '태연')
2  song = ('작은 것들을 위한 시', '나만, 봄', '소우주', 'Kill This Love', '사계')
3  print(singer)
4  print(song)
5
6  print(singer.count('BTS'))
7  print(singer.index('봄빨간사춘기'))
8  print(singer.index('BTS'))
9  print()
10
11 for _ in range(len(singer)):
12     print('%s: %s' % (singer[_], song[_]))
13
```

▲ **K-pop 가수와 곡으로 구성된 튜플의 참조 활용 예제 5 - 12 코드**

Chapter 06 키와 값인 쌍의 나열인 딕셔너리

◆ 딕셔너리의 개념과 생성

◆ 키와 값의 쌍을 항목으로 관리하는 딕셔너리

- 딕셔너리는 키와 값의 쌍인 항목을 나열한 시퀀스다. 즉, 딕셔너리는 콤마로 구분된 항목(또는 요소, 원소)들의 리스트로 표현된다. 각각의 항목은 키:값과 같이 키와 값을 콜론으로 구분하고 전체는 **중괄호 { … }**로 묶는다.
  - 딕셔너리는 중괄호 { … } 사이에 키와 값의 항목을 기술하며 값은 키로 참조된다..
  - 딕셔너리의 항목 순서는 의미가 없으며, 키는 중복될 수 없다.
  - 키는 수정될 수 없지만, 값은 수정될 수 있다.

dct = {<key>: <value>,<key>: <value>, … ,<key>: <value>} #딕셔너리

```
>>> groupnumber = {'엑소': 9, '트와이스': 9, '블랙핑크': 4, '방탄소년단': 7}
>>> coffeeprice= {'에스프레소', 2500, '아메리카노': 2800, '카페라떼': 3200}
>>> mycar = {"brand": "현대", "model": "제네시스", "year": 2016}
```

3 lines (3 sloc) | 138 Bytes Raw | Blame | History

```
1  groupnumber = {'잔나비': 5, '트와이스': 9, '블랙핑크': 4, '방탄소년단': 7}
2  print(groupnumber)
3  print(type(groupnumber))
```

▲ **K-pop그룹의 인원수 활용 예제 6 - 1 코드**

◆ 다양한 인자의 함수 dict( )로 생성하는 딕셔너리

◆ 리스트 또는 튜플로 구선된 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')])
>>> 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' }
```

15 lines (12 sloc) | 627 Bytes | Raw | Blame | History

```
1  bts1 = {'그룹명': '방탄소년단', '인원수': 7, '리더': '김남준'}
2  bts1['소속사'] = '빅히트 엔터테인먼트';
3  print(bts1)
4  bts2 = dict([['그룹명', '방탄소년단'], ['인원수', 7]])
5  print(bts2)
6  bts3 = dict((('리더', '김남준'), ('소속사', '빅히트 엔터테인먼트')))
7  print(bts3)
8
9  bts = dict(그룹명 = '방탄소년단', 인원수=7, 리더='김남준', 소속사='빅히트 엔터테인먼트')
10 # 구성원 추가
11 bts['구성원'] = ['RM', '진', '슈가', '제이홉', '지민', '뷔', '정국']
12
13 print(bts) # 전체 출력
14 print(bts['구성원']) # 구성원 출력
```

▲ **방탄소년단 정보를 저장하는 다양한 딕셔너리 생성과 참조 활용 예제 6 - 3 코드**

Chapter 06 중복과 순서가 없는 집합

◆ 수학에서 배운 집합을 처리하는 자료형

◆ 원소는 유일하고 순서는 의미 없는 집합

- 집합은 중복되는 요소가 없으며, 순서도 없는 원소의 모임(collections)이다. 파이썬에서 집합은 수학과 같이 원소를 콤마로 구분하며 **중괄호**로 둘러싸 표현한다.

{ 원소 1, 원소 2, ... 원소 n }

◆ 내장 함수 set( )을 활용한 집합 생성

◆ 집합을 만드는 내장 함수 set( )

set (원소로 구성된 리스트_or_튜플_or_문자열)

- 인자가 없으면 빈 집합인 공집합이 생성된다.
- 인자가 있으면 하나이며, 리스트와 튜플, 문자열 등이 올 수 있다.

◆ 함수 set( )으로 공집합 만들기

- 내장 함수 set( )으로 만들어지는 다음 집합 a는 공집합이며, 집합의 **자료형 이름은 클래스 set**이다.

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

◆ 리스트나 튜플을 인자로 사용하는 함수 set( )

- 함수 set( )에서는 인자로 리스트 또는 튜플 자체를 사용할 수 있으며, 결과는 시퀀스 항목에서 **중복을 제거**한 원소로 구성된다. 생성된 집합은 수학에서 배운 것과 같이 중괄호 안에 원소가 콤마로 구분되어 표시된다.

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

◆ 문자열을 인자로 사용하는 함수 set( )

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

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

◆ 수학에서 배운 집합을 처리하는 자료형

◆ 집합 메소드 add(원소)로 추가

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

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

- 원소의 삭제는 메소드 remove(원소)를 사용한다. 메소드 remove(원소)호출에서 삭제하려는 원**소가 없으면 KeyError가 발생**하므로 주의하자.

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

- 메소드 discard(원소)로도 원소를 삭제할 수 있으며, **원소가 없어도 오류가 발생하지 않는다**.

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

- **임의의 원소를 삭제**하려면 pop( )을 사용해야 한다.

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

◆ 메소드 clear( )로 집합의 모든 원소 삭제

- **집합의 모든 원소를 삭제**하려면 메소드 clear( )를 사용해야 한다.

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

Ⅲ. 학습 내용 작성 후

◆ 포트폴리오 소감

- 개인적으로 2학년에 들어서 제일 기대가 됐던 과목이 파이썬이었다.

  종강 때 무료로 진행했던 파이썬 강의를 듣지 못해 혼자 유튜브로 독학을 했었다.
  확실히 다른 전공 과목 코딩들에 비해 파이썬의 장점대로 배우기 쉽고 코딩들이 간결해
  즐겁고 쉽게 배울 수 있었던 것 같다.
  이번에 작성한 파이썬 포트폴리오를 통해 각 단원에 대해 한번 더 복습했던 계기가 되었고 ,
  단계별로 어떤식으로 정리를 해야 하는지 나만의 방식이 생긴 것 같다.
  포트폴리어 작성으로 시간이 꽤 많이 걸렸지만, 그동안의 내가 놓친 부분과 미약했던 부분을
  다시 복습했던 계기가 되어 완성하고 보니 매우 뿌듯했다.

- 강의가 매주 진행될수록 단원별 어려운 부분이 늘어나는 것 같다.

  매주 차 배우는 강의 진행 속도보다 늦춰지지 않게 열심히 복습하며 놓치는 부분이 있더라도
  절대 포기하지 않고 열심히 공부해야겠다.

- **분량이 꽤 많지만 끝까지 봐주셔서 감사드립니다.**