AI Programming

Python

01. Python Basic 1

first coding

파이썬 Overview

Python이란?

파이썬 특징

Python으로 무엇을 할 수 있나?

파이썬이란?

- 파이썬(Python)

  - 1990년 암스테르담의 귀도 반 로섬(Guido Van Rossum)이 개발한 인터프리터 언어
  - 파이썬의 사전적 의미
    - 고대 신화에 나오는 파르나소스 산의 동굴에 살던 큰 뱀
  - 구글에서 만든 소프트웨어의 50% 이상이 파이썬으로 작성됨
  - 인스타그램(Instagram), 넷플릭스(Netflix), 아마존(Amazon) 등에서 사용
  - 공동 작업과 유지 보수가 매우 쉽고 편리함

파이썬의 특징

- 파이썬은 인간다운 언어이다
  - 파이썬은 사람이 생각하는 방식을 그대로 표현할 수 있는 언어

```
If 3 in [1, 2, 3, 4]:  print('3이 있습니다.')
```

=

만약 3이 1, 2, 3, 4 중에 있으면 '3이 있습니다'를 출력한다.

파이썬의 특징

- **문법이 쉬워 상대적으로 빠르게 배울 수 있다**
  - 문법 자체가 아주 쉽고 간결하며 사람의 사고 체계와 매우 닮아 있음
  - 유명한 프로그래머인 에릭 레이먼드(Eric Raymond)는 공부한 지 단 하루 만에 자신이 원하는 프로그램을 작성!

- **무료이지만 강력하다**
  - 오픈 소스 → 무료로 언제 어디서든 파이썬을 다운로드하여 사용 가능
  - 파이썬과 C는 찰떡 궁합
    - 프로그램의 전반적인 뼈대는 파이썬으로 만들고, 빠른 실행 속도가 필요한 부분은 C로 만들어서 파이썬 프로그램 안에 포함

파이썬의 특징

- 간결하다
  - 프로그램이 정상적으로 실행되게 하려면 줄(들여쓰기)을 반드시 맞추어야 함
    - 가독성 ↑

```
languages = ['python', 'perl', 'c', 'java']

for lang in languages:
    if lang in ['python', 'perl']:
        print("%6s need interpreter" % lang)
    elif lang in ['c', 'java']:
        print("%6s need compiler" % lang)
    else:
        print("should not reach here")
```

파이썬의 특징

- **파이썬으로 할 수 있는 일**

  - **웹 프로그래밍**
    - 파이썬은 웹 프로그램을 만들기에 매우 적합한 도구
    - 파이썬으로 제작한 웹 사이트는 셀 수 없이 많을 정도

  - **인공지능과 머신러닝**
    - 자연어 처리, 음성 인식, 이미지 인식과 같은 인공지능 기술 구현
    - 인공지능과 머신러닝 프로그래밍을 쉽게 할 수 있도록
      scikit-learn, TensorFlow, PyTorch, Keras 등과 같은 다양한 라이브러리 제공

  - **수치 연산 프로그래밍**
    - C로 작성된 넘파이(Numpy) 수치 연산 모듈을 통해 빠른 수치 연산 가능

파이썬으로 무엇을 할 수 있을까?

- 파이썬으로 할 수 있는 일

  - 데이터 분석

    - NumPy, Pandas, Matplotlib 등의 라이브러리를 활용해
      데이터 처리, 통계 분석, 시각화를 손쉽게 수행
    - 판다스가 등장한 이래 데이터 분석에 R보다 파이썬을 사용하는 사례 증가

  - 데이터베이스 프로그래밍

    - Sybase, Infomix, Oracle, MySQL, PostgreSQL 등의 데이터베이스에 접근하기 위한 도구 제공
    - 자료를 변형 없이 그대로 파일에 저장하고 불러오는 파이썬 모듈 피클(pickle)

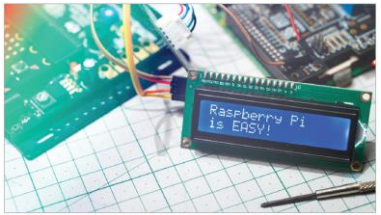
  - 시스템 유틸리티 제작하기

    - 운영체제(윈도우, 리눅스 등)의 시스템 명령어를 사용하는 도구를 통한 시스템 유틸리티 제작

파이썬으로 무엇을 할 수 있을까?

- **파이썬으로 할 수 있는 일**

  - **GUI(Graphic User Interface) 프로그래밍**
    - 화면에 윈도우 창을 만들고 프로그램을 동작시킬 수 있는 메뉴나 버튼, 그림 등을 추가하는 것
    - GUI 프로그래밍을 위한 도구들을 갖추고 있어, GUI 프로그램을 만들기 쉬움 (예 - Tkinter(티케이인터))
  - **C/C++와 결합하기**
    - C나 C++로 만든 프로그램을 파이썬에서, 파이썬으로 만든 프로그램을 C나 C++에서 사용 가능
  - **사물 인터넷**
    - 라즈베리파이를 제어하며 사물 인터넷 구현



파이썬으로 무엇을 할 수 있을까?

- 파이썬으로 할 수 없는 일

  - 시스템과 밀접한 프로그래밍 영역
    - 운영체제, 엄청난 횟수의 반복과 연산이 필요한 프로그램 등
      매우 빠른 속도를 요구하거나 하드웨어를 직접 건드려야 하는 프로그램에는 적합하지 않음

  - 모바일 프로그래밍
    - 안드로이드 네이티브 앱(android native app) 개발에는 아직 역부족
    - 아이폰 앱 개발은 할 수 없음

first coding

파이썬 개발환경

Python 개발 환경

파이썬 설치하기

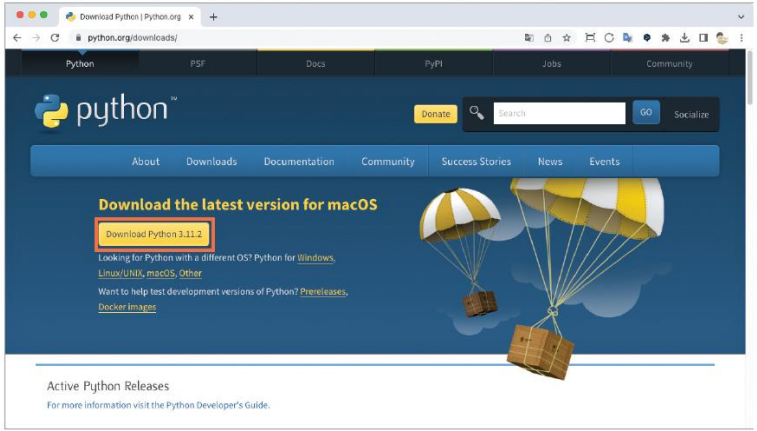
- 윈도우에서 파이썬 설치하기

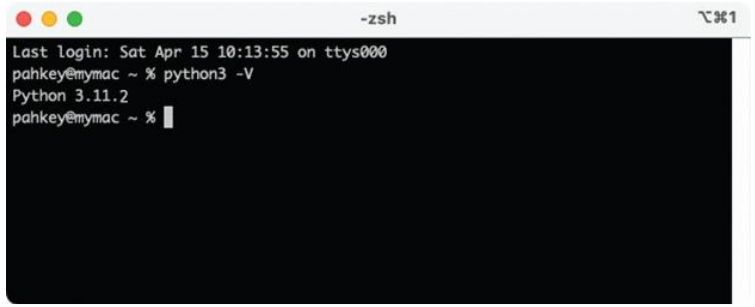
1. 파이썬 공식 홈페이지(www.python.org/downloads)에서 Python 3.x 최신 버전 다운로드
2. 파이썬이 어느 곳에서든지 실행될 수 있도록 'Add python.exe to PATH' 옵션 선택

파이썬 설치하기

- 맥에서 파이썬 설치하기

1. 파이썬 공식 홈페이지(www.python.org)의 [Downloads] 메뉴에서 맥용 설치 파일 다운로드
2. 내려 받은 파일을 실행하여 설치
3. 터미널에서 명령을 입력해 자신의 맥에 설치된 파이썬 버전 확인



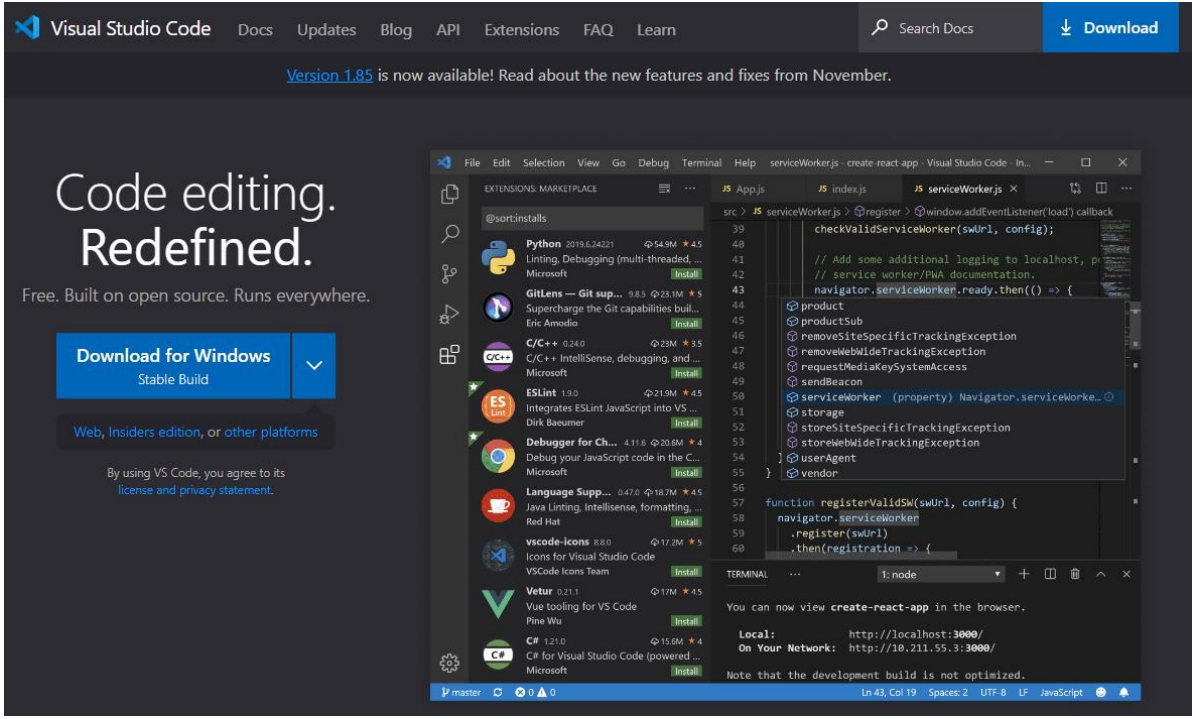


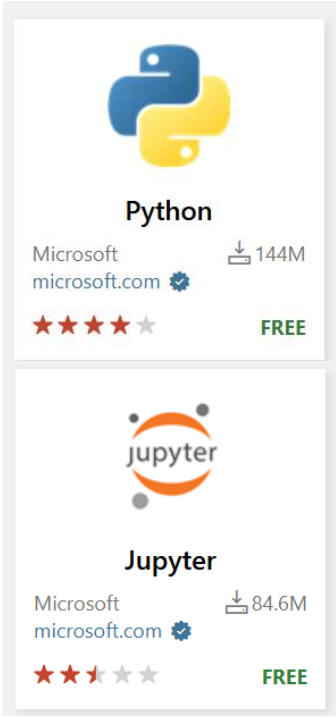
파이썬과 에디터

- 에디터
  - JupiterNotebook
  - 비주얼 스튜디오 코드 (code.visualstudio.com)
  - 파이참 (www.jetbrains.com/pycharm/download)

VSCODE 설치 와 환경 설정

- VSCode 설치
  - https://code.visualstudio.com/





first coding

변수와 자료형

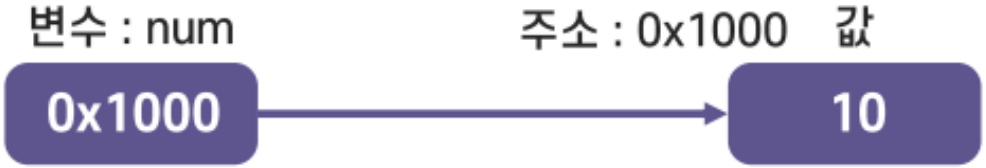
Python 개발 환경

자료형의 값을 저장하는 공간, 변수

- 변수란?
  - 파이썬에서 사용하는 변수는 객체를 가리키는 것

```
num = 10
```

→ 10 데이터를 메모리공간에 저장하고,
100이 저장된 메모리공간의 주소를
num이라고 하는 메모리 공간에 저장한다.



자료형의 값을 저장하는 공간, 변수

- **변수란?**

  - 파이썬에서 사용하는 변수는 객체를 가리키는 것

    우리가 사용하는 변수는 메모리의 주소 값을 가지고 있다!

    - **객체(저장 된 데이터) = 자료형**

    - [1, 2, 3, 4, 5] 값을 가지는 리스트 데이터(객체)가 자동으로 메모리에 생성됨
    - 변수 a는 [1, 2, 3, 4, 5] 리스트가 저장된 메모리의 주소를 가리킴
      - id( ) 함수를 사용하여 메모리 주소 확인

자료형의 값을 저장하는 공간, 변수

- 변수
  - 변수의 예

```
>>> a = 1
>>> b = "python"
>>> c = [1, 2, 3]
```

  - 변수 선언 방법
    - =(assignment) 기호를 사용

변수 이름 = 변수에 저장할 값(데이터)

변수의 명명 규칙

- 문자(A-z)와 숫자(0-9), _(underscore)를 사용할 수 있음
  - data = 0, _a1 = 2, _name = 'king'
- 변수명은 대소문자가 구분됨
  - Num과 num은 다른 변수임
- 변수명은 의미 있는 단어로 사용하는 것이 좋음
  - age, student_name, student_number
- 변수명이 숫자로 시작하면 할 수 없음
  - 1num
- 변수명에는 공백이 포함될 수 없음
  - student name
- 예약어를 변수명으로 사용할 수 없음
  - for, if, True 등

자료형의 값을 저장하는 공간, 변수

- 리스트의 복사

  - b 변수에 a 변수를 대입하면 b가 가지는 값은?

    - Id( ) 함수로 확인
    - 파이썬 명령어로 확인
    - 다른 방법으로 확인

```
>>> a = [1, 2, 3]
>>> b = a
```

```
>>> id(a)
4303029896
>>> id(b)
4303029896
```

```
>>> a is b
True
```

```
>>> a[1] = 4
>>> a
[1, 4, 3]
>>> b
[1, 4, 3]
```

자료형의 값을 저장하는 공간, 변수

- 변수를 만드는 여러 가지 방법
  - 튜플

```
>>> (a, b) = 'python', 'life'
```

  - 괄호 생략 가능

```
>>> a, b = ('python', 'life')
```

- 여러 개 변수에 같은 값 대입하기

```
>>> a = b = 'python'
```

- 파이썬에서 두 변수 값 바꾸기

```
>>> a = 3
>>> b = 5
>>> a, b = b, a  <-- a와 b의 값을 바꿈.
>>> a
5
>>> b
3
```

first coding

프로그래밍의 기초, 자료형

숫자형

문자형

bool

- 자료형

자료형

프로그램에서 나타낼 수 있는 데이터의 종류

- 할당 받는 메모리 공간의 크기는 **변수의 자료형(Data Type)**에 의해 결정됨
  → 변수에 저장될 자료의 형에 따라 구분
- 숫자형 : 정수(int), 실수(float)
- 문자형 : 문자열(str)
- 논리형 : 불린(boolean)
- 컬렉션형 : 리스트(list), 튜플(tuple), 집합(set), 딕셔너리(dictionary)
- None

02 동적 타이핑

- 변수의 메모리 공간을 생성하는 시점이 프로그램이 실행되는 시점에 생성되는 것을 의미

예 컴파일 언어의 경우

- int num=10과 같이 **실행 이전에 변수의 타입을 정의**

예 파이썬의 경우

- num=8과 같이 선언함

- 프로그램이 실행되는 시점인 8의 값이 저장될 때 인터프리터가 정수(int) 임을 판단해서 메모리의 타입을 결정함

✔ 파이썬은 프로그램의 실행 시점에 동적으로 판단해서 적용되는 것을 동적으로 자료형을 결정함

숫자형

- 숫자형(number)

  – 숫자 형태로 이루어진 자료형

항목	파이썬 사용 예
정수	123, -345, 0
실수	123.45, -1234.5, 3.4e10
8진수	0o34, 0o25
16진수	0x2A, 0xFF

숫자형

- **정수형(integer)**
  - 정수를 뜻하는 자료형
  - 8진수(octal)
    - 숫자 0 + 알파벳 소문자 o 또는 대문자 O
  - 16진수(hexadecimal)
    - 숫자 0 + 알파벳 소문자 x

```
>>> a = 123   <-- 양의 정수 대입
>>> a = -178  <-- 음의 정수 대입
>>> a = 0     <-- 숫자 0 대입
```

```
>>> a = 0o177
>>> print(a)
127  <-- 1×8^2+7×8+7=127
```

```
>>> a = 0x8ff
>>> b = 0xABC
>>> print(b)
2748  <-- 10×16^2+11×16+12=2748
```

- **실수형(floating-point)**
  - 소수점이 포함된 숫자

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

```
>>> a = 4.24E10   <-- 4.24×10^10
>>> a = 4.24e-10  <-- 4.24×10^-10
```

숫자형

- 숫자형을 활용하기 위한 연산자

  - 사칙 연산 (+, -, *, /)
  - x의 y제곱을 나타내는 **** 연산자**
  - 나눗셈 후 나머지를 반환하는 **% 연산자**
  - 나눗셈 후 몫을 반환하는 **// 연산자**

```
>>> a = 3
>>> b = 4
>>> a + b
7
>>> a - b
-1
>>> a * b
12
>>> a / b
0.75
```

```
>>> 7 % 3
1
>>> 3 % 7
3
```

```
>>> a = 3
>>> b = 4
>>> a ** b
81
```

```
>>> 7 // 4
1
```

문자열 자료형

- 문자열(string)

  - 문자, 단어 등으로 구성된 문자들의 집합, 시퀀스 라고도 함.

```
“hello~ Python!!”
“A”
“100”
```

문자열 자료형

- 문자열 생성

1. 큰따옴표(“)
2. 작은따옴표(‘)
3. 큰따옴표 3개(“””)
4. 작은따옴표 3개(‘’’)

```
"Hello World"
```

```
'Python is fun'
```

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

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

문자열 자료형

- **문자열 안에 작은따옴표나 큰따옴표를 포함**

1. 작은따옴표(')
    - 큰따옴표(")로 둘러싸기

```
>>> food = "Python's favorite food is perl"
```

2. 큰따옴표(")
    - 작은따옴표(')로 둘러싸기

```
>>> say = '"Python is very easy." he says.'
```

3. 역슬래시 사용하기
    - 역슬래시(₩) 뒤의 작은따옴표(')나 큰따옴표(")는 문자열을 둘러싸는 기호의 의미가 아니라 문자 ('), (") 그 자체를 의미

```
>>> food = 'Python\'s favorite food is perl'
>>> say = "\"Python is very easy.\" he says."
```

문자열 자료형

- 개행이 있는 문자열을 변수에 대입

1. 이스케이프 코드 '₩n' 삽입

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

2. 작은따옴표 3개(''')

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

작은따옴표 3개를 사용한 경우

3. 큰따옴표 3개(""")

```
>>> multiline = """
... Life is too short
... You need python
... """
```

큰따옴표 3개를 사용한 경우

문자열 자료형

- 문자열 연산하기

1. 문자열 더해서 연결하기

```
>>> head = "Python"
>>> tail = " is fun!"
>>> head + tail
'Python is fun!'
```

2. 문자열 곱하기

```
>>> a = "python"
>>> a * 2
'pythonpython'
```

3. 문자열 길이 구하기
   - 파이썬 기본 내장 함수 len() 사용

```
>>> a = "Life is too short"
>>> len(a)
17
```

문자열 자료형

- 문자열 인덱싱과 슬라이싱

L	i	f	e		i	s		t	o	o		s	h	o	r	t	,		Y	o	u		n	e	e	d		P	y	t	h	o	n
0	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	32	33

  - 문자열 인덱싱(indexing)
    - 0 index : 0 ~ N-1

```
>>> a = "Life is too short, You need Python"
>>> a[0]
'L'
>>> a[12]
's'
>>> a[-1]
'n'
```

    - a[index]
      - 문자열 안의 특정 값을 뽑아 냄
      - 마이너스(-)
        - » 문자열 마지막 index 뒤부터 시작

```
a[0]: 'L', a[1]: 'i', a[2]: 'f', a[3]: 'e', a[4]: ' ', ···
```

문자열 자료형

- 문자열 인덱싱과 슬라이싱

  - 문자열 슬라이싱(slicing)
    - 각 요소 문자를 특정 위치에서부터 잘라 냄

    - a[시작 index : 끝 index]
      - 시작 번호부터 끝 번호까지의 문자를 뽑아 냄
      - 끝 번호에 해당하는 것은 포함하지 않음 → 탈출 위치

L	i	f	e		i	s		t	o	o		s	h	o	r	t	,		Y	o	u		n	e	e	d		P	y	t	h	o	n
0	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	32	33

```
>>> a = "Life is too short, You need Python"
>>> a[0:4]
'Life'
```

```
>>> a = "20230331Rainy"
>>> date = a[:8]
>>> weather = a[8:]
>>> date
'20230331'
>>> weather
'Rainy'
```

문자열 자료형

- **문자열 포매팅**

  - **문자열 포매팅**(formatting)
    - 특정 문자열 형식으로 문자열 형식 정의 → 문자열의 특정 위치에 값을 삽입

1. 숫자 바로 대입
   - 문자열 포맷 코드 : 정수 → %d
2. 문자열 포맷 코드 %s
   - 문자열 바로 대입
3. 숫자 값을 나타내는 변수로 대입
4. 2개 이상의 값 넣기

```
>>> "I eat %d apples." % 3
'I eat 3 apples.'
```

```
>>> "I eat %s apples." % "five"
'I eat five apples.'
```

```
>>> number = 3
>>> "I eat %d apples." % number
'I eat 3 apples.'
```

```
>>> number = 10
>>> day = "three"
>>> "I ate %d apples. so I was sick for %s days." % (number, day)
'I ate 10 apples. so I was sick for three days.'
```

문자열 자료형

- 문자열 포맷 코드
  - 문자열 포맷 코드의 종류

코드	설명
%s	문자열(string)
%c	문자 1개(character)
%d	정수(integer)
%f	부동소수(floating-point)
%o	8진수
%x	16진수
%%	Literal %(문자 % 자체)

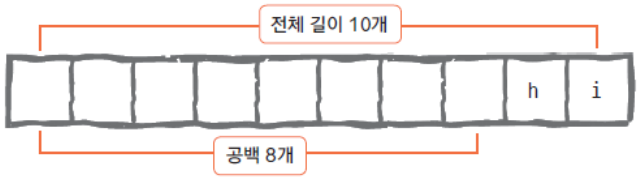
```
>>> "I have %s apples" % 3
'I have 3 apples'
>>> "rate is %s" % 3.234
'rate is 3.234'
```

문자열 자료형

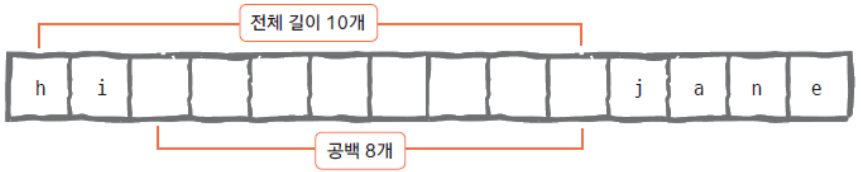
- **포맷 코드와 숫자 함께 사용하기**

1. 정렬과 공백
    - %s를 숫자와 함께 사용하면, 공백과 정렬 표현 가능

```
>>> "%10s" % "hi"
'        hi'      <— hi가 오른쪽 정렬됨.
```




```
>>> "%-10sjane." % 'hi'
'hi        jane.'  <— hi가 왼쪽 정렬됨.
```



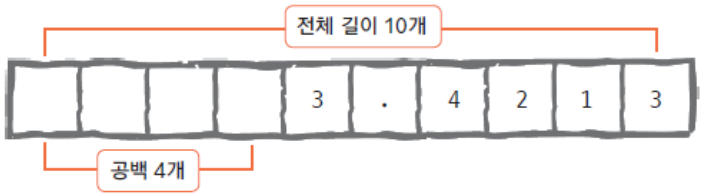
문자열 자료형

- **포맷 코드와 숫자 함께 사용하기**

2. 소수점 표현하기
    - %f를 숫자와 함께 사용하면, 소수점 뒤에 나올 숫자의 개수 조절 및 정렬 가능

```
>>> "%0.4f" % 3.42134234
'3.4213'
```

```
>>> "%10.4f" % 3.42134234
'    3.4213'
```



문자열 자료형

- format 함수를 사용한 포매팅
  - 숫자 바로 대입하기
  - 숫자 값을 가진 변수로 대입하기
  - 문자열 바로 대입하기

```
>>> "I eat {0} apples".format(3)
'I eat 3 apples'
```

```
>>> number = 3
>>> "I eat {0} apples".format(number)
'I eat 3 apples'
```

```
>>> "I eat {0} apples".format("five")
'I eat five apples'
```

문자열 자료형

- format 함수를 사용한 포매팅
  - 2개 이상의 값 넣기
  - 이름으로 넣기

```
>>> number = 10
>>> day = "three"
>>> "I ate {0} apples. so I was sick for {1} days.".format(number, day)
'I ate 10 apples. so I was sick for three days.'
```

```
>>> "I ate {number} apples. so I was sick for {day} days.".format(number=10, day=3)
'I ate 10 apples. so I was sick for 3 days.'
```

문자열 자료형

- format 함수를 사용한 포매팅
  - 인덱스와 이름을 혼용해서 넣기
  - 왼쪽 정렬
  - 오른쪽 정렬
  - 가운데 정렬

```
>>> "I ate {0} apples. so I was sick for {day} days.".format(10, day=3)
'I ate 10 apples. so I was sick for 3 days.'
```

```
>>> "{0:<10}".format("hi")
'hi        '
```

```
>>> "{0:>10}".format("hi")
'        hi'
```

```
>>> "{0:^10}".format("hi")
'    hi    '
```

문자열 자료형

- format 함수를 사용한 포매팅
  - 공백 채우기
  - 소수점 표현하기
  - { 또는 } 문자 표현하기

```
>>> "{0:=^10}".format("hi")
'====hi===='
>>> "{0:!<10}".format("hi")
'hi!!!!!!!!'
```

```
>>> y = 3.42134234
>>> "{0:0.4f}".format(y)
'3.4213'
```

```
>>> "{0:10.4f}".format(y)
'    3.4213'
```

```
>>> "{{ and }}".format()
'{ and }'
```

문자열 자료형

- f 문자열 포매팅
  - 파이썬 3.6 버전부터 f 문자열 포매팅 기능 제공
  - 문자열 앞에 f 접두사를 붙이면 f 문자열 포매팅 기능 사용 가능

```
>>> name = '홍길동'
>>> age = 30
>>> f'나의 이름은 {name}입니다. 나이는 {age}입니다.'
'나의 이름은 홍길동입니다. 나이는 30입니다.'
```

```
>>> age = 30
>>> f'나는 내년이면 {age + 1}살이 된다.'
'나는 내년이면 31살이 된다.'
```

문자열 자료형

- f 문자열 포매팅
  - 딕셔너리에서 사용
  - 정렬
  - {} 문자 그대로 표시

```
>>> d = {'name':'홍길동', 'age':30}
>>> f'나의 이름은 {d["name"]}입니다. 나이는 {d["age"]}입니다.'
'나의 이름은 홍길동입니다. 나이는 30입니다.'
```

```
>>> f'{"hi":<10}' <- 왼쪽 정렬
'hi        '
>>> f'{"hi":>10}' <- 오른쪽 정렬
'        hi'
>>> f'{"hi":^10}' <- 가운데 정렬
'    hi    '
```

```
>>> f'{{ and }}'
'{ and }'
```

문자열 자료형

- f 문자열 포매팅
  - 공백 채우기
  - 소수점 표현

```
>>> f'{"hi":=^10}'  <- 가운데 정렬하고 '='로 공백 채우기
'====hi===='
>>> f'{"hi":!<10}'  <- 왼쪽 정렬하고 '!'로 공백 채우기
'hi!!!!!!!!'
```

```
>>> y = 3.42134234
>>> f'{y:0.4f}'      <- 소수점 4자리까지만 표현
'3.4213'
>>> f'{y:10.4f}'     <- 소수점 4자리까지 표현하고 총 자릿수를 '10'으로 맞춤.
'    3.4213'
```

문자열 자료형

- **문자열 관련 함수들**

  - 문자열 자료형이 가진 내장 함수
    - 문자 개수 세기 - count
    - 문자열 삽입 – join
    - 위치 알려 주기 1 – find
      - 찾는 문자열이 처음 나온 위치 반환
      - 없으면 -1 반환
    - 위치 알려 주기 2 – index
      - find와 마찬가지로,
        찾는 문자열이 처음 나온 위치 반환
      - 단, 찾는 문자열이 없으면 오류 발생

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

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

```
>>> a = "Python is the best choice"
>>> a.find('b')
14    <-- 문자열에서 b가 처음 나온 위치
>>> a.find('k')
-1
```

```
>>> 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
```

k가 없으므로 오류 발생

문자열 자료형

- 문자열 관련 함수들

  - 문자열 자료형이 가진 내장 함수
    - 소문자를 대문자로 바꾸기 – upper
    - 왼쪽 공백 지우기 – lstrip
    - 대문자를 소문자로 바꾸기 – lower
    - 오른쪽 공백 지우기 - rstrip

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

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

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

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

문자열 자료형

- **문자열 관련 함수들**

  - 문자열 자료형이 가진 내장 함수
    - 양쪽 공백 지우기 – **strip**
    - 문자열 바꾸기 – **replace**
      - replace(바뀔_문자열, 바꿀_문자열)
    - 문자열 나누기 – **split**
      - 공백 또는 특정 문자열을 구분자로 해서 문자열 분리
      - 분리된 문자열은 리스트로 반환됨

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

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

```
>>> a = "Life is too short"
>>> a.split()     <-- 공백을 기준으로 문자열 나눔.
['Life', 'is', 'too', 'short']
>>> b = "a:b:c:d"
>>> b.split(':') <-- :를 기준으로 문자열 나눔.
['a', 'b', 'c', 'd']
```

불 자료형

- 불 자료형(bool)이란?
  - 참(True)과 거짓(False)을 나타내는 자료형
    - type( ) 함수를 사용하여 자료형 확인
    - 조건문의 반환 값으로도 사용됨

```
>>> a = True
>>> b = False
```

```
>>> type(a)
<class 'bool'>
>>> type(b)
<class 'bool'>
```

```
>>> 1 == 1
True
```

```
>>> 2 > 1
True
```

```
>>> 2 < 1
False
```

불 자료형

- 자료형의 참과 거짓

값	참 또는 거짓
"python"	참
""	거짓
[1, 2, 3]	참
[]	거짓
(1, 2, 3)	참
()	거짓
{'a': 1}	참
{}	거짓
1	참
0	거짓
None	거짓

```
>>> if []:    <- 만약 []가 참이면
...     print("참")    <- '참' 문자열 출력
... else:    <- 만약 []가 거짓이면
...     print("거짓") <- '거짓' 문자열 출력
...
거짓
```

불 자료형

- **불 연산**
  - bool( ) 함수
    - 자료형의 참/거짓을 식별하는 내장 함수
    - 문자열의 참/거짓
      - 'python' 문자열은 빈 문자열이 아니므로 True 리턴
      - ' ' 문자열은 빈 문자열이므로 False 반환
      - 리스트, 숫자의 참/거짓

```
>>> bool('python')
True
```

```
>>> bool('')
False
```

```
>>> bool([1, 2, 3])
True
>>> bool([])
False
>>> bool(0)
False
>>> bool(3)
True
```

연산자

- 연산자

04 산술연산자

연산기호	설명	사용 예(x=10, y=20, z=3)
+	더하기	x + y → 30
-	빼기	x – y → -10
*	곱하기	x * y → 200
/	나누기	y / x → 2.0
%	나머지	y % x → 0
**	제곱	x ** z → 1000
//	몫	x // z → 3

연산자

- 연산자

05 관계연산자

연산기호	설명	사용 예(x=10, y=20)
==	값이 같다	x == y → False
!=	값이 같지 않다	x != y → True
>	왼쪽 값이 오른쪽 값보다 크다	x > y → False
<	왼쪽 값이 오른쪽 값보다 작다	x < y → True
>=	왼쪽 값이 오른쪽 값보다 크거나 동일하다	x >= y → False
<=	왼쪽 값이 오른쪽 값보다 작거나 동일하다	x <= y → True

- 연산자

06 논리연산자

- 여러 개의 관계연산자를 사용할 때 사용

연산기호	설명	사용 예(x = True, y= False)
and	논리 AND 연산. 둘 다 참일 때만 참	x and y = False
or	논리 OR 연산. 둘 중 하나만 참 이어도 참	x or y = True
not	논리 NOT 연산. 논리 상태를 반전	not (x and y) = True

연산자

- 연산자

  - 논리연산 값

a	b	a and b	a or b
true	true	true	true
true	false	false	true
false	true	false	true
false	false	false	false

연산자

- 연산자

07 대입연산자

연산기호	설명	사용 예(x = 10, y= 20)
=	왼쪽 변수에 오른쪽 값을 할당	z = x + y → z = x + y
+=	왼쪽 변수에 오른쪽 값을 더하고 결과를 왼쪽변수에 할당	z += x → z = z + x
-=	왼쪽 변수에서 오른쪽 값을 빼고 결과를 왼쪽변수에 할당	z -= x → z = z - x
*=	왼쪽 변수에 오른쪽 값을 곱하고 결과를 왼쪽변수에 할당	z *= x → z = z * x

연산기호	설명	사용 예(x = 10, y= 20)
/=	왼쪽 변수에서 오른쪽 값을 나누고 결과를 왼쪽변수에 할당	z /= x → z = z / x
%=	왼쪽 변수에서 오른쪽 값을 나눈 나머지의 결과를 왼쪽변수에 할당	z %= x → z = z % x
**=	왼쪽 변수에 오른쪽 값만큼 제곱을 하고 결과를 왼쪽변수에 할당	z **= x → z = z ** x
//=	왼쪽 변수에서 오른쪽 값을 나눈 몫의 결과를 왼쪽변수에 할당	z //= x → z = z // x

- 연산자

09 멤버연산자

- x = 10, y = 20, list = [10, 20, 30, 40, 50]

연산기호	설명	사용 예
in	list 내에 포함되어 있으면 참	(x in list) = True
not in	list 내에 포함되어 있지 않으면 참	(y not in list) = False

연산자

- 연산자

10 식별연산자

- 두 개체의 **메모리 위치**를 비교함
- x = 10, y = 10

연산기호	설명	사용 예
is	개체메모리 위치나 값이 같다면 참	(x is y) = True
is not	개체메모리 위치나 값이 같지 않다면 참	(x is not y) = False

1 | 코딩해보기

01 과목별 점수의 총합과 평균 및 총점과 평균 출력

- 국어, 영어, 수학 점수의 총합과 평균을 구하고 각 과목의 점수와 총점, 평균을 출력해보자.

1 | 코딩해보기

02 성별이 남성이고 18세 이상의 성인 여부를 판단하는 조건식

- 성별이 남성이고 18세 이상의 성인 여부를 판단하는 조건식을 만들어 봅시다.

성별 판별

gender = male | female

나이 판별

age = 숫자

first coding

시퀀스 자료형

리스트 (list)

튜플 (tuple)

딕셔너리 (dict)

집합 (set)

리스트 자료형

- 리스트(list)

  - 자료형의 집합을 표현할 수 있는 자료형
    - 대괄호([ ])로 감싸고 각 요소 값은 쉼표(,)로 구분

리스트 이름 = [요소, 요소, 요소, 요소, …]

  - 숫자와 문자열만으로 프로그래밍을 하기엔 부족한 점이 많음
    - 예) 1부터 10까지의 숫자 중 홀수 모음인 집합 {1, 3, 5, 7, 9}는 숫자나 문자열로 표현 불가능
    - 리스트로 해결 가능!

```
>>> odd = [1, 3, 5, 7, 9]
```

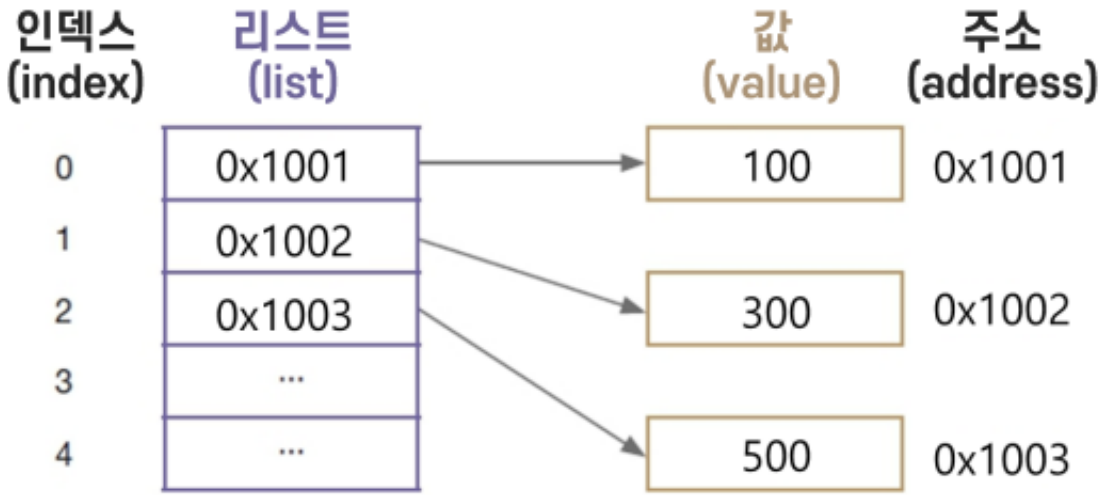
리스트 자료형

- 리스트(list)

  - 리스트의 요소 각각은 list[0], list[1], list[2], list[3] 처럼 **인덱스 번호**를 붙여 사용

리스트 자료형

- 리스트(list)

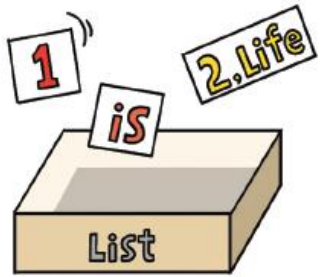


리스트 자료형

- 리스트(list)

  - 리스트의 구조
    - 리스트 안에는 어떠한 자료형도 포함 가능

```
>>> a = []
>>> b = [1, 2, 3]
>>> c = ['Life', 'is', 'too', 'short']
>>> d = [1, 2, 'Life', 'is']
>>> e = [1, 2, ['Life', 'is']]
```



리스트 자료형

- 리스트의 인덱싱

1. 문자열과 같이 인덱싱 적용 가능
2. 파이썬은 숫자를 0부터 세기 때문에 a[0]이 리스트 a의 첫 번째 요소
3. 요소 값 간의 덧셈
4. a[-1]은 리스트 a의 마지막 요소 값

```
>>> a = [1, 2, 3]
>>> a
[1, 2, 3]
```

```
>>> a[0]
1
```

```
>>> a[0] + a[2]  <-- 1 + 3
4
```

```
>>> a[-1]
3
```

리스트 자료형

- 리스트의 인덱싱

1. 리스트 내에 리스트가 있는 경우
2. a[-1]은 마지막 요소 값인 리스트 [‘a’, ‘b’, ‘c’] 반환
3. 리스트 a에 포함된 [‘a’, ‘b’, ‘c’] 리스트에서
   ‘a’ 값을 인덱싱을 사용해 반환할 방법은?
   - a[-1]로 리스트 [‘a’, ‘b’, ‘c’]에 접근하고, [0]으로 요소 ‘a’’에 접근

```
>>> a = [1, 2, 3, ['a', 'b', 'c']]
```

```
>>> a[0]
1
>>> a[-1]
['a', 'b', 'c']
>>> a[3]
['a', 'b', 'c']
```

```
>>> a[-1][0] <-- ['a', 'b', 'c'][0]
'a'
```

리스트 자료형

- 리스트의 슬라이싱

  - 문자열과 같이 슬라이싱 적용 가능
  - 슬라이싱에서는 시작 인덱스와 마지막 인덱스 마지막 자리에는 증가 값을 사용

변수명[시작index:마지막index:증가값]

```
>>> a = [1, 2, 3, 4, 5]
>>> a[0:2]
[1, 2]
```

```
>>> a = [1, 2, 3, 4, 5]
>>> b = a[:2]  <- 처음부터 a[1]까지
>>> c = a[2:]  <- a[2]부터 마지막까지
>>> b
[1, 2]
>>> c
[3, 4, 5]
```

리스트 자료형

- 리스트 연산하기

  - 더하기(+)
    - + 기호는 2개의 리스트를 합치는 기능
    - 문자열에서 “abc” + “def” = “abcdef”가 되는 것과 같은 의미

```
>>> a = [1, 2, 3]
>>> a * 3
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

  - 반복하기(*)
    - * 기호는 리스트의 반복을 의미
    - 문자열에서 “abc” * 3 = ”abcabcabc”가 되는 것과 같은 의미

```
>>> a = [1, 2, 3]
>>> b = [4, 5, 6]
>>> a + b
[1, 2, 3, 4, 5, 6]
```

리스트 자료형

- **리스트 연산하기**

  - 리스트 길이 구하기
    - len( ) 함수 사용
    - 문자열, 리스트 외에 앞으로 배울 튜플과 딕셔너리에서도 사용 가능한 내장 함수

```
>>> a = [1, 2, 3]
>>> len(a)
3
```

리스트 자료형

- 리스트의 수정과 삭제

  - 리스트의 값 수정하기
  - del 함수를 사용해 리스트 요소 삭제하기
    - del 키워드 사용

```
del [object]
```

```
>>> a = [1, 2, 3]
>>> a[2] = 4
>>> a
[1, 2, 4]
```

```
>>> a = [1, 2, 3]
>>> del a[1]
>>> a
[1, 3]
```

```
>>> a = [1, 2, 3, 4, 5]
>>> del a[2:] <— a[2]부터 끝까지 삭제
>>> a
[1, 2]
```

리스트 자료형

- 리스트 관련 함수
  - 리스트에 요소 추가하기 – append
    - 리스트의 맨 마지막에 요소 추가
    - 어떤 자료형도 추가 가능
  - 리스트 정렬 – sort
    - 리스트의 요소를 순서대로 정렬
    - 문자의 경우 알파벳 순서로 정렬 가능

```
>>> a = [1, 2, 3]
>>> a.append(4)  <- 리스트의 맨 마지막에 4를 추가
>>> a
[1, 2, 3, 4]
```

```
>>> a.append([5, 6])  <- 리스트의 맨 마지막에 [5, 6]을 추가
>>> a
[1, 2, 3, 4, [5, 6]]
```

```
>>> a = [1, 4, 3, 2]
>>> a.sort()
>>> a
[1, 2, 3, 4]
```

```
>>> a = ['a', 'c', 'b']
>>> a.sort()
>>> a
['a', 'b', 'c']
```

리스트 자료형

- **리스트 관련 함수**

  - 리스트 뒤집기 - reverse
    - 리스트를 역순으로 뒤집어 줌
    - 요소를 역순으로 정렬하는 것이 아닌, 현재의 리스트 그대로 뒤집음
  - 인덱스 반환 – index
    - 요소를 검색하여 위치 값 반환
    - 값이 존재하지 않으면, 값 오류 발생

```
>>> a = ['a', 'c', 'b']
>>> a.reverse()
>>> a
['b', 'c', 'a']
```

```
>>> a = [1, 2, 3]
>>> a.index(3) <— 3은 리스트 a의 세 번째(a[2]) 요소
2
>>> a.index(1) <— 1은 리스트 a의 첫 번째(a[0]) 요소
0
```

```
>>> a.index(0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: 0 is not in list
```

리스트 자료형

- 리스트 관련 함수

  - 리스트에 요소 삽입 - insert
    - insert(a, b)
      - a번째 위치에 b를 삽입하는 함수
  - 리스트 요소 제거 - remove
    - remove(x)
      - 리스트에서 첫 번째로 나오는 x를 삭제
      - 값이 여러 개인 경우 첫 번째 것만 삭제

```
>>> a = [1, 2, 3]
>>> a.insert(0, 4)  <-- a[0] 위치에 4 삽입
>>> a
[4, 1, 2, 3]
```

```
>>> a.insert(3, 5)  <-- a[3] 위치에 5 삽입
>>> a
[4, 1, 2, 5, 3]
```

```
>>> a = [1, 2, 3, 1, 2, 3]
>>> a.remove(3)
>>> a
[1, 2, 1, 2, 3]
```

```
>>> a.remove(3)
>>> a
[1, 2, 1, 2]
```

리스트 자료형

- **리스트 관련 함수**

  - 리스트 요소 끄집어 내기 - **pop**
    - 리스트의 맨 마지막 요소를 돌려주고 해당 요소 삭제
    - pop(x)
      - 리스트의 x번째 요소를 돌려주고 해당 요소 삭제
  - 리스트에 포함된 요소 x의 개수 세기 - **count**
    - 리스트에 포함된 요소의 개수 반환
    - count(x)
      - 리스트 안에 x가 몇 개 있는지 조사하여 그 개수를 돌려주는 함수

```
>>> a = [1, 2, 3]
>>> a.pop()
3
>>> a
[1, 2]
```

```
>>> a = [1, 2, 3]
>>> a.pop(1)
2
>>> a
[1, 3]
```

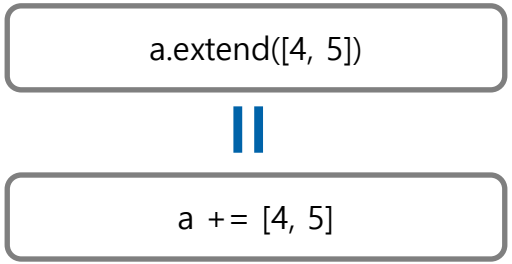
```
>>> a = [1, 2, 3, 1]
>>> a.count(1)
2
```

리스트 자료형

- **리스트 관련 함수**

  - 리스트 확장 - **extend**
    - 리스트에 리스트를 더하는 함수
    - extend(x)
      - x에는 리스트만 올 수 있음

```
>>> a = [1, 2, 3]
>>> a.extend([4, 5])
>>> a
[1, 2, 3, 4, 5]
>>> b = [6, 7]
>>> a.extend(b)
>>> a
[1, 2, 3, 4, 5, 6, 7]
```



튜플 자료형

- 튜플(tuple)이란?
  - 리스트와 유사한 자료형

리스트	튜플
[ ]로 둘러쌈	( )로 둘러쌈
요소 값 생성 / 삭제 / 수정 가능	요소 값 변경 불가능

```
t1 = ()
t2 = (1,)
t3 = (1, 2, 3)
t4 = 1, 2, 3
t5 = ('a', 'b', ('ab', 'cd'))
```

- 튜플은 1개의 요소만을 가질 때는 요소 뒤에 쉼표(,)를 반드시 붙여야 함
  (예) t2 = (1,)
- 소괄호( )를 생략해도 무방함
  (예) t4 = 1, 2, 3
- 프로그램이 실행되는 동안 값을 유지해야 한다면 튜플을, 수시로 값을 변경해야 하면 리스트 사용

튜플 자료형

- 튜플의 요소 값을 지우거나 변경하려고 하면 어떻게 해야 할까?

1. 튜플 요소 값을 삭제하려 할 때
2. 튜플 요소 값을 변경하려 할 때

```
>>> t1 = (1, 2, 'a', 'b')
>>> del t1[0]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object doesn't support item deletion
```

```
>>> t1 = (1, 2, 'a', 'b')
>>> t1[0] = 'c'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

튜플 자료형

- **튜플 다루기**

1. 인덱싱하기
2. 슬라이싱하기
3. 튜플 더하기와 곱하기
4. 튜플 길이 구하기

```
>>> t1 = (1, 2, 'a', 'b')
>>> t1[0]
1
>>> t1[3]
'b'
```

```
>>> t1 = (1, 2, 'a', 'b')
>>> t1[1:]
(2, 'a', 'b')
```

```
>>> t1 = (1, 2, 'a', 'b')
>>> t2 = (3, 4)
>>> t3 = t1 + t2
>>> t3
(1, 2, 'a', 'b', 3, 4)
```

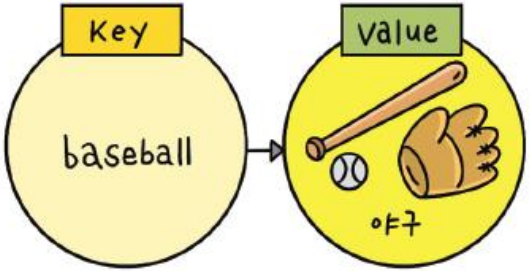
```
>>> t2 = (3, 4)
>>> t3 = t2 * 3
>>> t3
(3, 4, 3, 4, 3, 4)
```

```
>>> t1 = (1, 2, 'a', 'b')
>>> len(t1)
4
```

딕셔너리 자료형

- 딕셔너리(dictionary)

  - 대응 관계를 나타내는 자료형
  - 연관 배열(associative array) 또는 해시(hash)
  - Key와 Value를 한 쌍으로 갖는 자료형
  - 순차적으로(sequential) 해당 요소 값을 구하지 않고, Key를 통해 Value를 바로 얻는 것이 특징



딕셔너리 자료형

- 딕셔너리는 어떻게 만들까?

  - Key와 Value의 쌍 여러 개 (Key : Value)가 { }로 둘러싸임
  - 각 요소는 쉼표(,)로 구분됨

```
{key1: value1, key2: value2, key3: value3, …}
```

```
>>> dic = {'name': 'pey', 'phone': '010-9999-1234', 'birth': '1118'}
```

```
>>> a = {1: 'hi'}
```

```
>>> a = {'a': [1, 2, 3]}
```

딕셔너리 자료형

- 딕셔너리 쌍 추가, 삭제하기
  - 딕셔너리 쌍 추가하기
  - 딕셔너리 요소 삭제하기
    - del 함수 사용

```
>>> a = {1: 'a'}
>>> a[2] = 'b'  <-- {2: 'b'} 쌍 추가
>>> a
{1: 'a', 2: 'b'}
```

```
>>> a['name'] = 'pey'  <-- {'name': 'pey'} 쌍 추가
>>> a
{1: 'a', 2: 'b', 'name': 'pey'}
```

```
>>> a[3] = [1, 2, 3]  <-- {3: [1, 2, 3]} 쌍 추가
>>> a
{1: 'a', 2: 'b', 'name': 'pey', 3: [1, 2, 3]}
```

```
>>> del a[1]  <-- Key가 1인 Key: Value 쌍 삭제
>>> a
{2: 'b', 'name': 'pey', 3: [1, 2, 3]}
```

딕셔너리 자료형

- 딕셔너리를 사용하는 방법

  - 딕셔너리에서 Key를 사용해 Value 얻기
    - 리스트나 튜플, 문자열은 요소 값 접근 시 인덱싱이나 슬라이싱 기법 중 하나를 사용
    - 딕셔너리는 Key를 사용해 Value 접근

```
>>> grade = {'pey': 10, 'julliet': 99}
>>> grade['pey']      <-- Key가 'pey'인 딕셔너리의 Value를 반환
10
>>> grade['julliet'] <-- Key가 'julliet'인 딕셔너리의 Value를 반환
99
```

```
>>> a = {1:'a', 2:'b'}
>>> a[1]  <-- Key가 1인 요소의 Value를 반환
'a'
>>> a[2]  <-- Key가 2인 요소의 Value를 반환
'b'
```

딕셔너리 자료형

- 딕셔너리를 사용하는 방법

  - 딕셔너리 만들 때 주의할 사항
    - 딕셔너리에는 동일한 Key가 중복으로 존재할 수 없음
    - 리스트는 그 값이 변할 수 있기 때문에 Key로 쓸 수 없으나 Value에는 아무 값이나 가능

```
>>> a = {1:'a', 1:'b'}  <-- 1이라는 Key 값이 중복으로 사용
>>> a
{1: 'b'}  <-- 1: 'a' 쌍이 무시됨.
```

```
>>> a = {[1,2] : 'hi'}  <-- 리스트를 key로 사용
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>  <-- 오류 발생
TypeError: unhashable type: 'list'
```

딕셔너리 자료형

- 딕셔너리 관련 함수

  - Key 리스트 만들기 – **Keys**
    - Key만을 모아서 dict_keys 객체 리턴
    - 리스트처럼 사용할 수 있지만, 리스트 관련 함수(append, insert, pop, remove, sort 등)는 사용 불가능
    - dict_keys 객체를 리스트로 변환하는 방법

```
>>> a = {'name': 'pey', 'phone': '010-9999-1234', 'birth': '1118'}
>>> a.keys()
dict_keys(['name', 'phone', 'birth'])
```

```
>>> list(a.keys())
['name', 'phone', 'birth']
```

```
>>> for k in a.keys():
...     print(k)
...
name
phone
birth
```

딕셔너리 자료형

- 딕셔너리 관련 함수

  - Value 리스트 만들기 - **values**
    - Value만을 모아 dict_values 객체 리턴
  - Key, Value 쌍 얻기 - **items**
    - Key와 Value의 쌍을 튜플로 묶어 dict_items 객체 리턴

```
>>> a.values()
dict_values(['pey', '010-9999-1234', '1118'])
```

```
>>> a.items()
dict_items([('name', 'pey'), ('phone', '010-9999-1234'), ('birth', '1118')])
```

딕셔너리 자료형

- 딕셔너리 관련 함수

  - Key: Value 쌍 모두 지우기 - clear
    - 딕녀서리 내의 모든 요소 삭제
    - 빈 딕셔너리는 { }로 표현

```
>>> a.clear()
>>> a
{}
```

  - 해당 Key가 딕셔너리 안에 있는지 조사하기 - in
    - Key가 딕셔너리 안에 존재하면 True, 존재하지 않으면 False 리턴

```
>>> a = {'name':'pey', 'phone':'010-9999-1234', 'birth': '1118'}
>>> 'name' in a
True
>>> 'email' in a
False
```

딕셔너리 자료형

- 딕셔너리 관련 함수

  - Key로 Value 얻기 – **get**
    - Key에 대응되는 Value 리턴
    - 존재하지 않는 키 사용 시 None 리턴
      - 오류를 발생시키는 list와 차이가 있음
    - Key 값이 없을 경우 미리 정해 둔 디폴트 값을 대신 가져오도록 지정 가능

```
>>> a = {'name': 'pey', 'phone': '010-9999-1234', 'birth': '1118'}
>>> a.get('name')
'pey'
>>> a.get('phone')
'010-9999-1234'
```

```
>>> a = {'name':'pey', 'phone':'010-9999-1234', 'birth': '1118'}
>>> print(a.get('nokey'))
None
>>> print(a['nokey'])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'nokey'
```

```
>>> a.get('nokey', 'foo')
'foo'
```

집합 자료형

- **집합**(set)

  - 집합에 관련된 것을 쉽게 처리하기 위해 만든 자료형
  - set 키워드를 사용하여 생성
    - set( )에 리스트를 입력하여 생성
  - set( )에 문자열을 입력하여 생성

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

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

집합 자료형

- **집합 자료형의 특징**

1) 중복을 허용하지 않음
2) 순서가 없음(Unordered)

- 리스트나 튜플은 순서가 있기 때문에 인덱싱을 통해 요소 값을 얻지만 set 자료형은 순서가 없기 때문에 인덱싱 사용 불가 (딕셔너리와 유사)

- 인덱싱 사용을 원할 경우 리스트나 튜플로 변환 필요
  - list( ), tuple( ) 함수 사용

```
>>> s1 = set([1, 2, 3])
>>> l1 = list(s1) <-- 리스트로 변환
>>> l1
[1, 2, 3]
>>> l1[0]
1
>>> t1 = tuple(s1) <-- 튜플로 변환
>>> t1
(1, 2, 3)
>>> t1[0]
1
```

집합 자료형

- **교집합, 합집합, 차집합 구하기**

  - set 자료형을 유용하게 사용 가능
    - 교집합 구하기
      - '&' 기호나 intersection( ) 함수 사용
    - 합집합 구하기
      - '|' 기호나 union( ) 함수 사용
    - 차집합 구하기
      - '-' 기호나 difference( ) 함수 사용

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

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

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

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

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

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

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

집합 자료형

- 집합 자료형 관련 함수

  - 값 1개 추가하기 – add
  - 값 여러 개 추가하기 - update
  - 특정 값 제거하기 - remove

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

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

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