Python Introduction

WeGo & WeCAR

WeGo

목 차

WeGo

01 Python 소개

Introduction

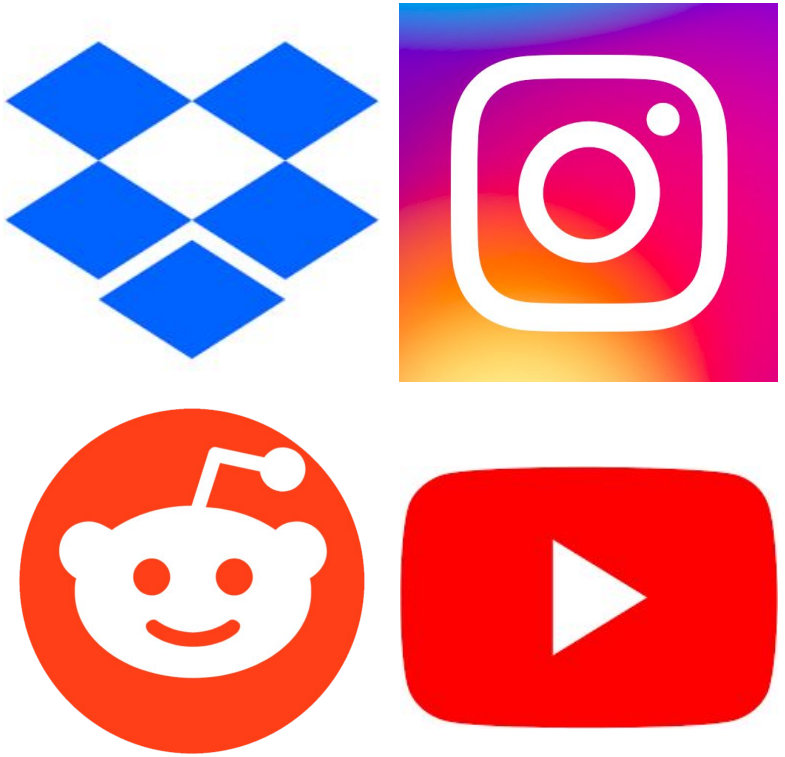
01 Python 소개

- Python은 직관적인 코딩 스타일과 뛰어난 가독성으로 인해,
  많은 개발자들이 사용하는 언어입니다.
- Python은 문법이 쉽다.
- Python은 무료이며, C와 함께 사용하기가 용이하다.
- Python은 간결하다.
- Python은 다양한 라이브러리를 가지고 있다.
- Python은 개발속도가 빠르다.

WeGo

01 Python 소개

- Python으로 할 수 있는 일
  - 시스템 유틸리티 개발
  - GUI Programming
  - C/C++ 결합 프로그래밍
  - 웹프로그래밍
  - 수치 연산 프로그래밍
  - 데이터베이스 프로그래밍
  - 데이터 분석, 사물 인터넷
- Python으로 할 수 없는 일
  - 시스템 프로그래밍 (하드웨어 관련 프로그래밍)
  - 모바일 프로그래밍 (모바일 앱)



01 Python 소개

- Python 설치
  - https://www.python.org/downloads
  - 윈도우용 파이썬 언어 패키지 설치
  - 설치 중, 어디든 실행 가능하도록 환경 변수를 등록하는 Add Python to PATH를 꼭 확인
  - 리눅스의 경우, 기본적으로 파이썬이 설치되어 있음
  - $ python -V 를 이용하여 설치된 파이썬 버전을 확인할 수 있음
  - Ubuntu에서는 $ sudo apt install python<<VERSION>>
  - $ sudo apt install python3
  - $ sudo apt install python3.6

01 Python 소개

- Python 사용 (이 후의 내용은 Linux를 기준으로 설명)
- Python은 Interpreter 언어로서, 터미널을 통해서 실시간으로 명령을 전달 가능
- 다른 언어와 같이 Script를 작성하여, Python을 실행하는 방법
  - 새로운 파일을 확장자를 .py로 생성
  - Python 문법에 맞도록 내용을 작성
  - $ python <<PYTHON.py>> 명령어를 통해 실행
  - $ python hello.py $ python3 hello.py $ python3.6 hello.py
- 이 후의 모든 내용은

  https://github.com/JacksonK9/python_study.git 에 포함되어 있습니다.

WeGo

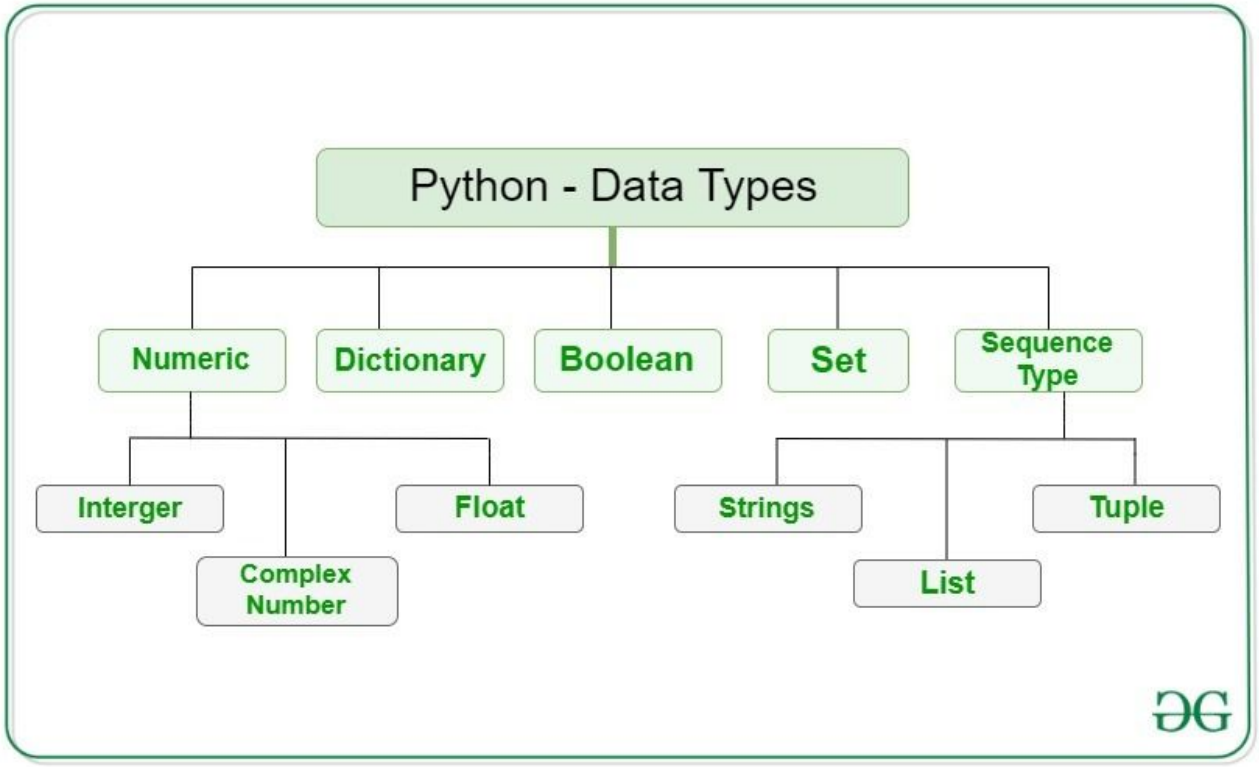
02 Python 자료형

Data type

WeGo

02 Python 자료형

- 모든 프로그래밍 언어는 자료형을 기반으로 동작하게 된다.
- Python은 자료형 변환이 자유롭고, 자동 변환 기능을 제공하기에, 사용하기 쉽다.
- Python 기본 자료형은 다음과 같이 숫자형, 문자열 자료형, 리스트 자료형, 튜플 자료형, 딕셔너리 자료형, 집합 자료형, 참과 거짓을 나타내는 자료형을 제공한다.



WeGo

02 Python 자료형

- 숫자형
- 숫자는 정수, 실수, 복소수, 2진수, 8진수, 16진수 등의 형태를 표현
- a = 123, a = -178, a = 0 (정수형)
- a = 1.2, a = -3.45, a = 4.24E10, a = 4.24e-10 (실수형)
- a = 0o177, a = 00177 (8진수)
- a = 0x8ff, a = 0xABC (16진수)
- a = 1+2j, a = 3-4J (복소수)
- 복소수의 경우, 몇가지 기능을 제공하는데, a = 1+2j에 대해, a.imag를 입력하면, 허수 부분인 2를 리턴하며, a.real은 실수부인 1.0을 리턴하게 된다. 또한, a.conjugate()를 실행하면, a의 켤레복소수인 1-2j를 리턴하게 된다.
- 또한, abs(a)를 실행하면, 복소수의 절대값을 계산하여 2.23606을 반환해준다.

02 Python 자료형

- 숫자형 연산자
- 숫자형은 기본적인 사칙연산은 모두 제공한다(+, -, *, /)
- 이 때 주의할 점은 사칙연산의 입력으로 들어가는 자료형과 같은 형태의 자료형으로 결과가 나온다는 점이며, 상황에 따라 입력의 자료형이 다를 경우, 결과의 자료형이 달라질 수 있음을 알고 있어야한다.
- x의 y제곱을 할 때는 **연산자를 사용할 수 있다. (2**3 = 8)
- 나눗셈 후, 나머지를 반환하는 % (mod) 연산자가 있다.(7 % 3 = 1)
- 나눗셈 후, 소수점 아래를 버리는 // 연산자 (7 / 4 = 1.75, 7 // 4 = 1)

02 Python 자료형

- 문자열 자료형
- 문자열은 문자 또는 단어 등으로 구성된 문자들의 집합을 의미한다.
- 대부분의 언어에서 문자열은 큰따옴표로 묶여서 표현하게 되지만, 파이썬에서는 큰따옴표 또는 작은따옴표 모두 사용할 수 있다.
- 상황에 따라 적절한 내용을 사용하면된다.
- "Hello World"
- 'Python is fun'
- """ Life is too short, You need python."""
- ''' life if too short, you need python. '''

WeGo

02 Python 자료형

- 문자열 자료형
- 문자열의 경우, 한 줄의 내용만 포함하는 것이 아닌 여러 줄의 내용을 포함할 수 있으며, 이 경우, 문자열 내부에 \n과 같은 이스케이프 코드가 포함된 부분이 존재하게 된다.
- 이스케이프 코드는 백슬래시와 다른 명령어를 합쳐서, 출력값을 보기 좋게 할 때 사용한다.
- \n → 문자열 안에서 줄을 바꿀 때 사용
- \t → 문자열 사이에 탭 간격을 줄 때 사용
- \\ → 문자 \를 그대로 표현할 때 사용
- \' → 문자 '를 그대로 표현할 때 사용
- \" → 문자 "를 그대로 표현할 때 사용
- \r → 캐리지 리턴(줄바꿈 문자, 현재 커서를 가장 앞으로 이동)
- \f → 폼 피드(줄바꿈 문자, 현재 커서를 다음 줄로 이동)
- \a → 벨 소리(출력 시 PC 스피커에서 '삑' 소리가 난다)
- \b → 백 스페이스
- \000 → 널 문자

02 Python 자료형

- 문자열 포매팅
- 문자열은 결과를 보기 좋게 출력하기 위해 자주 사용되며. 따라서 결과 부분을 상황에 따라서 다른 변수를 이용하여 전달해주어야하는 경우도 있다. 이때 문자열 포매팅을 사용한다.
- 'I eat %d apples.' % 3와 같이 입력하면. %d에 해당하는 부분이 3으로 치환되며 'I eat 3 apples.' 와 같은 결과를 출력한다.
- 이와 같이 %d 대신 %s. %f 등을 이용하여 다양한 결과를 출력할 수 있다.
- %s → 문자열로 대체할 수 있는 코드
- %d → 정수로 대체할 수 있는 코드
- %c → 문자 1개로 대체할 수 있는 코드
- %f → 부동 소수로 대체할 수 있는 코드
- %o → 8진수로 대체할 수 있는 코드
- %x → 16진수로 대체할 수 있는 코드
- %% → 문자 % 자체로 출력하는 코드

02 Python 자료형

- 문자열 포매팅
- 이와 같은 코드에 숫자를 추가하여, 더 보기 좋게 만들 수 있다.
- %10s → 10칸의 공백을 만든 후, 오른쪽 정렬로 출력
- %-10s → 10칸의 공백을 만든 후, 왼쪽 정렬로 출력
- %0.4f → 소수점 네번째 자리까지만 출력
- %10.4f → 소수점을 네번째 자리까지만 출력하며, 전체 길이가 10으로 하여 오른쪽 정렬

02 Python 자료형

- 문자열 포매팅
- %를 사용하는 포매팅 외에 다음과 같이 사용할 수도 있다.
- a = 'world', print('hello {}'.format(a))
- a = 'world', print(f'hello {a}')
- 위의 두 문장의 결과는 동일.

  Python 2에서는 일반적으로 format을 사용한 포매팅
- Python 3에서는 f-string을 권장하고 있으며, 속도가 가장 빠르다.

```
# 문자열 포매팅
a = 'I eat %d apples.'%3
tmp = 'world'
b = 'Hello {}'.format(tmp)
c = f'Hello {tmp}'
print(a)
print(b)
print(c)
```

WeGo

02 Python 자료형

- 문자열 관련 함수
- 문자열의 경우, 편의성을 위해 몇가지 내장 함수가 포함되어있다.
- count 내장함수는 개수를 세어준다. (a = 'hobby', a.count('b') = 2)
- find 내장함수는 위치를 알려준다. (a = 'Python is best choice', a.find('b') = 10, a.find('k') = -1) → 없는 경우 -1을 반환하게 된다.
- index 내장함수는 위와 마찬가지로 동작하지만, 없는 경우 오류가 발생한다.
- join 내장함수는 문자열 사이에 변수 값을 삽입한다. (a = '.', a.join('abcd') = 'a.b.c.d')
- upper 내장함수는 소문자를 대문자로 바꿔준다 (a = 'hi', a.upper() = 'HI')
- lower 내장함수는 대문자를 소문자로 바꿔준다.(a = 'HI', a.lower() = 'hi')
- strip 내장함수는 공백을 지워준다. (a = " hi ", a.strip() = 'hi')
- replace 내장함수는 문자열을 대체해준다. (a = "New world", a.replace("New", "old") = "old world")
- split 내장함수는 문자열을 나눠준다. (a = "a:b:c:d", a.split(':') = ['a', 'b', 'c', 'd']

02 Python 자료형

- 리스트 자료형
- 리스트의 경우, 다양한 자료형을 포함할 수 있으며, 중간의 값을 지우거나, 값을 추가할 경우 매우 유용하게 사용할 수 있다.
- 리스트는 대괄호로 감싸서 생성할 수 있으며, 빈 리스트의 생성도 가능하다.
- 리스트는 내부에 다양한 자료형을 포함할 수 있으며, 심지어 리스트도 포함시킬 수 있다.

02 Python 자료형

- 리스트 인덱싱 슬라이싱
- 리스트도 문자열과 유사하게 인덱싱과 슬라이싱이 가능하다.
- a = [1, 2, 3], a[0] = 1, a[0] + a[2] = 4, a[-1] = 3
- a = [1, 2, 3, ['a', 'b', 'c']], a[-1] = ['a', 'b', 'c'], a[-1][0] = 'a'
- a = [1,2,3,4,5], a[0:2] = [1, 2]
- 리스트 연산자
- 리스트를 더하면, 문자열과 같이 합쳐진다. (a = [1, 2, 3], b = [4, 5], a+b = [1, 2, 3, 4, 5]
- 리스트를 곱하면, 여러번 반복된다.(a = [1, 2, 3], a * 3 = [1, 2, 3, 1, 2, 3, 1, 2, 3]
- 리스트의 값의 변경
- a = [1, 2, 3], a[2] = 4 → a = [1, 2, 4]
- a[1:2] = ['a', 'b', 'c'] → a = [1, 'a', 'b', 'c', 4]
- a[1:3] = [] → a = [1, 'c', 4]
- del a[1] → a[1, 4]

WeGo

02 Python 자료형

- 리스트 내장함수
- 리스트에 요소를 추가하는 append (a = [1, 2, 3], a.append(4) → a = [1, 2, 3, 4], a.append([5, 6]) → a = [1, 2, 3, 4, [5, 6]])
- 리스트를 정렬하는 sort (a = [1, 4, 3, 2], a.sort() → a = [1, 2, 3, 4])
- 리스트의 순서를 반대로 바꿔주는 reverse (a = [1, 3, 2], a.reverse() → a = [2, 3, 1])
- 위치를 반환하는 index (a = [1, 2, 3], a.index(3) = 2, 없을 경우 오류 발생)
- 리스트 중간에 요소를 삽입하는 insert(a = [1, 2, 3], a.insert(0, 4) → a = [4, 1, 2, 3], a.insert(3, 5) → [4, 1, 2, 5, 3])
- 리스트 요소를 제거하는 remove (a=[1, 2, 1, 2], a.remove(2) → a[1, 1, 2], 여러 개가 존재해도 하나만 삭제되는 것을 볼 수 있다.)
- 리스트 요소를 꺼내오는 pop (a = [1, 2, 3], a.pop() =3 → a = [1, 2])
- a = [1, 2, 3], a.pop(1) = 2 → a = [1, 3]
- 리스트 내부의 요소를 세는 count (a = [1, 2, 3, 1], a.count(1) =2)
- 리스트를 확장하는 extend (a = [1, 2, 3], a.extend([4, 5]) → a = [1, 2, 3, 4, 5])

02 Python 자료형

- 튜플 자료형
- 튜플은 리스트와 거의 유사하지만, 대괄호가 아닌 소괄호를 이용하며, 튜플은 값의 변경이 불가능하다는 특징이 있다.
- a = (1, 2)와 같이 선언이 가능하며, 대부분의 슬라이싱 및 인덱싱이 동일하게 적용된다.
- 단 값의 변경 또는 삭제 등이 불가능하다는 특징을 가진다.

02 Python 자료형

- 딕셔너리 자료형
- 딕셔너리의 경우, Key와 Value 두 가지의 값으로 이루어지며, Key를 통해 Value를 얻어내는 방식으로 사용 가능한 자료형이다.
- 생성 시에는 중괄호를 이용하여 생성할 수 있다.
- a = {1 : 'a'} 와 같이 생성할 수 있으며, 중간에 키와 밸류를 추가할 때는 a[2] = 'b'와 같은 형태로 추가가 가능하다.
- 삭제 시에는 리스트와 동일하게 del[1]과 같이 삭제가 가능하다.
- 딕셔너리를 사용할 때는 키 값을 기억하고 있어야하며, 이를 통해 호출할 수 있다.

02 Python 자료형

- 딕셔너리 내장 함수
- Key 리스트를 생성하는 keys (a = {'name' : 'pey', 'phone' : '0109993323'}, a.keys() = dict_keys(['name', 'phone'])
- dict_keys 자료형의 경우, list와 유사하게 사용할 수 있지만, 약간의 제약사항이 생긴다. list와 동일하게 사용하고 싶을 경우, list(a.keys())로 사용하면 list로 결과를 받을 수 있다.
- Value 리스트를 생성하는 values(a.values = dict_values(['pey', '0109993323']))
- Key와 Value를 쌍으로 얻어오는 items (a.items() = dict_items([('name', 'pey'), ('phone', '0109993323')])
- Key와 Value를 모두 지우는 clear (a.clear() → a = {})
- Key를 이용해서 Value를 얻어오는 get
- a = {'name':'pey', 'phone':'0109993323'}
- a.get('name') = 'pey'
- a.get('hello') = None을 반환 (a['hello']로 호출 시, 오류 발생)
- Key가 있는지 조사하는 in ('name' in a → True, 'email' in a → False)

WeGo

02 Python 자료형

- 집합 자료형
- 집합에 관련된 자료형이며 set을 이용하여 생성할 수 있다.
- s1 = set([1, 2, 3]) → s1 → {1, 2, 3}
- s2 = set('Hello') → s2 → {'e', 'l', 'o', 'H'}
- 집합의 경우, 중복을 허용하지 않으며, 순서가 따로 존재하지 않는 형태이다.
- 교집합의 경우, & 연산자 또는 intersection 함수를 이용한다.
- 합집합의 경우, | 연산자 또는 union 함수를 사용한다.
- 차집합의 경우, - 연산자 또는 difference 함수를 사용한다.
- 값을 하나만 추가할 경우는 add함수를 사용할 수 있으며, 여러 개의 값을 추가할 때는 update 함수를 이용하여 추가할 수 있다.
- 특정 값을 제거할 때는 remove함수를 사용할 수 있다.

02 Python 자료형

- 자료형의 참과 거짓
- 각각의 자료형은 참과 거짓을 나타내는 특징을 나타낸다.
- 일반적으로 값이 있을 경우 참이 되며, 값이 없을 경우 거짓이 된다
- 문자열의 경우 어떤 문자라도 있을 경우 True로 취급되며, ""과 같이 아무것도 없을 경우 False가 된다.
- 리스트도 문자열과 동일하게, 값이 있을 경우, True, 없을 경우 False
- 튜플도 리스트와 동일하게 적용된다.
- 딕셔너리도 키와 밸류가 하나도 없는 경우 False 나머지는 True이다.
- 숫자형의 경우, 0이 아닌 숫자는 True로 취급되며, 0 또는 None은 False가 된다.

WeGo

03 Python 제어문

Control statement

WeGo

03 Python 제어문

- if 문
- if문은 코드의 흐름을 분기하는 역할을 하며, if, elif, else와 같이 함께 사용된다.
- 파이썬의 경우, 들여쓰기를 이용하여, 각 블록을 구분하게되며, 일반적으로 들여쓰기는 4칸을 기본적으로 사용한다. 들여쓰기가 잘못된 경우도, 오류가 발생하므로 주의해야한다.

  if 조건문:

      수행 문장 1

- 위와 같은 형태로 구성하여 쓸 수 있으며, elif, else 등도 마찬가지로 조건문 뒤에는 :를 이용하여 구분하여 수행할 문장을 적을 수 있다.
- if - elif - else의 경우, if - elif - else 순서대로 처리 되며, if 또는 elif에서 조건이 참인 경우, else는 실행되지 않는다.

WeGo

03 Python 제어문

- 조건에 들어갈 내용은 True 또는 False와 같은 Boolean 형태로 변환이 가능한 결과로 입력이 되어야한다.
- 앞서 설명한 자료형의 True, False와 같은 내용도 사용이 가능하며, 비교 연산자를 이용한 비교를 통해서도 사용이 가능하다
- x > y , x < y, x >= y, x <= y, x != y, x == y와 같은 비교 연산자를 사용할 수 있다.
- 두 가지 이상의 조건의 비교를 위해서는 or, and, not과 같은 연산자를 사용할 수 있으며, a and b인 경우, a와 b 조건이 모두 참 인 경우만 실행되고, a or b의 경우 a와 b 조건 중 하나만 참이어도 실행이 되며, not a인 경우는 a가 거짓일 경우 실행이 되는 형태이다.
- 리스트, 튜플, 문자열을 통한 내용으로는 특정 값이 리스트, 튜플, 문자열 내에 존재하는지 여부를 확인하는 방법을 통해 조건문을 실행 시킬 수 있으며, a in [list], a in [tuple], a in [string] 과 같이 확인하려는 리스트, 튜플, 문자열의 이름을 입력하면 사용할 수 있다.

03 Python 제어문

- 조건 이후, 구현 내용이 미정이거나, 아무것도 하고 싶지 않을 경우에는 pass로 처리
- pass의 경우, 이 후에 배울 class의 구체적인 구현 전에 틀을 잡을 때도 사용한다.

03 Python 제어문

- 반복문
- 반복문은 대표적으로 while과 for문이 있다.
- while문의 경우, while 조건: 과 같은 형태로 사용할 수 있으며, 조건이 참일 경우 아래의 블록이 지속해서 실행되도록 되어있다.
- for문의 경우, 특정 리스트의 값을 순서대로 불러오는 for a in [list]와 같은 형태나, 0부터 특정 값까지 지속적으로 값을 불러오며 반복하는 for i in range(x)와 같은 형태로 사용할 수 있다.
- 반복문을 특정 조건에서 이 후 부분을 무시하고 다시 실행할 때는 continue를 쓴다.
- 반복문을 특정 조건에서 강제적으로 탈출할 때는 break를 사용하면 된다.

04 Python 함수

Function

04 Python 함수

- 함수는 입력에 대해 결과를 만들어주는 역할을 하며, 일반적으로 코드를 작성할 때, 같은 내용을 여러 번 입력하는 것을 대체하여, 코드를 가독성 있게 만들어주며, 반복 작업 시 실수를 방지할 수 있게 해준다.
- 함수의 경우 오른쪽과 같은 형태로 구성되어 있다.
- def의 경우, 함수를 정의한다는 시작 구문을 의미하며, sum은 정의하려는 함수의 이름, sum뒤의 괄호 안의 a, b는 입력으로 a와 b의 2개의 변수를 받겠다는 것을 의미하며, 조건문과 유사하게, 실제 동작 내용은 : 이후에 들여쓰기를 하여 작성하게 된다.
- 최종 함수의 출력에 해당하는 값은 return 이후에 있는 내용을 출력해준다 (a + b)
- 함수의 경우, 입력값이 없는 경우 및 출력값이 없는 경우도 모두 가능하다.

```
def sum(a, b):
    return a + b
```

WeGo

04 Python 함수

- 함수를 사용할 때, 입력값의 개수를 정확하게 알지 못할 경우는 입력 변수 앞에 *을 추가하여 사용할 수 있다.
- 오른쪽과 같이 작성하면, 여러 인자를 입력 받아서
- 튜플 형태로 작성하여 함수에 전달하게 된다.
- *이후의 이름은 아무것이나 사용해도 상관 없다.
- 위와 같이 입력 *args와 같은 입력변수를 하나만
- 사용할 수도 있으며, 일반적인 변수와 함께 사용 가능

```
def sum_many(*args):
    sum = 0
    for i in args:
        sum = sum + i
    return sum
```

```
def sum_mul(choice, *args):
    if choice == 'sum':
        result = 0
        for i in args:
            result = result + i
    elif choice == 'mul':
        result = 1
        for i in args:
            result = result * i

    return result
```

WeGo

04 Python 함수

- 함수의 경우, 여러 개의 출력값을 함께 출력해줄 수 있으며, 이 경우 튜플 형태로 출력
- return 자체는 값을 반환해주는 역할 및 함수를 탈출 기능으로 사용 가능
- 함수 정의 부에서 함수에 입력되는 변수의 기본값을 a = True와 같은 형태로 지정 가능

  주의)) 기본값이 있는 변수와 없는 변수가 공존할 경우, 기본값이 있는 변수가 가장 뒤에 와야한다는 점이다.
- 일반적으로 함수 내부에서는 함수 밖에서 정의된 변수에 대해서는
- 접근이 불가능해지지만, 오른쪽과 같이 global 명령어를 통해
- 함수 내부에서 사용이 가능해지며, 오른쪽의 결과는
- a = 1 + 10이 되므로 2가 출력이 되는 것을 확인할 수 있다.

```
a = 1
def vartest():
    global a
    a = a + 1

vartest()
print(a)
```

WeGo

05 Python 클래스

Class

WeGo

05 Python 클래스

- 클래스는 객체 지향 언어의 특징으로, 물체를 정의하고, 물체에 해당하는 동작을 정의하는 것을 목표로 한다.
- C와 같은 프로그래밍 언어에서 사용하는 구조체와는 달리, 내부에서 사용할 수 있는 함수가 포함되어 있으며, 활용할 수 있는 영역이 다양하다.
- 또한, 객체에 대한 틀을 생성하는 개념으로 하나의 틀로 여러 가지 객체를 생성할 수 있다는 장점을 가지고 있다.
- 인스턴스는 클래스를 통해 생성된 객체를 의미하며, 메소드는 클래스 내부에 정의된 함수(동작)을 의미한다.
- 메소드는 필수적으로 입력으로 self를 받아야만 한다.

05 Python 클래스

- Class를 기본적으로 생성하고 정의하는 방법은 아래의 그림과 같다.
- class <Class Name>: 을 통해 정의할 class의 이름을 지정한다.
- 이 후, __init__(self)의 경우, 생성자의 역할을 한다.
- 생성자는 class를 통해 인스턴스가 생성될 때, 자동으로 동작하며, class 내부에서 사용되는 변수를 초기화하는 역할을 한다. 내부적으로 사용할 변수를 정의하는 역할도 하고 있다.
- 마지막으로 내부적으로 사용할 메소드를 정의하여 객체를 완성하게 된다.

```
class Calculator:
    def __init__(self):
        self.result = 0

    def adder(self, num):
        self.result += num
        return self.result
```

WeGo

```python
class FourCal:
    def __init__(self):
        self.first = 0
        self.second = 0
        self.result = 0
    def setdata(self, first, second):
        self.first = first
        self.second = second

    def sum(self):
        self.result = self.first + self.second
        return self.result

    def sub(self):
        self.result = self.first - self.second
        return self.result

    def mul(self):
        self.result = self.first * self.second
        return self.result

    def div(self):
        self.result = self.first / self.second
        return self.result
```

05 Python 클래스

- Class는 상속 기능을 가지고 있다.
- 상속은 말 그대로 물려받는 기능이며, 부모의 기능을 그대로 받아올 수 있으며, 이에 대해 추가적으로 기능을 정의하여 사용할 수 있다.
- Method overriding이란, 상속 받은 객체에서 동일한 이름으로 다른 역할을 하는 메소드를 생성하는 것을 의미하며, 이 경우 부모에 있는 메소드보다 자식에 있는 메소드가 우선적으로 동작하게 된다.

05 Python 클래스

```python
class Country:
    """Super Class"""

    name = '국가명'
    population = '인구'
    capital = '수도'

    def show(self):
        print('국가 클래스의 메소드입니다')

    def show_name(self):
        print('국가의 이름을 알려줍니다')

class Korea(Country):
    """Sub Class"""

    def __init__(self, name):
        self.name = name

    def show_name(self):
        print(f'국가 이름은 : {self.name}')
```

```python
tmp = Country()
tmp.show()
tmp.show_name()
print(tmp.capital)
print(tmp.population)
print(tmp.name)
a = Korea('대한민국')
a.show()
a.show_name()
print(a.capital)
print(a.population)
print(a.name)
```

```
국가 클래스의 메소드입니다
국가의 이름을 알려줍니다
수도
인구
국가명
국가 클래스의 메소드입니다
국가 이름은 : 대한민국
수도
인구
대한민국
```

05 Python 클래스

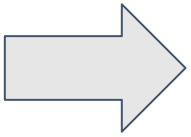
- Method overriding과 유사하게, 연산자 오버로딩(Overloading)도 존재한다.
- 이는 수식에 사용하는 연산자(+, -, *, /) 를 객체끼리 사용할 수 있게 하는 기능이다.
- 내부 메소드에서 아래와 같은 이름으로 정의하면 연산자로 사용할 수 있다.

메서드(Method)	연산자(Operator)	사용 예
__add__(self, other)	+ (이항)	A + B, A += B
__pos__(self)	+ (단항)	+A
__sub__(self, other)	- (이항)	A - B, A -= B
__neg__(self)	- (단항)	-A
__mul__(self, other)	*	A * B, A *= B
__truediv__(self, other)	/	A / B, A /= B
__floordiv__(self, other)	//	A // B, A //= B
__mod__(self, other)	%	A % B, A %= B
__pow__(self, other)	pow(), **	pow(A, B), A ** B
__lshift__(self, other)	<<	A << B, A <<= B
__rshift__(self, other)	>>	A >> B, A >>= B
__and__(self, other)	&	A & B, A &= B
__xor__(self, other)	^	A ^ B, A ^= B
__or__(self, other)	|	A | B, A |= B
__invert__(self)	~	~A
__abs__(self)	abs()	abs(A)

WeGo

```python
class NumBox:
    def __init__(self, num):
        self.Num = num
    def __add__(self, num):
        self.Num += num
    def __sub__(self, num):
        self.Num -= num

n = NumBox(40)
n + 100
print(n.Num)
n - 110
print(n.Num)
```



```
140
30
```

06 Python 모듈&패키지

Module

& Package

WeGo

06 Python 모듈 & 패키지

- Module은 함수, 변수 또는 클래스를 모아놓은 파일을 의미한다.
- 사용을 위해서 import 명령어를 통해 다른 스크립트에서 모듈을 불러올 수 있다.
- 특정 함수 또는 클래스만 불러오고 싶을 때는

  from <모듈 이름> import <함수 or 클래스 이름> 으로 불러올 수도 있다.
- 오픈 소스를 보다보면, if __name__ == "__main__": 와 같은 구문을 자주 볼 수 있는데,

  이는 모듈로 호출할 때는 동작하지 않고, 모듈을 스크립트로 직접 실행할 때만 동작하도록

  하는 기능을 부여하는 구문이다.
- Package는 여러 개의 Module을 계층적으로 묶어놓은 것을 의미하며, 실질적인 사용

  방법은 계층에 따라 도트(.)을 이용하여 import하여 사용할 수 있다.

07 Python 예외처리

Exception

WeGo

07 Python 예외처리

- 코드 작성 중에는 수많은 오류가 발생하는데, Python에서는 두 가지 종류의 오류가 있다.
- 첫 번째로는 문법 자체적인 오류로, 스크립트를 실제 구동 자체가 불가능한 오류가 있다.
- 두 번째로는 문법 자체는 문제가 없으나, 내부적으로 동작할 때 발생하는 예외가 있다.
- 이러한 예외 발생을 처리하는 방법으로는 try, except 구문을 이용할 수 있다.
- try, except는 if, else와 동일한 문법으로 작성할 수 있으며, try 블록에 있는 구문을 우선 실행하고, 이에서 예외가 발생할 경우, except 블록이 동작하게 된다.
- 만약 try 블록에서 예외가 발생하지 않을 경우, except는 동작하지 않는다.
- 또한 except는 여러 가지로 나누어, 발생하는 예외의 종류에 따라 나누어서 동작하게 할 수 있다.(except 발생 오류: 또는 except 발생 오류 as 오류 메시지 변수:)

```
try:
    4/0
except ZeroDivisionError as e:
    print(e)
```

WeGo

07 Python 예외처리

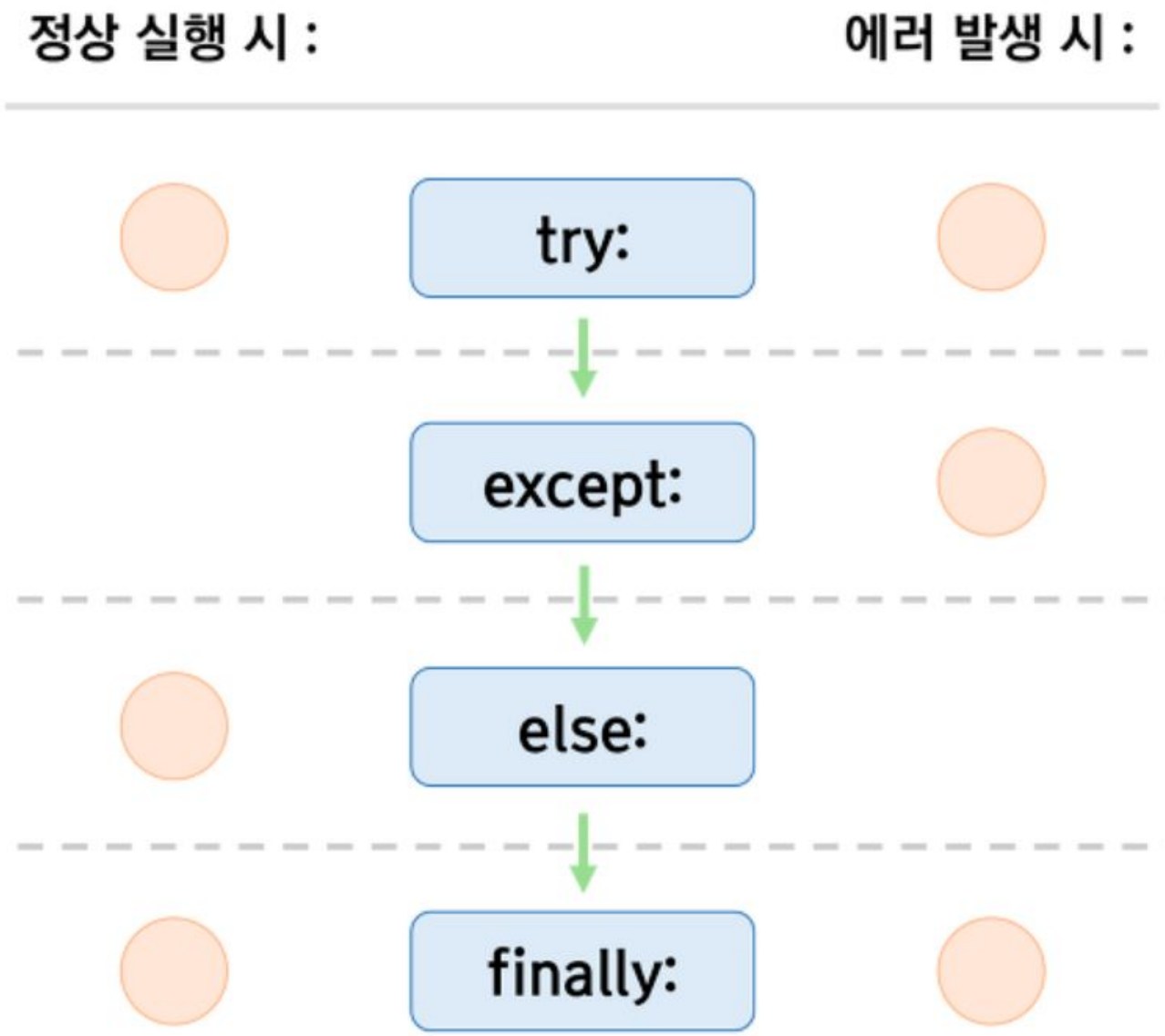
- try, except 이 외에, else와 finally를 함께 사용할 수 있는데, else의 경우 try 구문이 예외 없이 동작했을 때만 실행이 되며, finally의 경우 try 구문에서 예외가 있으나 없으나 상관 없이 무조건 동작하게 된다.
- 또한 pass 명령어를 통해, except를 동작 없이 넘어가게 할 수 있다.
- 전체적인 안정성을 위해, 일부러 예외를 발생시킬 수도 있는데, raise 명령어를 이용하여, 직접 예외 발생도 가능하다.
- 아래의 구문을 실행하면, NotImplementedError가 발생한다.

```
class Bird:
    def fly(self):
        raise NotImplementedError

class Eagle(Bird):
    pass

eagle = Eagle()
eagle.fly()
```

07 Python 예외처리



08 Python 내장 함수

Built-In Function

08 Python 내장 함수

- 내장 함수(Built-In Function)은 다른 모듈을 임포트 하지 않고도 사용 가능한 함수들을 의미하며, 이는 파이썬 배포판에 이미 포함되어있다.
- abs() → 절대값을 반환해준다
- all() → iterable한 자료형을 입력으로 받아서, 내용이 모두 True이면 True 하나라도 False이면 False를 리턴
- any() → iterable한 자료형을 입력으로 받아서, 하나라도 참이면 True, 아닐 경우 False
- chr() → ASCII Code를 입력으로 받아서, 이에 해당하는 문자를 출력해준다.
- dir() → 객체가 자체적으로 가지고 있는 변수와 함수를 보여준다.
- divmod() → a, b 두개의 숫자를 입력으로 받아서, a를 b로 나눈 몫과 나머지를 튜플로 리턴
- enumerate() → 리스트, 튜플, 문자열을 받아서, 인덱스와 값을 포함하여 리턴
- eval() → 실행 가능한 문자열을 입력으로 받아서, 문자열을 직접 실행한 결과를 리턴
- filter() → 첫번째 인자로 함수 이름을, 두번째 인자로 반복 가능한 자료형을 입력 받아, 리턴이 참인 것만 묶어서 돌려준다.

08 Python 내장 함수

- 내장 함수(Built-In Function)은 다른 모듈을 임포트 하지 않고도 사용 가능한 함수들을 의미하며, 이는 파이썬 배포판에 이미 포함되어있다.
- hex() → 입력된 정수값을 16진수로 변환하여 리턴
- id() → 객체의 고유 주소값을 리턴해주는 함수
- input() → 사용자의 입력을 받는 함수
- int() → 입력받은 값을 정수형으로 형 변환하는 함수, int(x, n) 로 받으면, n진수로 받은 값을 변환
- isinstance() → 첫 번째는 인스턴스, 두 번째는 클래스 이름을 입력하여, 입력된 인스턴스가 클래스로 만들어진 인스턴스 인지 판별해준다.
- lambda() → 간략한 함수 생성을 위한 예약어로, lambda a, b : a+b와 같이 사용하며, 입력으로 받는 값을 a, b, c, ...과 같이 입력하고 : 후 결과에 해당하는 내용을 적으면 함수가된다.
- len() → 입력값의 길이를 반환해주는 함수이다.
- list() → 반복 가능한 자료형을 입력받아 리스트로 변환해준다.

08 Python 내장 함수

- 내장 함수(Built-In Function)은 다른 모듈을 임포트 하지 않고도 사용 가능한 함수들을 의미하며, 이는 파이썬 배포판에 이미 포함되어있다.
- map() → 입력으로 함수와 반복 가능한 자료형을 입력으로 받아서 자료형의 각 요소가 함수에 각각 입력으로 동작했을 때의 결과를 묶어서 돌려준다.
- max() → max는 반복 가능한 자료형을 입력 받아서, 최대값을 돌려준다.
- min() → max와 반대로 최소값을 돌려준다.
- oct() → 정수 형태의 숫자를 8진수로 변환하여 리턴한다.
- ord() → 문자의 아스키 코드를 리턴해준다.
- pow() → a, b 두개의 숫자를 입력으로 받아서, a의 b제곱을 리턴해준다.
- sorted() → 입력값을 정렬하여 리스트로 리턴해준다.
- str() → 문자열 형태로 객체를 변환하는 형변환
- tuple() → list()와 동일하며 결과를 튜플로 변환한다.
- type() → 입력값의 자료형이 무엇인지 알려준다.
- zip() → 동일한 개수로 이루어진 자료형을 묶어준다.

09 Python 외장 함수

Externel Function

09 Python 외장 함수

- 외장 함수(External Function)은 반대로, 사용자들이 만들어놓은 내용을 사용하는 것을 의미하며, 필요에 따라 라이브러리를 불러서 사용할 수 있다(임포트 이용)
- sys 모듈은 인터프리터가 제공하는 변수들과 함수들을 직접 제어할 수 있게 해준다.
- 실행을 위해서는 아래와 같이 직접 python 모듈을 실행하면서 뒤에 다른 내용을 전달할 수 있으며, 이 경우, sys.argv에 자동으로 입력이 되는 것을 확인할 수 있다.

```
import sys
print(sys.argv)
```

```
wego/~/python_study/ python3 sys_module.py hello world this is python
['sys_module.py', 'hello', 'world', 'this', 'is', 'python']
```

WeGo

09 Python 외장 함수

- 외장 함수(External Function)은 반대로, 사용자들이 만들어놓은 내용을 사용하는 것을 의미하며, 필요에 따라 라이브러리를 불러서 사용할 수 있다(임포트 이용)
- pickle은 객체의 형태를 그대로 유지하면서 파일로 저장 및 불러오는 모듈

```
import pickle
f = open('test.txt', 'wb')
data = {1:'python', 2:'you need'}
pickle.dump(data, f)
f.close()

del data

try:
    print(data)
except:
    pass
f = open('test.txt', 'rb')
data = pickle.load(f)
print(data)
```

```
{1: 'python', 2: 'you need'}
```

WeGo

09 Python 외장 함수

- 외장 함수(External Function)은 반대로, 사용자들이 만들어놓은 내용을 사용하는 것을 의미하며, 필요에 따라 라이브러리를 불러서 사용할 수 있다(임포트 이용)
- OS모듈은 환경 변수나 디렉터리, 파일 등의 OS자원을 제어할 수 있게 해주는 모듈이다.
- Linux에서 사용할 수 있는 폴더 사이의 이동 및 폴더 생성 및 삭제 등이 가능하다.

09 Python 외장 함수

- 외장 함수(External Function)은 반대로, 사용자들이 만들어놓은 내용을 사용하는 것을 의미하며, 필요에 따라 라이브러리를 불러서 사용할 수 있다(임포트 이용)
- time모듈은 시간에 관련된 유용한 함수들이 많이 있다.

```
# -*- coding: utf-8 -*-

import time

print(time.time()) # 1970년 1월 1일 0시 0분 0초를 기준으로 경과한 Time을 초단위로 리턴
print(time.localtime(time.time())) # time.time를 입력으로 받아서, 이를 연도, 월, 일, 시, 분 초의 형태로 변환해준다.
print(time.asctime(time.localtime(time.time()))) # 날짜와 time을 알아보기 쉬운 형태로 리턴하는 함수
print(time.ctime()) # 항상 현재 time을 리턴하는 함수
time.sleep(10) # 코드를 10초 동안 정지 시키는 함수
```

09 Python 외장 함수

- 외장 함수(External Function)은 반대로, 사용자들이 만들어 놓은 내용을 사용하는 것을 의미하며, 필요에 따라 라이브러리를 불러서 사용할 수 있다(임포트 이용)
- random 모듈은 난수 발생에 관련된 모듈이다.

```python
# -*- coding: utf-8 -*-

import random
def random_pop(data):
    number = random.randint(0, len(data)-1)
    return data.pop(number)

if __name__ == "__main__":
    data = [1, 2, 3, 4, 5]
    while data:
        print(random_pop(data))

print(random.randint(1, 10)) # 1 ~ 10 사이의 정수 중 하나를 생성

print(random.random()) # 0~1 사이의 난수 생성
```

WeGo

09 Python 외장 함수

- 외장 함수(External Function)은 반대로, 사용자들이 만들어놓은 내용을 사용하는 것을 의미하며, 필요에 따라 라이브러리를 불러서 사용할 수 있다(임포트 이용)
- Threading 모듈은 프로세스를 Thread로 분할 → 여러 일을 동시에 처리하는 기능

```
# -*- coding: utf-8 -*-

import threading
import time

def say(msg):
    while True:
        time.sleep(1)
        print(msg)

for msg in ['you', 'need', 'python']:
    t = threading.Thread(target=say, args=(msg,))
    t.daemon = True
    t.start()

for i in range(100):
    time.sleep(0.1)
    print(i)
```

WeGo Robotics

Tel. 031 – 229 – 3553

Fax. 031 – 229 – 3554

제품 문의: go.sales@wego-robotics.com

기술 문의: go.support@wego-robotics.com