기상청

2. Python 프로그래밍

Python을 활용한 분석 기초 가이드 북

Ⅱ. Python 프로그래밍

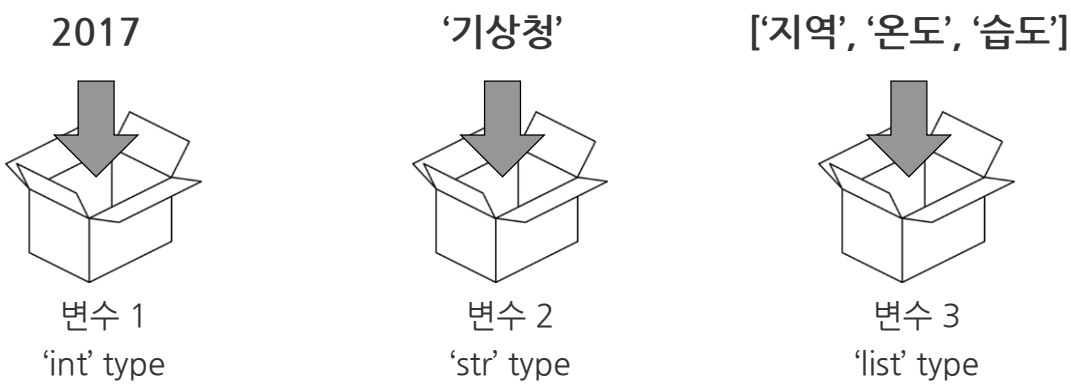
1. 자료형
2. 제어문
3. 함수

1. 자료형 (1/3)

Python의 기본 자료형(Data Type; 데이터 또는 자료의 형식)에 대해 배워봅니다. 본 장에서는 변수, 부울, 숫자, 문자, 리스트, 딕셔너리에 대해 알아봅니다.

● 변수

- 변수는 데이터를 담는 메모리 공간이며, 이름을 지정할 수 있습니다.
- 변수에는 부울, 숫자, 문자, 목록, 이미지 등을 담을 수 있으며, Python은 코드가 실행될 때 변수의 자료형을 판단하는 동적 형식 언어(Dynamic typed language)입니다.



● 부울

- 부울(bool 혹은 불린 boolean) 자료형은 논리 자료형이라고도 하며, 참(True)과 거짓(False)을 나타내는데 쓰입니다.
- 주로 참은 1, 거짓은 0에 대응하며, True와 False 문자를 그대로 사용하기도 합니다.
- 숫자의 대소나 논리연산을 통한 결과로 참과 거짓을 출력할 때 사용하거나, 프로그램의 흐름을 제어할 때 주로 사용합니다. 부울형의 쓰임에 대해서는 제어문에서 조금 더 다루어 보겠습니다.

TRUE(1) or FALSE(0)

1. 자료형 (2/3)

● 숫자

- Python은 기본적으로 세 종류의 수(정수, 실수, 복소수)를 지원합니다.
- 정수(int) : 정수는 음의 정수, 0, 양의 정수를 모두 포함하며, 메모리가 허용하는 한 무한대의 정수를 다룰 수 있습니다.
- 실수(float) : 컴퓨터로 소수의 소수점을 표현하는 방식 중 하나인 부동 소수형을 제공합니다. 여기서 부동 소수형이란, 소수점을 움직여서 소수를 표현하는 자료형입니다.
- 복소수(complex) : 실수와 허수의 합으로 이루어진 수로, Python에서는 허수 단위를 나타내는 부호로 i 대신 j를 사용합니다.
- 숫자형을 이용한 기본적인 사칙연산과 math 모듈을 이용해 파이, 자연상수, 팩토리얼, 도, 라디안, 삼각함수, 로그 등을 계산할 수 있습니다.

● 문자

- 텍스트를 다루는 자료형으로 string(문자형)을 제공합니다.
- Python 코드에서는 문자열 데이터를 작은따옴표 ’ 또는 큰따옴표 ”의 쌍으로 텍스트를 감싸서 표현하며, 여러 줄로 이루어진 문자열은 작은따옴표 3개 ’’’ 또는 큰따옴표 3개 ”””의 쌍으로 텍스트를 감싸서 표현합니다.
- 숫자와 문자는 int(), float(), complex(), str() 등의 함수를 이용해 형태를 변경할 수 있습니다.
- 문자열을 합칠 때에는 + 연산자를 사용합니다.

‘기상청’

‘‘‘기상청
2017 파이팅
’’’

1. 자료형 (3/3)

● 리스트

- 데이터의 목록을 다루는 자료형으로 list를 제공합니다. 리스트를 알아두면 많은 데이터를 다룰 때 한결 정돈된 코드를 만들 수 있습니다.
- 리스트 안에는 어떤 데이터든 넣을 수 있으며, 리스트 안의 개별 데이터를 일컬어 요소(Element)라고 합니다.
- 리스트를 만들 때는 대괄호 [와] 사이에 데이터 또는 변수 목록을 입력하며, 각 데이터는 콤마 , 로 구분합니다.
- 리스트 안에 있는 개별 데이터를 참조할 때는 리스트의 이름 뒤에 대괄호를 붙이고 [와] 사이에 참조하고자 하는 첨자를 입력하면 됩니다.
- 리스트를 결합할 때에는 + 연산자를 사용합니다.

```
['기상청', 2017, '파이팅']
```

● 딕셔너리

- 딕셔너리는 키-값의 쌍으로 구성되며, 첨자는 키(Key)라 하고, 이 키가 가리키는 데이터를 일컬어 값(Value)라고 합니다.
- 딕셔너리(Dictionary)는 리스트처럼 첨자를 이용해서 요소에 접근하고, 또 그 요소를 변경할 수 있습니다. 보통 탐색 속도가 빠르고, 사용하기 편리합니다.
- 리스트와 다른 점이라면 리스트는 데이터를 저장할 요소의 위치로 인덱스를 사용하는 반면에, 딕셔너리는 키 데이터를 그대로 사용합니다.
- Python에서 딕셔너리를 만들 때는 중괄호 {와 }을 이용합니다.
- 새로운 키-값을 입력하거나 그 안에 있는 요소를 참조할 때는 리스트처럼 대괄호 [와]를 이용합니다.

```
{'key': 'value'}
```

2. 제어문 (1/3)

프로그램의 흐름을 제어하기 위해 Python에서 사용되는 기본 제어문(Statements)에 대해 배워봅니다. 본 장에서는 연산자, if, while, for에 대해 알아봅니다.

● 연산자

• 논리연산자

연산자	설명
not(!)	피연산자 부정한 결과를 True/False로 반환
and(&)	두 피연산자 간의 논리곱 결과를 반환, 두 피연산자가 True인 경우 True, 그렇지 않은 경우는 항상 False로 반환
or(|)	두 피연산자 간의 논리합 결과를 반환, 두 피연산자가 False인 경우 False, 그렇지 않은 경우는 항상 True로 반환

• 비교연산자

연산자	설명
==	양쪽에 위치한 피연산자가 같으면 TRUE, 다르면 FALSE
!=	양쪽에 위치한 피연산자가 다르면 TRUE, 다르면 FALSE
>	왼쪽에 위치한 피연산자가 오른쪽 피연산자보다 크면 TRUE, 다르면 FALSE
>=	왼쪽에 위치한 피연산자가 오른쪽 피연산자보다 크거나 같으면 TRUE, 다르면 FALSE
<	왼쪽에 위치한 피연산자가 오른쪽 피연산자보다 작으면 TRUE, 다르면 FALSE
<=	왼쪽에 위치한 피연산자가 오른쪽 피연산자보다 작거나 같으면 TRUE, 다르면 FALSE

2. 제어문 (2/3)

프로그램의 흐름을 제어하기 위해 Python에서 사용되는 기본 제어문(Statements)에 대해 배워봅니다. 본 장에서는 연산자, if, while, for에 대해 알아봅니다.

if

- if 제어문을 이용해 조건을 사용해서 조건에 맞는 경우 프로그램이 수행되도록 할 수 있습니다. If 문의 기본 구조는 아래와 같습니다.

[구조]

```
if 조건 :
    수행할 코드 1
else :
    수행할 코드 2
```

[예제]

```
alarm = "강우"
If  name == "강우" :
    print("우산을 챙기세요")
else :
    print("오늘 날씨는?")
```

- 만약, 다중 조건을 이용할 경우 elif 를 이용해 구조화할 수 있습니다. elif 는 위에서 먼저 제시된 조건이 거짓인 경우에 수행되는 결과입니다. elif 는 else if 를 줄여 만든 파이썬 키워드 입니다. elif 를 포함한 예제는 아래와 같습니다.

[구조]

```
if 조건1 :
    수행할 코드 1
elif 조건2 :
    수행할 코드 2
else :
    수행할 코드 3
```

[예제]

```
RN_DAY = 10
If  RN_DAY >= 20 :
    print("폭우경보")
elif RN_DAY >= 10 :
    print("폭우주의보")
else :
    print("정상")
```

2. 제어문 (3/3)

프로그램의 흐름을 제어하기 위해 Python에서 사용되는 기본 제어문(Statements)에 대해 배워봅니다. 본 장에서는 연산자, if, while, for에 대해 알아봅니다.

● while

• 조건이 참인 동안 코드를 반복 수행합니다. while 을 사용할 때는 종료 조건을 지정하여 무한 루프에 걸리지 않도록 주의해야 합니다.

[구조]

```
while 조건 :
    수행할 코드
```

[예제]

```
i = 0
while i < 10 :
    print i
    i = i + 1
```

● for

• 순서열의 끝에 도달할 때까지 반복 수행. 순서열은 range() 함수를 많이 이용하며, range() 함수에 시작값, 멈춤값, 그리고 두 수의 차(간격)을 대입하여 순서열 생성

[구조]

```
for 반복변수 in 순서열 :
    수행할 코드
```

[예제]

```
for i in [1, 2, 3] :
    print (i)
```

```
for i in range(0, 10, 2) :
    print (i)
```

반복문 제어하기

• **continue**
- 특정 조건에서만 코드 실행없이 다음 반복으로 넘어갈 때 사용

```
num = 0
while num < 10:
    num += 1
    if num == 5:
        continue
    print(num)
```

• **break**
- 특정 조건에서만 반복문을 중단할 때 사용

```
num = 0
while 1:
    print(num)
    if num == 10:
        break
    num += 1
```

* while 함수를 예시로 들었으나, for에서도 continue 와 break 을 이용해 제어할 수 있음

3. 함수

Python을 이용해 함수(Function)를 정의하는 것과 함수를 호출하고 반환하는 과정에 대해 알아봅니다.

● 정의

• 정의를 뜻하는 difinition을 줄인 def 키워드를 이용해 함수를 정의할 수 있습니다. 정의는 어떤 이름을 가진 함수가 어떻게 동작하는지 '구체적으로 기술'하는 것을 말합니다.

[구조] * 함수 정의의 기본 구조

```
def 함수명(매개변수 목록) :
    수행할 코드
    return 결과
```

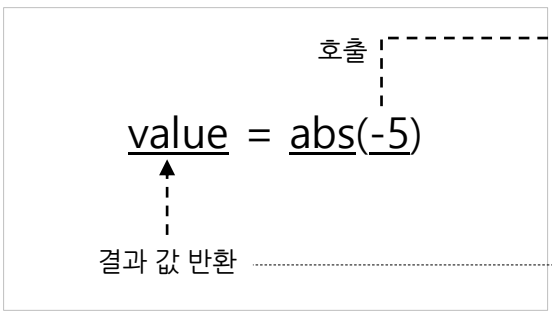
[예제]

```
def hello() :
    print("hello world")
```

● 호출과 반환

• 호출(Call)은 함수를 부르는 행위를 의미하고, 이때 함수를 부르는 코드를 호출자(Caller)라고 합니다. 그리고 함수가 호출자에게 결과를 주는 것을 반환(Return)이라고 합니다.

[원리]



[예제]

```
def abs(num) :
    if(num < 0) :
        result = num * -1
    else:
        result = num

    return result
```

본 문서의 내용은 기상청의 기상기후 빅데이터 분석(http://bd.kma.go.kr)의
분석 플랫폼 활용을 위한 Python 분석 기초 교육 자료 입니다.