변수 및 자료형

박범진

Department of Statistics
University of Seoul

변수 I

변수

- 프로그래밍에서 변수는 어떤 값 혹은 데이터가 저장된 이름을 가진 공간을 나타냄.
- 변수를 통해 우리는 Python이 실행되고 있는 동안 메모리에 Python에서 처리되는 대부분의 것들을 이름을 붙여 저장할 수 있음.

변수 II

- 예를 들어, Python을 이용해 123456789 + 123456789를 계산했다고 하자. 123456789 + 123456789의 결과에 2배를 하고싶다면 2 * (123456789 + 123456789)와 같이 또 타이핑해야함.
- 하지만 변수를 이용해 x라는 변수에 123456789 + 123456789의 결과를 저장해놓으면 두배를 계산하는 것은 쉽게 2 * x로 계산할 수 있고 3배도 3 * x로 쉽게 계산이 가능함.

변수 이름

변수 이름 짓기

- 변수의 이름을 짓는 것에는 몇가지 규칙이 존재함.
  1. 변수의 첫시작은 숫자가 올 수 없다.
  2. 연산자(operator)는 변수에 사용할 수 없다.
     - 예를 들어 *, /, - 등
  3. 특수문자는 변수에 사용할 수 없다.
     - 예를 들어 $, , 등
- 이외에 영문, 영문과 숫자 혼용, 한글 변수명 모두 가능함.
- 하지만 한글 변수명은 추천하지 않음 (인코딩 문제가 자주 발생함).

변수 할당

- Python에서 변수 할당 방법은 = 연산자를 이용.

```
# Variable assignment
>>> x = 1 + 2
>>> x
3
```

- Python에서는 다중 할당도 가능.

```
# Multiple assignment
>>> x, y, z = 1 + 2, 2 + 3, 3 + 4
>>> x
3
>>> y
5
>>> z
7
```

자료의 종류 I

자료의 종류

- Python에는 다양한 종류의 자료형과 객체가 존재함.

  자료의 종류

  - 숫자형, 문자 그리고 논리형 (bool형).

  객체의 종류

  - 리스트, 튜플, 딕셔너리 그리고 집합 등.

자료의 종류 II

숫자형

- Python에서 숫자는 크게 integer(정수), float(실수) 그리고 complex(복소수)로 나눌 수 있음.
- integer는 수학에서 정수에 해당하며 일반적으로 소수점이 없는 숫자는 정수형을 가짐.
- float은 수학에서 실수에 해당하고 소수점이 존재하는 수나 나눗셈등의 연산시 산출되는 수는 float형을 가짐.
- complex는 복소수로 허수 부분에 j가 포함된 형태를 나타냄.
- Python에서 정수형과 실수형의 구분은 엄격하지는 않지만 때에 따라 구분해야하는 경우가 있으므로 주의해야함.

자료의 종류 III

- type()라는 함수를 통해 해당 숫자의 종류를 알 수 있음.
- int(), float(), complex() 함수로 다른 종류의 숫자를 함수에 해당하는 숫자 종류로 바꿀 수 있음.
- isinstance() 함수로 입력되는 숫자가 원하는 숫자형이 맞는지 확인이 가능함.

```
# integer
>>> type(1)
<class 'int'>
# float
>>> type(1.1)
<class 'float'>
# complex
>>> type(1 + 3j)
<class 'complex'>
```

자료의 종류 IV

```
# convert to integer
>>> int(3.1)
3
# convert to float
>>> float(1)
1.0
# convert to complex
>>> complex(1)
(1+0j)
```

자료의 종류 V

```
# check integer type
>>> isinstance(1, int)
True
# check float type
>>> isinstance(1, float)
False
>>> isinstance(1.1, float)
True
>>> isinstance(1 + 2j, complex)
True
```

자료의 종류 VI

사칙연산

- 숫자형에 대한 사칙연산
  - 덧셈 : +
  - 뺄셈 : -
  - 곱셈 : *
  - 나눗셈 : /
  - 거듭제곱 : **
  - 나머지 : %
  - 몫 : //

자료의 종류 VII

```
# Add
>>> 1 + 2
3

# Subtraction
>>> 4 - 10
-6

# Multiplication
>>> 2 * 2
4

# Division
>>> 4 / 2
2.0
```

- 나눗셈은 정수(integer)끼리 나누어도 결과는 실수(float)임을 주의!

자료의 종류 VIII

```
# Power
>>> 2 ** 3
9

# Quotient
>>> 5 // 2
2

# Remainder
>>> 5 % 2
1
```

자료의 종류 IX

문자(string)

- Python에서 문자(string)는 "(큰따옴표) 또는 '(작은따옴표)로 표시함.
- 예를 들어 '1'은 숫자 1이 아닌 문자 1을 나타냄.
- str() 함수를 통해 문자로 변환할 수 있음.

```
# character
>>> "1"
'1'

>>> type("1")
<class 'str'>

>>> str(100)
'100'
```

자료의 종류 X

- 만약 문자에 큰따옴표를 넣고 싶다면 작은따옴표로 묶어주고 작은따옴표를 넣고 싶다면 큰따옴표로 묶어주면 됨.

```
# character
>>> x = "'Hello~'"
>>> x
"'Hello~'"

>>> y = '"Hello"'
>>> y
'"Hello"'

>>> x = ""Hello~""
SyntaxError: invalid syntax (<string>, line 1)
```

자료의 종류 XI

- 만약 문자를 여러줄에 걸쳐 작성하고 싶다면 작은따옴표 세개 '''나 큰따옴표 세개 """로 묶어주면 됨.

```
>>> """
␣␣␣␣Hi␣~
␣␣␣␣My␣name␣is␣Beomjin␣Park
␣␣␣␣"""
>>> '''
␣␣␣␣Hi␣~
␣␣␣␣my␣name␣is␣Beomjin␣Park
␣␣␣␣'''
```

자료의 종류 XI

문자열에서의 연산자

- Python에서는 문자형에 대해 더하기 ("+") 와 곱하기 ("*") 연산이 가능함.
- "+"는 두개의 문자열을 연결해주고 "*"는 해당 문자열을 n배 반복함.

```
>>> x = "My␣name␣is␣Beomjin␣Park"
>>> x + "ABC"
'My␣name␣is␣Beomjin␣ParkABC'
>>> x * 2
'My␣name␣is␣Beomjin␣ParkMy␣name␣is␣Beomjin␣Park'
```

자료의 종류 XII

문자열 포맷팅 (Formatting)

- 문자열 포맷팅은 반복적으로 출력하는 문자열에서 특정 값 또는 문자만 변화하여 출력할 때 사용됨.
- 예를 들어 "My name is Beomjin Park"과 "My name is Sion Park" 이라는 두 문자열이 존재할 때, "My name is"는 반복되는 문자열로 "Beomjin Park"부분만 "Sion Park"으로 변환해주면 됨.
- 문자열 포맷팅은 자동화된 코드 등에서 변하는 값에 대해 출력을 할 때 주로 사용됨.

```
>>> x = "My␣name␣is␣%s"
>>> y = "Beomjin␣Park"
>>> x %(y)
'My␣name␣is␣Beomjin␣Park'
```

자료의 종류 XIII

- 앞선 예제에서 처럼 기존 "Beomjin Park"이 입력되어 있는 위치 대신에 %s를 입력한 것을 알 수 있고 %s 자리에 들어갈 값은 문자열 마지막에 %() 안에 입력하는 것을 알 수 있음.
- 대입하는 자료형에 따라서 대입할 자리에 있는 %s의 코드가 달라지는데 앞선 예제의 %s는 대입하는 자료형이 문자라는 것을 뜻함.
- 만약 대입하는 값이 정수형이라면 %d, 실수형이라면 %f를 사용함.

```
>>> x = "%d␣is␣a␣number"
>>> y = 1
>>> x %(y)
'1␣is␣a␣number'
```

자료의 종류 XIV

- 주의해야할 점은 정수형 포맷 코드인 %d를 사용한 후 입력하는 데이터가 실수형일 경우 강제로 정수형으로 바꾼뒤 출력함.
- 반대로 %f를 사용하고 정수를 입력하면 실수로 변환되어 출력됨.
- %d나 %f와 같이 숫자를 나타내는 코드를 사용한 후 문자를 입력하면 에러가 나지만 %s는 숫자형 자료를 입력해도 무방함.

```
>>> x = "%d␣is␣a␣number"
>>> y = 1.1
>>> x %(y)
'1␣is␣a␣number'

>>> x = "%f␣is␣a␣number"
>>> y = 1
>>> x %(y)
'1.000000␣is␣a␣number'
```

자료의 종류 XV

- 똑같은 방식으로 두개 이상의 값도 대입 가능함.

```
>>> x = "%d␣+␣%d␣=␣%d"
>>> x %(1, 2, 1 + 2)
'1␣+␣2␣=␣3'

>>> y = 1, 2, 1 + 2
>>> x %(y)
'1␣+␣2␣=␣3'
```

자료의 종류 XVI

- format 함수를 사용해 포맷팅을 수행할 수도 있음.
- 문자열에 대입이 필요한 곳에 {0}을 입력한 후 .format() 함수안에 값을 입력함.

```
>>> x = "My␣name␣is␣{0}"
>>> x.format("Beomjin␣Park")
'My␣name␣is␣Beomjin␣Park'
```

- format() 함수를 써서 여러개의 값을 대입할 수 있는데 이때는 문자열의 {0} 항목의 숫자를 0부터 숫자를 증가시켜서 표현하고 format함수에서 숫자에 해당하는 위치의 값이 해당 숫자에 대입됨.

```
>>> x = "{1}␣+␣{0}␣=␣{2}"
>>> x.format(2, 1, 3)
'1␣+␣2␣=␣3'
```

자료의 종류 XVII

- format 함수의 또 하나의 유용한 점은 이름으로 대입이 가능하다는 것.
- {0}, {1} 등 숫자 대신에 변수처럼 임의의 이름을 지정한 후, format 함수내에 해당 이름에 대입할 값을 입력해 대입할 수 있음.

```
>>> x = "My␣name␣is␣{name}"
>>> x.format(name = "Beomjin␣Park")
'My␣name␣is␣Beomjin␣Park'

>>> x = "{num1}␣+␣{num2}␣=␣{num3}"
>>> x.format(num1 = 1, num2 = 2, num3 = 1 + 2)
'1␣+␣2␣=␣3'
```

자료의 종류 XVIII

논리 자료형과 논리연산자

- 논리 자료형은 True와 False, 즉, 참과 거짓을 나타내는 값임.
- Python에서 논리 자료형은 True와 False로 표현 (큰따옴표나 작은 따옴표로 묶지 않음, T와 F는 대문자).
- 진리값은 나중에 설명할 조건문의 사용과 인덱싱에 자주 사용되기 때문에 중요함.
- 대표적으로 두 값 사이의 대소를 비교했을 때의 결과값이 진리값으로 표현됨.

```
# Truth value
# less than
>>> 2 < 3
True

# greater than
>>> 2 > 3
False
```

자료의 종류 XIX

```
# Equals
>>> 1 == (3 - 2)
True

>>> "Beomjin" == "Wonbin"
False

>>> "Beomjin" == "Beomjin"
True

# Not equals
>>> 1 != 3
True

>>> "Beomjin" != "Wonbin"
True
```

자료의 종류 XX

```
# less than or equal to
>>> (1 - 3) >= (-1 * 2)
True

>>> "Beomjin" >= "Wonbin"
False

# greater than or equal to
>>> (3 - 1) <= (4 / 2)
True

>>> "Beomjin" <= "Wonbin"
True
```

자료의 종류 XXI

- Python에서 진리값 True는 숫자 1과 같고 False는 숫자 0과 같음.

```
>>> 1 == True
True

>>> int(True)
1

>>> 0 == False
True

>>> int(False)
0
```

자료의 종류 XXII

- 진리값은 논리 연사자를 통해 연산이 가능함.

논리연산자	기호	설명
NOT	not	True는 False, False는 True로 나타냄
AND	and	양쪽이 모두 True면 True, 한쪽이 False면 False
OR	or	양쪽 중 한쪽만 True여도 True, 모두 False면 False

- Python에서 "&", "|", " $\sim$ ", " $\wedge$ "는 비트(bit) 연산자로 위의 연산자와 구별해야함.
- 비트는 이진수(예: 01001) 등을 뜻하고 이번 강의에서 다루지 않음.

자료의 종류 XXIII

```
>>> not True
False

>>> not (3 < 2)
True

>>> True and True
True

>>> True and False
False

>>> False or True
True

>>> (3 < 2) or (10 > 11)
False
```

객체의 종류 I

리스트(list)

- 리스트는 앞서 배운 여러 자료형들을 하나로 묶음으로 표현할 수 있는 객체.
- 숫자, 문자 등 혼용해서 리스트의 원소를 구성할 수 있고 리스트 안에 또 다른 리스트를 구성하는 중첩된 구조를 가질 수 있음.
- 리스트는 대괄호를 통해 생성할 수 있음.

```
>>> x = [1, "a", True, [1, "1"]]
>>> x
[1, 'a', True, [1, '1']]
```

객체의 종류 II

- list() 함수를 통해 다른 객체를 리스트로 변환할 수 있음.

```
>>> x = (1, 2, 3)
>>> list(x)
[1, 2, 3]
```

객체의 종류 III

- 리스트간의 "+" 연산자는 두개의 리스트를 합치고 "*" 연산자는 리스트를 반복해 결합함.

```
>>> x = ["a", "b"]
>>> y = ["c", "d"]
>>> x + y
['a', 'b', 'c', 'd']

>>> x * 3
['a', 'b', 'a', 'b', 'a', 'b']
```

객체의 종류 IV

튜플(tuple)

- 튜플은 리스트와 유사하지만 몇가지 다른점이 존재함.
- 튜플은 리스트와 마찬가지로 여러 요소들을 묶는 객체지만 한번 튜플로 묶이고 나면 안의 원소를 삭제, 수정 등 변경이 불가능함 (immutable).

```
>>> y = (1, "a", True, [1, "1"])
>>> y
(1, 'a', True, [1, '1'])

>>> y[0] = "b"
TypeError: 'tuple' object does not support item assignme
```

객체의 종류 V

- 튜플에 대해 "+"와 "*" 연산은 리스트와 같음.

```
>>> x = (1, 2, 3)
>>> y = (4, 5, 6)
>>> x + y
(1, 2, 3, 4, 5, 6)

>>> x * 2
(1, 2, 3, 1, 2, 3)
```

- tuple() 함수를 통해 다른 객체를 튜플로 변환할 수 있음.

```
>>> x = [1, 2, 3]
>>> tuple(x)
(1, 2, 3)
```

객체의 종류 VI

딕셔너리(dictionary)

- 딕셔너리는 "key" : "value"과 같은 형식으로 자료를 구성할 수 있는 자료형.
- 리스트나 튜플과 다르게 "key"와 "value"가 쌍을 이루기 때문에 "key"를 통해 "value"를 얻을 수 있음.
- 딕셔너리는 중괄호 ({})를 통해 생성이 가능함.
- key에 대응되는 value가 여러개의 값을 가질 때는 리스트나 튜플로 묶음.

```
>>> dic1 = {"name" : ["Park", "Kim", "Choi"]}
>>> dic1
{'name': ['Park', 'Kim', 'Choi']}
>>> dic2 = {"name" : ("Park", "Kim", "Choi")}
>>> dic2
{'name': ('Park', 'Kim', 'Choi')}
```

객체의 종류 VII

집합(set)

- 집합은 수학에서의 집합 특성을 가진 자료형.
- 집합에는 동일한 원소 (중복)을 허용하지 않으며 원소에 순서가 존재하지 않음.
- 중복을 허용하지 않기 때문에 중복을 제거하기 위한 용도로도 사용됨.
- 집합은 set() 함수를 이용해 입력값으로 리스트, 튜플을 입력하면 생성됨.

```
>>> set1 = set([1, 1, 2, 2, "c", "c"])
>>> set1
{1, 2, 'c'}
```

객체의 종류 VIII

- 합집합은 "|" 또는 "union" 함수, 교집합은 "&" 또는 "intersection" 메소드 그리고 차집합은 "-" 또는 "difference" 메소드로 가능.

```
>>> set2 = set([1, 2, 3, 4, "d"])
# Union
>>> set1 | set2
{1, 2, 'c', 3, 4, 'd'}
>>> set1.union(set2)
{1, 2, 'c', 3, 4, 'd'}

# Intersection
>>> set1 & set2
{1, 2}
>>> set1.intersection(set2)
{1, 2}
```

데이터의 종류 IX

```
# Difference
>>> set1 - set2
{'c'}
>>> set1.difference(set2)
{'c'}
```