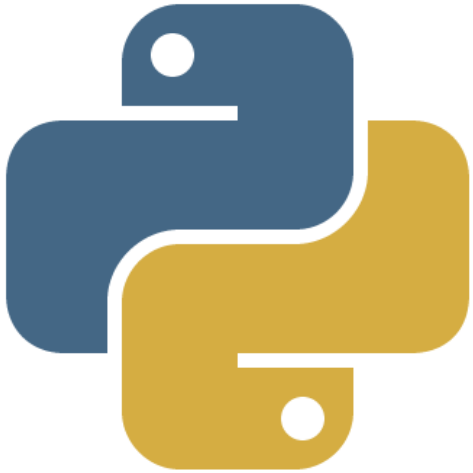
Programming in Python

02 – 자료형과 연산자



python

2016년 8월, 국민대학교 컴퓨터공학부

Python 객체 (Objects)

1. **프로그램**은 모듈들로 이루어지고,
   Programs are composed of modules.

2. **모듈**은 문장들을 담고 있다.
   Modules contain statements.

3. **문장**은 표현식을 포함하고 있으며,
   Statements contain expressions.

4. **표현식**은 객체를 생성하거나 처리한다.
   Expressions create and process objects.

Python 의 핵심 자료형 (Core Data Types)

Object type	Example literals/creation
수치 (Numbers)	1234, 3.1415, 3+4j, Decimal, Fraction
문자열 (Strings)	'spam', "guido's", b'a\x01c'
리스트 (Lists)	[1, [2, 'three'], 4]
사전 (Dictionaries)	{'food': 'spam', 'taste': 'yum'}
순서쌍 (Tuples)	(1, 'spam', 4, 'U')
파일 (Files)	myfile = open('eggs', 'r')
집합 (Sets)	set('abc'), {'a', 'b', 'c'}
기타 core type	Booleans, types, None
Program unit types	Functions, modules, classes
Implementation-related types	Compiled code, stack tracebacks

수치형과 연산자 (Operators)

연산자	의미	예제	결과
*	곱하기	2 * 3	6
/	나누기 (일반)	20 / 8	2.5
//	나누기 (정수)	20 // 8	2
%	나머지	20 % 8	4
+	더하기	2 + 3	5
-	빼기	7 - 3	4

```
>>> print("111을 4로 나누면? ", 111 / 4)
111을 4로 나누면? 27.75
>>> print("11을 4로 나누면? ", 11 / 4)
11을 4로 나누면? 2.75
```

Python 2.7

```
>>> 20 / 8
2
>>> 20 / 8.0
2.5
```

문자열 (Strings)

```
>>> print("국민대학교\n컴퓨터공학부")
국민대학교
컴퓨터공학부
>>> a = "국민대학교\n컴퓨터공학부"
>>> print a
국민대학교
컴퓨터공학부
>>> a
???
```

특수 문자열	내용
\n	문자열의 줄바꿈 (개행)
\t	탭
\\	역슬래시
\"	인용부호

Python 에서는 문자열을 표현하기 위해 큰따옴표 (") 와 작은 따옴표 (') 를 모두 이용할 수 있음
(문자열을 여닫는 데 사용한 것과 다른 따옴표는 escape 하지 않아도 됨 - 그러나?)

print('I say "High", you say "Low". You say "Why?" and I say "I don\'t know". Oh no.')

문자열 인덱싱 (Indexing)

```
>>> S = 'Spam'
>>> len(S)              # 문자열의 길이를 파악
4
>>> S[0]                # S 를 구성하는 첫번째 원소 (문자) - 포지션은 0 부터 시작함
'S'
>>> S[1]                # 왼쪽으로부터 두번째 위치하는 원소 (문자)
'p'
>>> S[-1]               # S 를 구성하는 마지막 문자 (가장 오른쪽)
'm'
>>> S[-2]               # 마지막으로부터 두번째 위치하는 원소 (문자)
'a'
>>> S[len(S) - 1]       # 어떻게?
???
```

문자열 슬라이싱 (Slicing)

```
>>> S = 'Spam'
>>> S[1:3]              # 포지션 1 부터 시작하여 2 까지 (3 미만) 로 이루어진 S 의 슬라이스
'pa'
>>> S[1:]               # 끝 인덱스를 명시하지 않으면 마지막까지
'pam'
>>> S[0:3]              # 처음부터 시작해서 3 미만까지
'Spa'
>>> S[:3]               # 시작을 명시하지 않으면 처음부터 (S[0:3] 와 동일)
'Spa'
>>> S[:-1]              # 끝 인덱스를 마지막 문자 하나를 제외하는 것으로 지정
'Spa'
>>> S[:]                # 범위는 [0:len(S)] 와 동일 (len(S) == 4)
'Spam'
```

문자열 연산

```
>>> S = 'Spam'
>>> S.find('pa')          # 처음으로 등장하는 substring 의 오프셋 (인덱스)
1
>>> S.find('xy')          # 발견되지 않으면?
-1
>>> S.replace('pa', 'XYZ')          # 'pa' 를 찾아 다른 문자열로 대체
'SXYZm'
>>> S                     # 하지만 S 는 변화하지 않음
'Spam'
>>> S.upper()             # 포함된 모든 문자를 대문자로
'SPAM'
>>> S.lower()             # 포함된 모든 문자를 소문자로
'spam'
>>> S                     # 하지만 S 는 변화하지 않음
'Spam'
>>> S.isalpha()           # 알파벳으로 이루어져 있는지? (그렇다면 isdigit() 는?)
True
```

문자열 연산

```
>>> line = 'aaa,bbb,ccccc,dd'
>>> x = line.split(',')    # ',' 를 구분자 (delimiter) 로 하여 문자열로부터 substring 의 리스트를 생성
>>> x
['aaa', 'bbb', 'ccccc', 'dd']

>>> ','.join(x)
'aaa,bbb,ccccc,dd'

>>> line = 'aaa,bbb,ccccc,dd\n'
>>> line = line.rstrip()                # 문자열의 오른쪽 끝에서 공백 문자나 개행 문자를 제거
>>> line
'aaa,bbb,ccccc,dd'

>>> line = '    abcd'
>>> line.lstrip()                       # 문자열의 왼쪽 끝에서 공백 문자나 개행 문자를 제거
'abcd'
```

리스트 (Lists)

```
>>> L = [123, 'spam', 1.23]        # 서로 다른 타입의 객체들이 리스트를 구성할 수 있음
>>> len(L)
3

>>> L[0]                           # 포지션을 이용한 인덱싱
123

>>> L[:-1]                         # 슬라이싱 - 새 리스트가 생성되어 리턴됨
[123, 'spam']

>>> L + [4, 5, 6]                  # 병합의 결과도 새로운 리스트가 생성됨
[123, 'spam', 1.23, 4, 5, 6]

>>> L                              # 위의 결과로 원래의 리스트는 아무런 영향을 받지 않음
[123, 'spam', 1.23]
```

리스트 연산

```
>>> L = [123, 'spam', 1.23]
>>> L.append('NI')                # 리스트에 새로운 원소를 덧붙임 - 원래 리스트가 자라남
>>> L
[123, 'spam', 1.23, 'NI']

>>> L.pop(2)                      # 포지션을 이용하여 원소를 뽑아냄 - 원래 리스트가 줄어듬
1.23
>>> L
[123, 'spam', 'NI']

>>> M = ['bb', 'aa', 'cc']
>>> M.sort()                      # 리스트를 정렬 - 리스트는 변화함
>>> M
['aa', 'bb', 'cc']
>>> M.reverse()                   # 리스를 역순으로 만듬
>>> M
['cc', 'bb', 'aa']
```

중첩된 리스트 (Nested Lists)

```
>>> M = [ [1, 2, 3],                # 3 x 3 행렬과 유사한 구조로 리스트를 구성
          [4, 5, 6],                # [ ] 에 들어있으므로 여러 라인에 걸쳐 쓸 수 있음
          [7, 8, 9] ]
>>> M
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]

>>> M[1]                          # M[1], 즉 두번째 원소는 리스트
[4, 5, 6]

>>> M[1][2]                       # 두 번의 인덱스를 통하여 행렬의 원소를 얻어낼 수 있음
6
```

리스트 내포 (List Comprehension)

```
>>> M = [ [1, 2, 3],
          [4, 5, 6],
          [7, 8, 9] ]
>>> col2 = [row[1] for row in M]              # 두 번째 열에 있는 원소들을 모아 리스트를 만듬
>>> col2
[2, 5, 8]

>>> M                                         # 원래의 리스트는 변화하지 않음
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]

>>> [row[1] + 1 for row in M]
[3, 6, 9]

>>> [row[1] for row in M if row[1] % 2 == 0]          # if 를 이용하여 홀수인 원소들을 필터링
[2, 8]
```

사전 (Dictionaries)

```
>>> D = {'food': 'Spam', 'quantity': 4, 'color': 'pink'}
>>> D['food']                    # 키에 의한 인덱싱
'Spam'

>>> D['quantity'] += 1           # 원래의 사전 내용이 변화함
>>> D
{'food': 'Spam', 'color': 'pink', 'quantity': 5}

>>> D = {}                       # 빈 사전을 초기화하여
>>> D['name'] = 'Bob'            # 키를 생성하고 그에 해당하는 값을 할당
>>> D['job'] = 'dev'
>>> D['age'] = 40
>>> D
{'age': 40, 'job': 'dev', 'name': 'Bob'}

>>> print(D['name'])
Bob
```

중첩된 사전 (Nested Dictionaries)

```
>>> rec = {'name': {'first': 'Bob', 'last': 'Smith'},
           'job': ['dev', 'mgr'],
           'age': 40.5}

>>> rec['name']                    # 'name' 은 중첩된 사전임
{'last': 'Smith', 'first': 'Bob'}
>>> rec['name']['last']            # 중첩된 사전에 대하여 또다시 인덱싱
'Smith'

>>> rec['job']                     # 'job' 은 중첩된 리스트
['dev', 'mgr']
>>> rec['job'][-1]                 # 중첩된 리스트에 대하여 인덱싱 (마지막 원소)
'mgr'

>>> rec['job'].append('janitor')   # 중첩된 리스트를 변경
>>> rec
{'age': 40.5, 'job': ['dev', 'mgr', 'janitor'], 'name': {'last': 'Smith', 'first': ' Bob'}}
```

순서쌍 (Tuples)

```
>>> T = (1, 2, 3, 4)                # 순서쌍을 생성
>>> len(T)
4

>>> T + (5, 6)                      # 순서쌍의 병합
(1, 2, 3, 4, 5, 6)

>>> T                               # 원래의 순서쌍에는 변화 없음
(1, 2, 3, 4)

>>> T[0]                            # 인덱싱
1

>>> T.index(4)                      # 특정 원소의 오프셋 (인덱스) 를 찾기
3
>>> T.count(4)                      # 특정 원소의 출현 회수
1
```

순서쌍 (Tuples)

```
>>> T = (1, 2, 3, 4)
>>> T[0] = 2                                  # 순서쌍의 원소는 변경 불가
TypeError: 'tuple' object does not support item assignment

>>> T = ('spam', 3.0, [11, 22, 33])          # 자료형이 다른 원소들로 이루어진 순서쌍
>>> T[1]
3.0

>>> T[2][1]                                   # 중첩된 리스트에 대한 인덱싱
22

>>> T.append(4)
AttributeError: 'tuple' object has no attribute 'append'
```

Quiz

- L = [1, 2, 3, 4] 일 때, L + [5, 6] 의 결과는?
  - 그 후, L 은 어떤 데이터를 가지고 있는가?
- L = [1, 2, 3, 4] 일 때, L.append(5), L.append(6) 의 결과는?
  - 그 후 L 은 어떤 데이터를 가지고 있는가?
- L = [1, 2, 3, 4] 일 때, L.append([5, 6]) 의 결과는?
  - 그 후 L 은 어떤 데이터를 가지고 있는가?
- L = {1: 'a', 2: 'b', 3: 'c'} 일 때, [x for x in L] 의 결과는?
  - 자료형은 무엇이며, 데이터 내용은 무엇인지?

Q & A