파이썬스타일 코드

Python Programming

국민대학교 경영대학원 AI빅데이터전공
문현실(hsmoon@kookmin.ac.kr)

Ch.01 파이썬 스타일 코드 I

파이썬 코딩 규칙(coding convention) 표준

- 들여쓰기는 4스페이스
- 한 줄은 최대 79자까지
- 불필요한 공백은 피함
- = 연산자는 1칸 이상 띄우지 않는다.
- 주석은 항상 갱신하고, 불필요한 주석은 삭제한다.
- 소문자 l, 대문자 O, 대문자 I는 사용을 금지한다.
- 함수명은 소문자로 구성하고, 필요하면 밑줄로 나눈다.

파이썬 스타일 코드의 개념

- 파이썬 스타일 코드(Pythonic Code)
  - 초기 파이썬은 기존 C나 자바와 비교해 훨씬 가볍고 간단한문법
  - 사람이 코드를 쉽게 이해하고 사용하기 쉬운 언어
  - 최근 들어 프로그래밍 언어 대부분이 파이썬의 간단한 문법적 특징을 도입

```
>>> colors = ['red', 'blue', 'green', 'yellow']
>>> result = ''
>>> for s in colors:
...     result += s
...
>>> print(result)
redbluegreenyellow
```

```
>>> colors = ['red', 'blue', 'green', 'yellow']
>>> result = ''.join(colors)
>>> print(result)
redbluegreenyellow
```

파이썬 스타일 코드의 사용 이유

- 파이썬의 철학
  - 인간의 시간이 컴퓨터의 시간보다 더 중요하다
  - 코드상으로 사람이 해야 하는 일을 최대한 줄이면서 목표를 달성할 수 있는 문법 체계
- 유명한 오픈소스부터 다양한 예제코드까지 파이썬 스타일 코드로 작성
  - 다른 사람이 작성한 코드를 쉽게 이해
- 효율적인 프로그래밍 측면
  - 코드 자체가 간결해지고 작성 시간도 줄임

문자열의 분리: split() 함수

- 특정 값을 기준으로 리스트 형태로 변환하는 방법
  - 매개변수가 없을 경우에는 공백을 기준으로 변환

```
>>> items = 'zero one two three'.split()          # 빈칸을 기준으로 문자열 분리하기
>>> print (items)
['zero', 'one', 'two', 'three']
```

문자열의 분리: split() 함수

- split() 활용 예제

```
>>> example = 'python,jquery,javascript'          # ","를 기준으로 문자열 나누기
>>> example.split(",")
['python', 'jquery', 'javascript']
>>> a, b, c = example.split(",")                  # 리스트에 있는 각 값을 a, b, c 변수로 언패킹
>>> print(a, b, c)
python jquery javascript
>>> example = 'theteamlab.univ.edu'
>>> subdomain, domain, tld = example.split('.')   # "."을 기준으로 문자열 나누기 → 언패킹
>>> print(subdomain, domain, tld)
theteamlab univ edu
```

문자열의 결합: join()함수

- 문자열로 구성된 리스트를 합쳐 하나의 문자열로 반환
  - 구분자.join(리스트)

```
>>> colors = ['red', 'blue', 'green', 'yellow']
>>> result = ''.join(colors)
>>> result
'redbluegreenyellow'
```

문자열의 결합: join()함수

- join() 활용 예제

```
>>> result = ' '.join(colors)              # 연결 시, 1칸을 띄고 연결
>>> result
'red blue green yellow'
>>> result = ', '.join(colors)             # 연결 시 ", "으로 연결
>>> result
'red, blue, green, yellow'
>>> result = '-'.join(colors)              # 연결 시 "-"으로 연결
>>> result
'red-blue-green-yellow'
```

문자열 formatting

- %서식
  - '%자료형' % 값
  - 자료형이 맞지 않으면 에러
  - 키워드 사용도 가능 : 딕셔너리로 값을 전달
  - 여러 자료형을 사용할 경우 순서대로 튜플로 값을 전달

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

문자열 formatting

- format() 메서드
  - %서식과 유사하나 문자열 형태가 있는 함수를 사용한다는 차이점
  - “{자료형}”.format(인수)
  - 자료형을 바로 지정해 주지 않고 순서대로 변수가 할당
  - 사용 방법
    - 중괄호만 사용 : 순서대로 .format()에 인수 입력
    - 중괄호내에 튜플 주소 입력 : 튜플 주소대로 인수가 나타남
    - 키워드 사용 : 인수에 키워드=인수 형태로 입력
  - 주의사항
    - 튜플 주소 입력과 중괄호만 사용하는 방식은 함께 사용 불가능
    - 키워드 사용하는 경우에는 키워드가 항상 마지막에 위치

리스트 컴프리헨션(List Comprehension)

- 기존 리스트형을 사용하여 간단하게 새로운 리스트를 만드는 기법
  - 포괄형 리스트, 포함형 리스트, 지능형 리스트, 축약형 리스트
  - 반복문을 한줄에 사용

```
>>> result = []
>>> for i in range(10):
...     result.append(i)
...
>>> result
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
>>> result = [i for i in range(10)]
>>> result
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

필터링

- if문과 함께 사용하여 코드를 축약

```
>>> result = []
>>> for i in range(10):
...     if i % 2 == 0:
...         result.append(i)
...
>>> result
[0, 2, 4, 6, 8]
```

```
>>> result = [i for i in range(10) if i % 2 == 0]
>>> result
[0, 2, 4, 6, 8]
```

- else문과 함께 사용도 가능
  - 조건을 만족하지 않을 때 else 뒤에 값을 할당

```
>>> result = [i if i % 2 == 0 else 10 for i in range(10)]
>>> result
[0, 10, 2, 10, 4, 10, 6, 10, 8, 10]
```

리스트값에 인덱스를 붙여 출력

- enumerate() 함수
  - 딕셔너리 형태로 출력되어 인덱스를 키로 사용

```
>>> for i, v in enumerate(['tic', 'tac', 'toe']):    # 리스트에 있는 인덱스와 값을 언패킹
...     print(i, v)
...
0 tic
1 tac
2 toe
```

리스트 값을 병렬로 묶어 출력

- zip()함수
  - 1 개 이상의 리스트 값이 같은 인덱스에 있을 때 병렬로 묶는 함수

```
>>> alist = ['a1', 'a2', 'a3']
>>> blist = ['b1', 'b2', 'b3']
>>> for a, b in zip(alist, blist):                    # 병렬로 값을 추출
...     print(a, b)
...
a1 b1
a2 b2
a3 b3
```

리스트 값을 병렬로 묶어 출력

- zip()함수의 활용
  - 같은 위치에 있는 값끼리의 덧셈

```
>>> a, b, c = zip((1, 2, 3), (10, 20, 30), (100, 200, 300))
>>> print (a, b, c)
(1, 10, 100) (2, 20, 200) (3, 30, 300)
>>> [sum(x) for x in zip((1, 2, 3), (10, 20, 30), (100, 200, 300))]
[111, 222, 333]
```

enumerate()와 zip()을 동시에 사용

- 인덱스와 병렬 결합의 결과를 동시에 출력

```
>>> alist = ['a1', 'a2', 'a3']
>>> blist = ['b1', 'b2', 'b3']
>>> for i, (a, b) in enumerate(zip(alist, blist)):
...     print (i, a, b)                    # (인덱스, alist[인덱스], blist[인덱스]) 표시
...
0 a1 b1
1 a2 b2
2 a3 b3
```

Ch.02 파이썬 스타일 코드 II

람다(lambda) 함수

- 함수의 이름없이 함수처럼 사용할 수 있는 익명의 함수

```
1 def f(x, y):
2     return x + y
3
4 print(f(1, 4))
```

```
5
```

```
1 f = lambda x, y: x + y
2 print(f(1, 4))
```

```
5
```

```
print((lambda x: x + 1)(5))
```

람다(lambda) 함수

- 람다 함수의 다양한 형태
  - 파이썬 3.x 버전부터는 람다 함수의 사용을 권장하지 않음

```
>>> f = lambda x, y: x + y
>>> f(1, 4)
5
>>>
>>> f = lambda x: x ** 2
>>> f(3)
9
>>>
>>> f = lambda x: x / 2
>>> f(3)
1.5
>>> f(3, 5)
Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
TypeError: <lambda>() takes 1 positional argument but 2 were given
```

파이썬에서 함수 인자를 사용하는 방법

- 키워드 인자
  - 함수의 인터페이스에 지정된 변수명을 사용하여 함수의 인자를 지정
- 디폴트 인자
  - 별도의 인자값이 입력되지 않을 때, 인터페이스 선언에서 지정한 초깃값을 사용
- 가변 인자
  - 함수의 인터페이스에 지정된 변수 이외의 추가 변수를 함수에 입력할 수 있게 지원
- 키워드 가변 인자
  - 매개변수의 이름을 따로 지정하지 않고 입력하는 방법

키워드 인자

- 매개변수의 변수명을 사용하여 함수의 인자를 지정
  - 인자 순서대로 입력
  - 입력 변수명과 함께 입력 : 순서가 상관없음

디폴트 인자

- 매개변수에 기본값을 지정
  - 아무런 값도 인자로 넘어오지 않으면 지정된 기본값 사용

가변 인자

- 매개변수 개수가 정해지지 않은 경우에 사용
  - * (asterisk) 는 파이썬에서 가변인자를 표현하는 방식
  - 가변 인자는 일반적으로 *args을 사용하여 표현
  - *args에 들어온 인자는 튜플(tuple) 형태로 묶임
  - 가변 인자는 반드시 일반적인 키워드 인자가 모두 끝난 뒤에 사용

키워드 가변 인자

- 가변 인자는 변수의 순서대로 튜플 형태로 저장
  - 변수의 이름을 지정할 수 없다는 단점이 존재
- ** 를 사용하여 함수의 매개변수를 표시
  - 입력된 값은 딕셔너리 형태로 묶임
- 키워드 가변 인자는 반드시 모든 매개 변수의 맨 마지막에 선언

별표(*)의 활용

- 함수의 가변인자 사용 예제

```
>>> def asterisk_test(a, *args):
...     print(a, args)
...     print(type(args))
...
>>> asterisk_test(1, 2, 3, 4, 5, 6)
1 (2, 3, 4, 5, 6)
<class 'tuple'>
```

```
>>> def asterisk_test(a, **kargs):
...     print(a, kargs)
...     print(type(kargs))
...
>>> asterisk_test(1, b=2, c=3, d=4, e=5, f=6)
1 {'b': 2, 'c': 3, 'd': 4, 'e': 5, 'f': 6}
<class 'dict'>
```

별표(*)의 활용

- 여러 개의 변수를 담는 컨테이너로서의 속성을 부여
- 별표의 언패킹

```
>>> def asterisk_test(a, args):
...     print(a, *args)
...     print(type(args))
...
>>> asterisk_test(1, (2, 3, 4, 5, 6))
1 2 3 4 5 6
<class 'tuple'>
```

```
1 (2 3 4 5 6)
```

별표(*)의 활용

- 여러 개의 변수를 담는 컨테이너로서의 속성을 부여
- 별표의 언패킹

```
>>> def asterisk_test(a, *args):
...     print(a, args)
...     print(type(args))
...
>>> asterisk_test(1, *(2, 3, 4, 5, 6))
1 (2, 3, 4, 5, 6)
<class 'tuple'>
```

```
>>> asterisk_test(1, 2, 3, 4, 5, 6)
```

별표(*)의 활용

- 여러 개의 변수를 담는 컨테이너로서의 속성을 부여
- 별표의 언패킹

```
>>> a, b, c = ([1, 2], [3, 4], [5, 6])
>>> print(a, b, c)
[1, 2] [3, 4] [5, 6]
>>>
>>> data = ([1, 2], [3, 4], [5, 6])
>>> print(*data)
[1, 2] [3, 4] [5, 6]
```

별표(*)의 활용

- 여러 개의 변수를 담는 컨테이너로서의 속성을 부여
- 별표의 언패킹
  - zip()과 함께 사용

```
>>> for data in zip(*[[1, 2], [3, 4], [5, 6]]):
        print(data)
        print(type(data))

(1, 3, 5)
<class 'tuple'>
(2, 4, 6)
<class 'tuple'>
```

별표(*)의 활용

- 두개의 별표(*)를 사용할 경우 딕셔너리형을 언패킹

```
>>> def asterisk_test(a, b, c, d,):
        print(a, b, c, d)

>>> data = {"b":1 , "c":2, "d":3}
>>> asterisk_test(10, **data)
10 1 2 3
```

Q&A

Thank you for your listening