Pythonic code

TEAMLAB director
최성철

파이썬 스타일 코드
Pythonic Code

Overview

- 파이썬 스타일의 코딩 기법
- 파이썬 특유의 문법을 활용하여 효율적으로 코드를 표현함
- 그러나 더 이상 파이썬 특유는 아님, 많은 언어들이 서로의 장점을 채용
- 고급 코드를 작성 할 수록 더 많이 필요해짐

예시: 여러 단어들을 하나로 붙일 때

```
>>> colors = ['red', 'blue', 'green', 'yellow']
>>> result = ''
>>> for s in colors:
     result += s
```

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

Contents

- split & join
- list comprehension
- enumerate & zip
- lambda & map & reduce
- generator
- asterisk

Why Pythonic Code?

남 코드에 대한 이해도

많은 개발자들이 python 스타일로 코딩한다

효율

단순 for loop append보다 list가 조금 더 빠르다

익숙해지면 코드도 짧아진다.

간지

쓰면 왠지 코드 잘 짜는 거처럼 보인다

Split & join

split 함수

- string type의 값을 "기준값"으로 나눠서 List 형태로 변환

```
>>> items = 'zero one two three'.split()  # 빈칸을 기준으로 문자열 나누기
>>> print (items)
['zero', 'one', 'two', 'three']
>>> example = 'python,java,javascript'  # ","을 기준으로 문자열 나누기
>>> example.split(",")
['python', ' java', 'javascript']
>>> a, b, c = example.split(",")
# 리스트에 있는 각 값을 a,b,c 변수로 unpacking
>>> example = ' teamlab.technology.io'
>>> subdomain, domain, tld = example.split('.')
# "."을 기준으로 문자열 나누기 → Unpacking
```

join 함수

- String으로 구성된 list를 합쳐 하나의 string으로 반환

```
>>> colors = ['red', 'blue', 'green', 'yellow']
>>> result = ''.join(colors)
>>> result
'redbluegreenyellow'
>>> result = ' '.join(colors) # 연결 시 빈칸 1칸으로 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'
```

list comprehension

Overview

- 기존 List 사용하여 간단히 다른 List를 만드는 기법
- 포괄적인 List, 포함되는 리스트라는 의미로 사용됨
- 파이썬에서 가장 많이 사용되는 기법 중 하나
- 일반적으로 for + append 보다 속도가 빠름

Examples (1/4)


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

General Style

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

List Comprehension

Examples (2/4)


```
>>> word_1 = "Hello"
>>> word_2 = "World"
>>> result = [i+j for i in word_1 for j in word_2]
    # Nested For loop
>>> result
['HW', 'Ho', 'Hr', 'Hl', 'Hd', 'eW', 'eo', 'er',
'el', 'ed', 'lW', 'lo', 'lr', 'll', 'ld', 'lW',
'lo', 'lr', 'll', 'ld', 'oW', 'oo', 'or', 'ol', 'od']
```

Examples (3/4)


```
>>> case_1 = ["A","B","C"]
>>> case_2 = ["D","E","A"]
>>> result = [i+j for i in case_1 for j in case_2]
>>> result
['AD', 'AE', 'AA', 'BD', 'BE', 'BA', 'CD', 'CE', 'CA']
>>> result = [i+j for i in case_1 for j in case_2 if not(i==j)]
# Filter: i랑 j과 같다면 List에 추가하지 않음
# [i+j if not(i==j) else i for i in case_1 for j in case_2]
>>> result
['AD', 'AE', 'BD', 'BE', 'BA', 'CD', 'CE', 'CA']
>>> result.sort()
>>> result
['AD', 'AE', 'BA', 'BD', 'BE', 'CA', 'CD', 'CE']
```

Examples (4/4)


```
>>> words = 'The quick brown fox jumps over
the lazy dog'.split()
# 문장을 빈칸 기준으로 나눠 list로 변환

>>> print (words)
['The', 'quick', 'brown', 'fox', 'jumps',
'over', 'the', 'lazy', 'dog']
>>>
>>> stuff = [[w.upper(), w.lower(), len(w)]
for w in words]

# list의 각 elemente들을 대문자, 소문자, 길이로 변
환하여 two dimensional list로 변환
```

```
>>> for i in stuff:
...     print (i)
...
['THE', 'the', 3]
['QUICK', 'quick', 5]
['BROWN', 'brown', 5]
['FOX', 'fox', 3]
['JUMPS', 'jumps', 5]
['OVER', 'over', 4]
['THE', 'the', 3]
['LAZY', 'lazy', 4]
['DOG', 'dog', 3]
```

Two dimensional vs One dimensional


```
>>> case_1 = ["A","B","C"]
>>> case_2 = ["D","E","A"]
['AD', 'AE', 'AA', 'BD', 'BE', 'BA', 'CD', 'CE', 'CA']
>>> result = [i+j for i in case_1 for j in case_2]
>>> result
['AD', 'AE', 'AA', 'BD', 'BE', 'BA', 'CD', 'CE', 'CA']
>>> result = [ [i+j for i in case_1] for j in case_2]
>>> result
[['AD', 'BD', 'CD'], ['AE', 'BE', 'CE'], ['AA', 'BA', 'CA']]
```

enumerate & zip

Example

- enumerate : list의 element를 추출할 때 번호를 붙여서 추출

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

>>> mylist = ['a', 'b', 'c', 'd']
>>> list(enumerate(mylist)) # list의 있는 index와 값을 unpacking하여 list로 저장
[(0, 'a'), (1, 'b'), (2, 'c'), (3, 'd')]
>>> {i:j for i,j in enumerate('Artificial intelligence (AI), is intelligence demonstrated by machines,
unlike the natural intelligence displayed by humans and animals.'.split())}
# 문장을 list로 만들고 list의 index와 값을 unpacking하여 dict로 저장
{0: 'Artificial', 1: 'intelligence', 2: '(AI),', 3: 'is', 4: 'intelligence', 5: 'demonstrated', 6: 'by',
7: 'machines,', 8: 'unlike', 9: 'the', 10: 'natural', 11: 'intelligence', 12: 'displayed', 13: 'by', 14:
'humans', 15: 'and', 16: 'animals.'}
```

Example

- zip : 두 개의 list의 값을 병렬적으로 추출함

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

>>> a,b,c =zip((1,2,3),(10,20,30),(100,200,300)) #각 tuple의 같은 index 끼리 묶음
(1, 10, 100) (2, 20, 200) (3, 30, 300)

>>> [sum(x) for x in zip((1,2,3), (10,20,30), (100,200,300))]
# 각 Tuple 같은 index를 묶어 합을 list로 변환
[111, 222, 333]
```

Example

- enumerate & zip 동시 사용 용례

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

lambda & map & reduce

Overview

- 함수 이름 없이, 함수처럼 쓸 수 있는 익명함수

- 수학의 람다 대수에서 유래함

General function

```
def f(x, y):
    return x + y

print(f(1, 4))
```

Lambda function

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

Overview

- Python 3부터는 권장하지는 않으나 여전히 많이 쓰임

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

```
f = lambda x: x ** 2
print(f(3))
```

```
f = lambda x: x / 2
print(f(3))
```

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

- Always use a def statement instead of an assignment statement that binds a lambda expression directly to an identifier.

Yes:

PEP 8에서는 lambda의 사용을 권장하지 않음

```
def f(x): return 2*x
```

No:

```
f = lambda x: 2*x
```

The first form means that the name of the resulting function object is specifically 'f' instead of the generic '<lambda>'. This is more useful for tracebacks and string representations in general. The use of the assignment statement eliminates the sole benefit a lambda expression can offer over an explicit def statement (i.e. that it can be embedded inside a larger expression)

https://www.python.org/dev/peps/pep-0008/

lambda problmes

- 어려운 문법
- 테스트의 어려움
- 문서화 docstring 지원 미비
- 코드 해석의 어려움
- 이름이 존재하지 않는 함수의 출현
- 그래도 많이 쓴다...

https://dev.to/lachlaneagling/python-lambda-functions-what-the-56io

map function overview

- 두 개 이상의 list에도 적용 가능함, if filter도 사용가능

```
ex = [1,2,3,4,5]
f = lambda x, y: x + y
print(list(map(f, ex, ex)))
```

```
list(
    map(
    lambda x: x ** 2 if x % 2 == 0
    else x,
    ex)
)
```

map function overview

- python3 는 iteration을 생성 → list을 붙여줘야 list 사용가능

- 실행시점의 값을 생성, 메모리 효율적

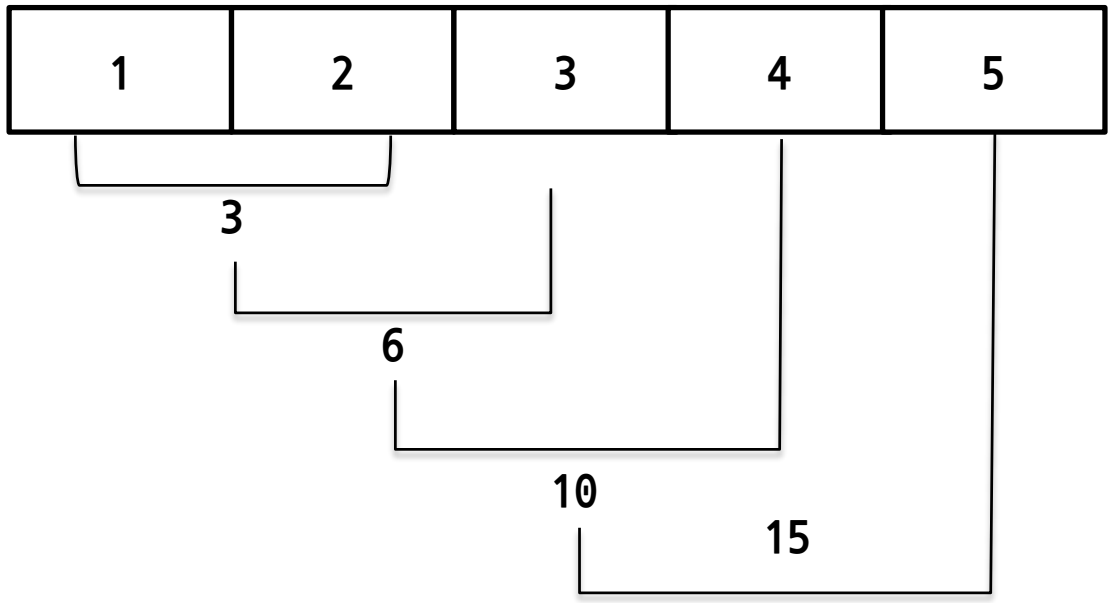
```
ex = [1,2,3,4,5]
print(list(map(lambda x: x+x, ex)))
print((map(lambda x: x+x, ex)))

f = lambda x: x ** 2
print(map(f, ex))
for i in map(f, ex):
    print(i)
```

```
result = map(f, ex)
print(next(result))
```

reduce function overview

- map function과 달리 list에 똑같은 함수를 적용해서 통합

```
from functools import reduce
print(reduce(lambda x, y: x+y, [1, 2, 3, 4, 5]))
```



Summary

- Lambda, map, reduce는 간단한 코드로 다양한 기능을 제공

- 그러나 코드의 직관성이 떨어져서 lambda나 reduce는 python3에서 사용을 권장하지 않음

- Legacy library나 다양한 머신러닝 코드에서 여전히 사용중

iterable object

Overview

- Sequence형 자료형에서 데이터를 순서대로 추출하는 object

```
for city in ["Seoul", "Busan", "Pohang"]:
    print(city, end="\t")
for language in ("Python", "C", "Java"):
    print(language, end="\t")
for char in "Python is easy":
    print(char, end = " ")
```

Characteristics

- 내부적 구현으로 __iter__ 와 __next__ 가 사용됨

- iter() 와 next() 함수로 iterable 객체를 iterator object로 사용

```
cities = ["Seoul", "Busan", "Jeju"]

iter_obj = iter(cities)

print(next(iter_obj))
print(next(iter_obj))
print(next(iter_obj))
next(iter_obj)
```

generator

Overview

- iterable object를 특수한 형태로 사용해주는 함수
- element가 사용되는 시점에 값을 메모리에 반환
  : yield를 사용해 한번에 하나의 element만 반환함

```
def general_list(value):
    result = []
    for i in range(value):
        result.append(i)
    return result
```

```
def geneartor_list(value):
    result = []
    for i in range(value):
        yield i
```

generator comprehension

- list comprehension과 유사한 형태로 generator형태의 list 생성
- generator expression 이라는 이름으로도 부름
- [ ] 대신 ( ) 를 사용하여 표현

```
gen_ex = (n*n for n in range(500))
print(type(g))
```

Why generator

- 일반적인 iterator는 generator에 반해 훨씬 큰 메모리 용량 사용

```
from sys import getsizeof
gen_ex = (n*n for n in range(500))
print(getsizeof(gen_ex))
print(getsizeof(list(gen_ex)))
list_ex = [n*n for n in range(500)]
print(getsizeof(list_ex))
```

When generator

- list 타입의 데이터를 반환해주는 함수는 generator로 만들어라!
  : 읽기 쉬운 장점, 중간 과정에서 loop 이 중단될 수 있을 때!
- 큰 데이터를 처리할 때는 generator expression을 고려하라!
  : 데이터가 커도 처리의 어려움이 없음
- 파일 데이터를 처리할 때도 generator를 쓰자

function passing arguments

Overview

- 함수에 입력되는 arguments의 다양한 형태

1) Keyword arguments

2) Default arguments

3) Variable-length arguments

Keyword arguments

- 함수에 입력되는 parameter의 변수명을 사용, arguments를 넘김

```
def print_somthing(my_name, your_name):
    print("Hello {0}, My name is {1}".format(your_name, my_name))


print_somthing("Sungchul", "TEAMLAB")
print_somthing(your_name="TEAMLAB", my_name="Sungchul")
```

Default arguments

- parameter의 기본 값을 사용, 입력하지 않을 경우 기본값 출력

```
def print_somthing_2(my_name, your_name="TEAMLAB"):
    print("Hello {0}, My name is {1}".format(your_name, my_name))


print_somthing_2("Sungchul", "TEAMLAB")
print_somthing_2("Sungchul")
```

variable-length asterisk

함수의 parameter가 정해지지 않았다?

다항 방정식? 마트 물건 계산 함수?

가변인자 using asterisk

가변인자 (variable-length)

- 개수가 정해지지 않은 변수를 함수의 parameter로 사용하는 법
- Keyword arguments와 함께, argument 추가가 가능
- Asterisk(*) 기호를 사용하여 함수의 parameter를 표시함
- 입력된 값은 tuple type으로 사용할 수 있음
- 가변인자는 오직 한 개만 맨 마지막 parameter 위치에 사용가능

가변인자 (variable-length)

- 가변인자는 일반적으로 *args를 변수명으로 사용
- 기존 parameter 이후에 나오는 값을 tuple로 저장함

```
def asterisk_test(a, b, *args):
    return a+b+sum(args)

print(asterisk_test(1, 2, 3, 4, 5))
```

가변인자 (variable-length)

- 가변인자는 일반적으로 *args를 변수명으로 사용
- 기존 parameter 이후에 나오는 값을 tuple로 저장함

```
def asterisk_test_2(*args):
    x, y, z = args
    return x, y, z

print(asterisk_test_2(3, 4, 5))
```

키워드 가변인자 (Keyword variable-length)

- Parameter 이름을 따로 지정하지 않고 입력하는 방법
- asterisk(*) 두개를 사용하여 함수의 parameter를 표시함
- 입력된 값은 dict type으로 사용할 수 있음
- 가변인자는 오직 한 개만 기존 가변인자 다음에 사용

키워드 가변인자 예제


```python
def kwargs_test_1(**kwargs):
    print(kwargs)


def kwargs_test_2(**kwargs):
    print(kwargs)
    print("First value is {first}".format(**kwargs))
    print("Second value is {second}".format(**kwargs))
    print("Third value is {third}".format(**kwargs))
```

키워드 가변인자 예제


```
def kwargs_test_3(one,two, *args, **kwargs):
    print(one+two+sum(args))
    print(kwargs)

kwargs_test_3(3,4,5,6,7,8,9, first=3, second=4, third=5)
```

asterisk

- 흔히 알고 있는 * 를 의미함
- 단순 곱셈, 제곱연산, 가변 인자 활용 등 다양하게 사용됨

***args**

```
def asterisk_test(a, *args):
    print(a, args)
    print(type(args))


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

****kargs**

```
def asterisk_test(a, **kargs):
    print(a, kargs)
    print(type(kargs))

asterisk_test(1, b=2, c=3,
d=4, e=5, f=6)
```

asterisk – unpacking a container

- tuple, dict 등 자료형에 들어가 있는 값을 unpacking
- 함수의 입력값, zip 등에 유용하게 사용가능

```
def asterisk_test(a, *args):
    print(a, args)
    print(type(args))


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

```
def asterisk_test(a, args):
    print(a, *args)
    print(type(args))


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

asterisk – unpacking a container


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


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

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

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

End of Document
Thank You.