DAY4
PRESENTATION
STUDY

CONTENTS

Daily Pythonic Code

- Wildcard *
- Permutation & iterator
- Counter의 응용
- Detail Sort

Wildcard *

- A = [[1,2,3],[4,5,6],[7,8,9]] , how get [[1,4,7],[2,5,8],[3,6,9]]?

→zip(A[0],A[1],A[2])

zip_obj = zip(A[0],A[1],A[2])

print(zip_obj) # <zip object at 0x000001D9B8FF6400>

print(*zip_obj) # (1,4,7) (2,5,8) (3,6,9) → ,가 없다 = iterable하고 list나 set처럼 frame이 없이 묶여있지 않는 상태

print(zip_obj[0]) # Type error- 'zip' object is not subscriptable

Wildcard *

• 그렇다면, zip_obj가 iterator의 집합인 증명?

for e in zip_obj: … / zip_iter=iter(zip_obj) print(zip_iter.__next__())

→멤버들 순차적으로 정상 출력 가능.

zip_obj는 즉, (1,4,7)->(2,5,8)->(3,6,9) 라는 순서로 iterator 객체가 대기를 타고 있는 unframe 상태를 유지하고 있다.

- mapping_list= list(map(list,zip_obj))

Print(mapping_list) # [[1,4,7],[2,5,8],[3,6,9]]

→ 틀 만들기 성공

Wildcard *

- But, A=[[1,2,3],[4,5,6],[7,8,9],….,[1e5,1e5+1,1e5+2]] 라면?

zip_obj=zip(A[0],A[1],A[2],A[3],A[4],…,A[100],A[101],…..A[2415],…)

→손이 부서진다.

Mapping_list=list(map(list,zip(*A)))

→한 줄로 뚝딱이 가능.

→어떻게 가능? Wildcard(*)는 단순히 구성물을 flat하게 풀어주는 것은 직관적으로 아는 내용이나, 더 중요한 것은 풀어주는 객체들이 전부 iterator로 반환된다는 것.

→*A = [1,2,3] -> [4,5,6] -> [7,8,9] ->……->[1e5,1e5+1,1e5+2]

→Zip(*A) = (1,4,7,10,13,…1e5) -> (2,5,8,11,14,…) -> (3,6,9,12,….)

Wildcard *

- But, A=[[1,2,3],[4,5,6],[7,8,9],….,[1e5,1e5+1,1e5+2]] 라면?

zip_obj=zip(A[0],A[1],A[2],A[3],A[4],…,A[100],A[101],…..A[2415],…)

→손이 부서진다.

Mapping_list=list(map(list,zip(*A)))

→한 줄로 뚝딱이 가능.

→어떻게 가능? Wildcard(*)는 단순히 구성물을 flat하게 풀어주는 것은 직관적으로 아는 내용이나, 더 중요한 것은 풀어주는 객체들이 전부 iterator로 반환된다는 것.

→*A = [1,2,3] -> [4,5,6] -> [7,8,9] ->……->[1e5,1e5+1,1e5+2]

→Zip(*A) = (1,4,7,10,13,…1e5) -> (2,5,8,11,14,…) -> (3,6,9,12,….)

Permutation & iterator

- 순열과 조합: itertools 패키지의 permutations(), combinations() 함수 제공.
- A=[1,2,3] -> print(permutations(A,#len(A))) # (1,2,3) (1,3,2) (2,1,3) …

→print(list(map(list,permutations(A))) # [[1,2,3],[1,3,2],[2,1,3],….]

- Target="123"?

for e in "123": → "1" -> "2" -> "3"

Permutaion("123") ("1","2","3") -> ("1","3","2") -> …

Print(list(map(list,permutations(target))) # [["1","2","3"],….]

Permutation & iterator

- How get [[“123”],[“132”],[“213”],…]?

→””.join(permutation(Target) # (“1”,”2”,”3”) –””.join-> “123”

Print(””.join(permutation(Target))) # “123” -> “132” ->”213”

Answer = list(map(””.join,permutations(Target)))

아까 거 : list(map(list,permutations(Target)))

Counter의 응용

- 프로그래머스 level1 해쉬 문제
- 동명이인을 포함할 때,

Participant와 completion를 뒤져
Unique한 member 하나를 찾는 문제.

participant	completion	return
["leo", "kiki", "eden"]	["eden", "kiki"]	"leo"
["marina", "josipa", "nikola", "vinko", "filipa"]	["josipa", "filipa", "marina", "nikola"]	"vinko"
["mislav", "stanko", "mislav", "ana"]	["stanko", "ana", "mislav"]	"mislav"

How we solve with O(N)?

→ return list((Counter(participant)-Counter(completion)).keys())[0]

Detail Sort

- 프로그래머스 sort 문제 command:[I,j,k] -> i번째부터 j번째 멤버까지 정렬 후 그 중 k 번째 멤버는 무엇인가?

- def solution(array, commands):
- 	return list(map(lambda x:sorted(array[x[0]-1:x[1]])[x[2]-1], commands))
- solution([1,5,2,6,3,7,4],[[2,5,3],[4,4,1],[1,7,3]])

pipenv

```
(base) C:₩Users₩user₩workspace>mkdir learn-pipeenv
```

```
$ pip install pipenv
```

```
$ pipenv --python 3.7
Creating a virtualenv for this project…
Pipfile: /Users/dale/learn/learn-python/Pipfile
Using /Users/dale/.pyenv/versions/3.7.6/bin/python3 (3.7.6) to create virtualenv…
⠸ Creating virtual environment...Already using interpreter /Users/dale/.pyenv/versions/3.
Using base prefix '/Users/dale/.pyenv/versions/3.7.6'
New python executable in /Users/dale/.local/share/virtualenvs/learn-python-XBV2stdv/bin/p
Also creating executable in /Users/dale/.local/share/virtualenvs/learn-python-XBV2stdv/bi
Installing setuptools, pip, wheel...
done.
Running virtualenv with interpreter /Users/dale/.pyenv/versions/3.7.6/bin/python3

✓ Successfully created virtual environment!
Virtualenv location: /Users/dale/.local/share/virtualenvs/learn-python-XBV2stdv
```

```
(base) C:₩Users₩user₩workspace₩learn-pipeenv>dir
 C 드라이브의 볼륨에는 이름이 없습니다.
 볼륨 일련 번호: 5219-2112

 C:₩Users₩user₩workspace₩learn-pipeenv 디렉터리

2021-01-22  오전 12:12    <DIR>          .
2021-01-22  오전 12:12    <DIR>          ..
2021-01-22  오전 12:12               138 Pipfile
               1개 파일                 138 바이트
               2개 디렉터리  152,339,517,440 바이트 남음
```

```
[[source]]
name = "pypi"
url = "https://pypi.org/simple"
verify_ssl = true

[dev-packages]

[packages]

[requires]
python_version = "3.7"
```

```
$ pipenv shell
Launching subshell in virtual environment…
 . /Users/dale/.local/share/virtualenvs/learn-python-XBV2stdv/bin/activate
(learn-python) $ which python
/Users/dale/.local/share/virtualenvs/learn-python-XBV2stdv/bin/python
```

```
(learn-pipeenv-NROCTeHi) (base) C:₩Users₩user₩workspace₩learn-pipeenv>pipenv install numpy==1.15.4
```

```
[[source]]
url = "https://pypi.org/simple"
verify_ssl = true
name = "pypi"

[packages]
numpy = "==1.15.4"

[dev-packages]

[requires]
python_version = "3.7"
```

```
(learn-pipeenv-NROCTeHi) (base) C:₩Users₩user₩workspace₩learn-pipeenv>exit

(base) C:₩Users₩user₩workspace₩learn-pipeenv>pip show numpy
Name: numpy
Version: 1.18.5
```

```
(base) C:₩Users₩user₩workspace₩learn-pipeenv>pipenv install matplotlib==2.2.5
Installing matplotlib==2.2.5...
Adding matplotlib to Pipfile's [packages]...
Installation Succeeded
Pipfile.lock (b6c04f) out of date, updating to (298213)...
Locking [dev-packages] dependencies...
Locking [packages] dependencies...
          Building requirements...
Resolving dependencies...
Success!
Updated Pipfile.lock (298213)!
Installing dependencies from Pipfile.lock (298213)...
  ================================ 0/0 - 00:00:00
To activate this project's virtualenv, run pipenv shell.
Alternatively, run a command inside the virtualenv with pipenv run.
```

```
(base) C:₩Users₩user₩workspace₩learn-pipeenv>pip show matplotlib
Name: matplotlib
Version: 3.2.2
Summary: Python plotting package
Home-page: https://matplotlib.org
Author: John D. Hunter, Michael Droettboom
Author-email: matplotlib-users@python.org
License: PSF
Location: c:₩users₩user₩anaconda3₩lib₩site-packages
Requires: cycler, kiwisolver, python-dateutil, pyparsing, numpy
Required by: seaborn, scikit-image, mlxtend
```

```
1   [[source]]
2   url = "https://pypi.org/simple"
3   verify_ssl = true
4   name = "pypi"
5
6   [packages]
7   numpy = "==1.15.4"
8   matplotlib = "==2.2.5"
9
10  [dev-packages]
11
12  [requires]
13  python_version = "3.7"
14
```

```
(base) C:₩Users₩user₩workspace₩learn-pipeenv>pipenv --rm
Removing virtualenv (C:₩Users₩user₩.virtualenvs₩learn-pipeenv-NROCTeHi)...
```

```
(base) C:₩Users₩user₩workspace₩learn-pipeenv>dir
 C 드라이브의 볼륨에는 이름이 없습니다.
 볼륨 일련 번호: 5219-2112

 C:₩Users₩user₩workspace₩learn-pipeenv 디렉터리

2021-01-22  오전 12:37    <DIR>          .
2021-01-22  오전 12:37    <DIR>          ..
2021-01-22  오전 12:37               180 Pipfile
2021-01-22  오전 12:37            10,335 Pipfile.lock
               2개 파일              10,515 바이트
               2개 디렉터리  151,745,642,496 바이트 남음
```

읽기 좋은 코드가 좋은 코드다

코드는 이해하기 쉬워야 한다!

- 저자(우리)가 좋은 코드 나쁜 코드를 오랫동안 분석했는데..
  - 공통적으로 내린 결론은 ‘코드는 이해하기 쉬워야 한다’

무엇이 코드를 '더 좋게' 만드는가?

```
Node* node = list->head;
if (node == NULL) return;

while (node->next != NULL) {
    Print(node->data)
    node = node=>next
}
if (node != NULL) Print(node=>data);
```

```
for (*Node node = list->head; node != Null; node = node->next) {
    Print(node->data);
}
```

무엇이 코드를 '더 좋게' 만드는가?

```
return exponent >= 0 ? mantissa * (1 << exponent) : mantissa / (1 << -exponent);
```

```
if (exponent >= 0) {
    return mantissa * (1 << exponent)
} else {
    return mantissa / (1 << -exponent)
}
```

무엇이 코드를 '더 좋게' 만드는가?

- 간결한게 중요한가?

- 친숙한게 중요한가?

가독성의 기본 정리

- 저자(우리)가 연구한 결과.. 가독성과 관련한 가장 중요한 통계적 사실을 발견했다고..
  - ‘코드는 다른 사람이 그것을 이해하는 데 들이는 시간을 최소화하는 방식으로 작성되어야 한다.’

- 동료가 코드를 봤을 때 ‘이해를 위한 시간’을 최소화 시켜야 한다

가독성의 기본 정리

- ‘이해’ – 매우 높은 기준을 적용
  - 어떤 사람이 코드를 완전히 ‘이해’한다는 것은 그가 **코드를 자유롭게 수정**하고, **버그를 짚어내고**, **수정된 내용이 어떻게 상호작용하는지** 알 수 있어야 한다

- 1인프로젝트 수행중- ‘나만 쓰는데 굳이?’
  - 6개월 후에 자신의 코드가 낯설게 보인다면, 어떤 사람이 바로 본인이 되는 것
  - 누군가 프로젝트에 합류할수도 있음
  - 내 코드가 다른 프로젝트에 적용될수도 있음

분량이 적으면 항상 더 좋은가?

- 일반적으로, 더 분량이 적은 코드로 똑같은 문제를 해결할 수 있다면 적은게 좋다!
  - Ex) 코드가 5,000줄일 때보다 2,000줄일 때 더 빨리 이해

분량이 적으면 항상 더 좋은가?

- 하지만, 분량이 좋다고 해서 좋은 것은 아니다

```
assert((!(bucket = FindBucket(key))) || !bucket->IsOccupied());
```

```
bucket = FindBucket(key);
if (bucket != Null) assert(!bucket->IsOccupied());
```

분량이 적으면 항상 더 좋은가?

- 마찬가지로 주석 처리는 '코드를 더하는 행위지만 코드를 더 빨리 이해하게 도와주기도 한다'

```
// "hash = (65599 * hash) + c"의 빠른 버전
hash = (hash << 6) + (hash << 16) - hash + c
```

- + 적은 분량으로 코드를 작성하는 것이 좋은 목표긴 하지만, **이해를 위한 시간을 최소화 하는게 더 좋은 목표**다

이해를 위한 시간은 다른 목표와 충돌하는가?

- **코드의 효율성**, **잘 구성된 아키텍처**, **테스트의 용이성** 등과 같은 다른 제약 조건도 고려해야 하지 않는가?
  - **이러한 조건들**이 때로는 **이해하기 쉬운 코드 작성과 출동**을 일으키지 않을까?
- 저자(우리)가 발견한 바로는 **이러한 조건들은 아무런 방해가 되지 않는다**

이해를 위한 시간은 다른 목표와 충돌하는가?

- 종종 정리되지 않은 코드를 어쩔 수 없이 수정해야 하는 경우
  - 뒤로 한걸음 물서서 스스로에게 물어본다면..
    - ‘이 코드는 이해하기 쉬운가?’ -> Yes, 다른 코드로 건너 뛰어도 됨

어려운 부분

- 상상 속에 존재하는 다른 자신의 코드를 읽고 이해하기 쉬운지 따져보려면 당연히 추가적인 시간과 노력이 든다.

Visibility

Java vs Python

Visibility

- 객체의 정보를 볼 수 있는 레벨을 조절하는 것
- 누구나 객체 안에 모든 변수를 볼 필요가 없음
- Encapsulation (캡슐화)
  - 정보 은닉
  - Class를 설계할 때, 클래스 간 간섭/정보 공유 최소화

Java Class에서의 Encapsulation

- 기본적으로 모든 변수와 함수는 동일 패키지에 속하는 클래스에서만 접근 가능 (default)
- 접근제어자를 사용하여 Visibility 레벨 변경
  - public: 모든 클래스에서 접근 가능
  - protected: 동일 패키지에 속하는 클래스와 하위 클래스에 의해 접근 가능
  - private: 클래스 내에서만 접근 가능

파이썬 Class에서의 Encapsulation

- 기본적으로 모든 변수와 함수는 Public
- 변수나 함수명 앞에 __ 붙이면 Private
- @property decorator
  - Private 변수에 접근 가능
  - 함수를 변수처럼 호출

Example

```python
class Test():
    def __init__(self, n, m):
        self.n = n # public
        self.__m = m # private

    @property
    def m(self):
        return self.__m

test = Test(1, 2)
print('n =', test.n)
# print(test.__m) # Error
print('m =', test.m)
```

```
(base) macaron:Day4 bomi$ /Use
p/Practice/Week1/Day4/test.py
n = 1
m = 2
```

Reference

- https://mainia.tistory.com/5574 (자바에서의 캡슐화)
- https://kimdoky.github.io/python/2018/10/17/python-encapsulation/ (파이썬에서의 캡슐화)