Python 컬렉션(리스트 & 딕셔너리)

키워드 : Python List (https://docs.python.org/3.7/tutorial/datastructures.html#dictionaries), Python Dictionary (https://docs.python.org/3.7/tutorial/datastructures.html#dictionaries), List Comprehension (https://docs.python.org/3.7/tutorial/datastructures.html#list-comprehensions)

우리는 자바스크립트를 공부하며, 배열이라는 객체에 대해서 알아보았습니다. 파이썬에는 배열이라는 객체가 존재하지 않습니다. 대신, 컬렉션(Collections) 혹은, 컨테이너 객체(Container object)라고 불리는, 특별한 네 가지 주요 개념이 있습니다. 이번 장에서는, 파이썬 코드의 굉장히 중요한 부분을 차지하는 컬렉션 자료형 중 'List'와 'Dictionary'에 대해서 정리해보도록 하겠습니다.

이번 장의 내용은 중요하기 때문에, 글이 조금 길고 난이도가 꽤 있을 수 있다는 점, 미리 말씀드립니다. 하지만, 모두 중요한 내용입니다! 하나하나 차근차근 따라와주세요! :)

리스트(List)

우선 리스트는 여러 자료를 쉼표(,)로 구분해서 하나의 자료형으로 묶는 역할을 합니다. 파이썬 컬레션 중에서 기본적이면서, 가장 많이 사용되는 자료형이라고 할 수 있습니다. 리스트는 파이썬 코드에서 사용할 수 있는 모든 것을 담을 수 있습니다. 그렇기 때문에, 리스트 안에 또 다른 리스트를 담거나, 인스턴스 객체를 담는 것 등이 모두 가능합니다.

아래 코드는 리스트를 선언하는 기본 형식에 대한 예시입니다.

```
# 빈 리스트 선언
empty_list = []
# 리스트 선언과 동시에 할당
num_list = [1, "2", 3.0, -4, [1, 2, 3]]
```

리스트는 위 예시와 같이, 리스트는 대괄호([])를 통해 선언하고, 여러 타입의 자료형을 모두 담을 수 있습니다. 이제 이 문법은 익숙한 형태일 것이라고 생각합니다. 또한, 아무 값도 할당되지 않은, 리스트 이름 = [] 형태로 초기화하고, 후에 필요에 따라 입력과 출력 등을 하는 형태로 많이 작성하게 됩니다.

리스트 인덱싱(Indexing)과 in 연산자

리스트를 선언하는 방법에 대해서 알아보았으니, 이번 절에서는 리스트에 값을 할당하는 방법과 출력하는 방법에 대해서 알아보겠습니다.

```
# 1번 코드
num_list = [1, 2, 3, 4, 5]
double_list = [num_list, num_list]

# 2번 코드
for num in num_list:
    print(num, end = ' ') # 1, 2, 3, 4, 5

# 3번 코드
for i in double_list:
    for j in i:
        print(j, end = ' ') # 1, 2, 3, 4, 5, 1, 2, 3, 4, 5

# 4번 코드
num_list[len(num_list) - 1] = 6
print(num_list) # [1, 2, 3, 4, 5, 6]

# 5번 코드
num_list = num_list + num_list
print(num_list) # [1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 5, 6]
num_list = num_list * 2
print(num_list) # [1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 5, 6, 1, 2,
3, 4, 5, 6]

# 6번 코드
double_list[1] = []
print(double_list) #[1, 2, 3, 4, 5, []]
```

우선, 위 코드를 보며 리스트 기초 사용법에 관한 대해서 알아보도록 하겠습니다.

1. 1번 코드는 num_list라는 이름으로 리스트를 선언하고, 1 ~ 5로 초기화하고 있습니다. 또한, dobule_list라는 이름으로 리스트를 하나 더 선언하고, 앞서 선언한 num_list 리스트 값을 0번째 인덱스와 1번째 인덱스에 값을 넣고, 초기화하고 있습니다.

2. 2번 코드는 num_list의 값을 출력하기 위해 for 구문을 이용한 반복문 코드입니다. Python 기본 문법에 대해서 이야기할 때 알아보았듯이, in이라는 멤버쉽 연산자를 이용하면, 컬렉션 안에 들어있는 값을 하나씩 꺼내올 수 있습니다. 마찬가지로, pirnt() 함수의 마지막 인자로 end = ' '를 작성해주면, 출력 값을 공백으로 구분하여 한 줄에 출력할 수 있습니다.

3. 3번 코드는 리스트 안에 들어있는 리스트를 출력하기 위해 중첩 반복문을 사용한 코드 예시입니다. 하나하나 살펴보면, double_list 안에 들어있는 리스트 값을 받아와서 다시 한번 반복문을 진행하고 있습니다. i 변수에는 num_list가 할당되며, j 변수에는 num_list의 값이 하나씩 할당되어 출력되게 됩니다.

4. 4번 코드는 대괄호([])를 이용해서 값에 접근하는 방법입니다. 자바스크립트의 배열과 같이, 0부터 시작하는 인덱스를 이용해 값에 접근할 수 있습니다. num_list[len(num_list) - 1] 코드는 다음 절에서 살펴볼 append() 메서드와 같은 역할을 할 수 있으며, 마지막 인덱스(리스트의 크기 - 1)에 값을 할당하는 예시입니다.

5. 5번 코드는 우리가 잘 알고 있는 +, * 기호를 이용해서 리스트를 확장하는 코드 예시입니다.

6. 6번 코드는, double_list의 첫번 째 인덱스 요소의 값을 빈 리스트로([]) 할당하는 코드입니다.

위 코드 예시의 반복문 코드에서 in 연산자를 이용해서 코드를 작성했습니다. 여러 데이터를 하나로 묶는 컬렉션에 값을 할당하거나, 출력하기 위해서 자주 사용되는 연산자이므로, 꼭 기억해주세요! :)

리스트 슬라이싱(Slicing)

다음은 리스트 슬라이싱 입니다. 리스트 슬라이싱이란, 리스트에 들어있는 값들 중 특정 부분만 따로 떼서 선택하는 방법입니다. 주로 다수의 값 중 조건에 만족하는 값을 검색하고 수정, 삭제 등을 하기 위해 사용 할 수 있습니다. 인덱스는 0부터 시작하며, range() 함수처럼, 마지막 요소의 n-1 까지만 슬라이싱 합니다. 표현법은 변수명[시작인덱스:끝인덱스+1]처럼 표현합니다.

아래 코드는 리스트 슬라이싱의 예시입니다.

```
a = [1, 2, 3, 4]

print(a[:]) # 1, 2, 3, 4 (0~끝)
print(a[0:]) # 1, 2, 3, 4 (0~끝)
print(a[:1]) # 1 (0~(1-1))
print(a[1:3]) # 2, 3 (1~(3-1))
print(a[:-1]) # 1, 2, 3 (0~(1-(-1)))
```

위 코드 예시를 통해 알 수 있듯이, 마지막 인덱스가 생략되면, **지정한 시작 인덱스부터, 마지막 인덱스까지의 값**을 가져올 수 있습니다. 마찬가지로, 시작 인덱스가 생략되면, 리스트의 **0번째 인덱스부터, 지정한 끝 인덱스 - 1** 까지의 값을 가져올 수 있습니다.

참고로, 앞선 장에서 리스트의 마지막 인덱스에 값을 할당하는 예시인 a[(len(a) - 1)] 코드는 슬라이스 코드인, a[len(a):]로도 표현 할 수 있습니다.

리스트 메서드

앞서, 리스트는 컨테이너 객체 중에 하나라고 말씀드렸습니다. 리스트도 하나의 객체이기 때문에, 굉장히 다양한 메서드를 제공합니다.

아래 표는 리스트 메서드의 종류와 의미를 나타냅니다.

메서드	의미
list.append(x)	리스트의 끝에 새로운 요소를 추가합니다.
list.extend(iterable)	순서를 갖는(iterable) 요소를 리스트의 끝에 새롭게 추가합니다.
list.insert(i, x)	포지션을 지정해서 리스트에 값을 추가합니다. 첫 번째 인자에 리스트의 인덱스를 작성하고, 두 번째 인자에 추가할 요소를 작성합니다.
list.remove(x)	리스트 안에 '값'을 특정지어 삭제합니다. 값을 찾을 수 없는 경우 에러가 발생합니다.
list.pop(i)	리스트 안에 마지막 인덱스 요소를 삭제함과 동시에, 삭제한 값을 반환합니다.
list.clear()	리스트의 모든 요소를 삭제합니다.
list.index(x, start, end)	리스트 안의 요소 중에 첫 번째 인자로 받은 값을 찾아 요소의 인덱스를 반환합니다. 요소는 0번째 요소부터 찾기 시작하여, 가장 먼저 찾은 인덱스를 반환합니다. 시작 인덱스와 끝 인덱스를 지정하여 특정 범위에서 찾는 것이 가능하며, 찾지 못할 경우에 에러를 발생시킵니다.
list.count(x)	리스트 안에 요소의 개수를 반환합니다.
list.sort(key=None, reverse=False)	리스트 안의 요소를 정렬합니다. key 인자를 이용하면 정렬할 기준을 정할 수 있습니다. reverse 인자를 통해서 내림차순, 오름차순을 결정할 수 있습니다. 기본 값은 오름차순입니다.
list.reverse()	리스트 안의 요소를 내림차순으로 정렬합니다.

list.copy()	리스트를 얕은 복사(Shallow Copy)합니다.

조금 많죠..? ^^ 하지만, 우리가 잘 알고 있는 영어 단어로 메서드 이름이 구성되어 있기 때문에, 어렵지 않게 이해하고, 사용하실 수 있을 거라 생각합니다! :) 위 내용 중 `sort()` 메서드의 `key` 인자와 `reverse` 인자에 대한 더 자세한 내용은 다음 장에서 알아보도록 하겠습니다. 마찬가지로, 얕은 복사와 관련된 내용은 따로 정리하도록 하겠습니다.

아래 코드는 리스트의 메서드 사용 예시입니다.

```
rainbow = ['red', 'orange', 'yellow', 'green', 'blue', 'indigo', 'violet']

del rainbow[len(rainbow):] # 마지막 인덱스 요소 삭제
print(rainbow) # ['red', 'orange', 'yellow', 'green', 'blue', 'indigo']
rainbow.append('violet') # 추가
print(rainbow) # ['red', 'orange', 'yellow', 'green', 'blue', 'indigo']
print(rainbow.pop(len(rainbow) - 1)) # violet
print(rainbow.index('red')) # 0
print(rainbow.count('red')) # 1
del rainbow[:] # 전체 요소 삭제
print(rainbow) # []
```

위의 내용 중 메서드에서 다루지 않은 내용이 있습니다. `del`이라는 키워드인데요, 리스트의 `remove()` 메서드를 이용하면, 리스트의 값으로 요소를 삭제할 수 있습니다. 반대로 `del` 구문을 이용해서 값을 삭제하면, 리스트의 인덱스를 통해 값을 삭제할 수 있습니다.

리스트 시퀀스(Sequence)

우리가 이번 절에서 알아보고 있는 리스트는 시퀀셜(sequential)한 자료형이라고 할 수 있습니다. 파이썬에서 이러한 자료형은 리스트, 튜플, 문자열 등이 있는데요, 이렇게 시퀀셜한 자료형은 순서를 갖습니다.

파이썬에선 이러한 순서성 정보를 제공하는 `enumerate` 함수를 이용하면 값과 인덱스를 함께 반환 받을 수 있습니다.

아래 코드는 반복문을 통해 `enumerate` 함수를 이용해 리스트 요소의 순서와 함께 값을 반환하는 예시입니다.

```
rainbow = ['red', 'orange', 'yellow', 'green', 'blue', 'indigo', 'violet']

for i, v in enumerate(rainbow):
    print(i, v, end= ' ') # 0 red 1 orange 2 yellow 3 green 4 blue 5 indigo 6
violet
```

`enumerate` 함수의 인자에 값을 반복문을 통해 출력하면, 인덱스와 값을 함께 반환합니다. 때문에 `for` 구문의 인자를 `i`와 `v` 두 개를 작성하 것을 알 수 있습니다. 반복 구문에서 유용하게 사용될 수 있는 키워드이므로, 참고로 알아주세요! :)

여기서 더 주의 깊게 봐야 할 내용은, 리스트 컬렉션은 순서 성을 갖는다는 점입니다. 어떻게 보면 당연하죠..?^^ 하지만, 잠깐 언급했듯이, 리스트, 튜플, 문자열 등은 순서 성을 갖지만, 그렇지 않은 자료형도 존재한다는 것을 알 수 있습니다. 또한, 그러한 차이로 인해서 메서드라던지, 함수의 사용 등이 천차만별로 달라지게 됩니다.

코드를 작성할 때, 어떤 컬렉션에 값을 입력하고, 출력할지 정하는 문제는 생각보다 어렵습니다. 기본은 리스트이지만, 앞으로 더 살펴볼 다른 컬렉션 자료형의 특징과 차이점을 기억하고, 최적의 자료 구조를 구성할 수 있도록 노력해보세요! :)

딕셔너리(Dictionary)

이번 절에서는, 리스트와 더불어 파이썬 코드를 작성할 때 자주 사용되는 객체인 딕셔너리에 대해서 알아보겠습니다. 딕셔너리는 키(Key)와 값(Value)을 하나의 요소로 갖는 컬렉션입니다.

딕셔너리는 흔히 Map 이라고도 불리는데, 키(Key)로 신속하게 값(Value)을 찾아내는 해시 테이블(Hash Table) 구조를 갖습니다. 선언 방법은 중괄호({})를 사용하며, `a = {키:값, 키:값, ...}` 형태로 콜론(:)을 통해 키와 값을 구분합니다.

아래 코드는 딕셔너리를 선언하고 사용하는 기본 예시입니다.

```
a = {"a":1, "b":2, "c":3}
a["a"] = 0 # 수정
a["d"] = 4 # 추가
del a["a"] # 삭제
print(a) # {'b': 2, 'c': 3, 'd': 4}

for key in a:
  print(a[key], end = ' ') # 2 3 4
```

우선, 딕셔너리는 키와 값 형태로 이루어져 있고, 기본적으로 키를 통해서 값에 접근하는 형태입니다. 리스트의 인덱스로 값을 접근하기 위해서 정수 값을 입력했다면, 딕셔너리에서는 키 값을 입력합니다. 만약, 딕셔너리에 접근하기 위해 작성한 키 값이 없을 경우엔, 에러를 발생시킵니다.

딕셔너리 메서드

딕셔너리도 리스트와 마찬가지로, 여러 가지 메서드를 제공합니다. 여러 메서드 중에서 값을 출력하기 위해서 사용할 수 있는 주요 메서드에 대해서 알아보도록 하겠습니다.

아래 코드는 딕셔너리의 키, 값을 반환하는 메서드를 사용하는 예시입니다.

```
a = {"a": 1, "b": 2, "c": 3}
# keys
for k in a.keys():
    print(k, end=' ') # a b c

# values
for v in a.values:
    print(v, end=' ') # 1 2 3

# items
for k, v in a.items():
    print(k, v, end = ' ') # a1 b2 c3
```

위 코드 예시 내용은 어렵지 않게 이해하실 수 있을 거라고 생각합니다. 딕셔너리의 `keys()` 메서드는 딕셔너리 안에 있는 키만을 반환합니다. `values()` 메서드는 딕셔너리 안에 있는 값만을 반환합니다. 만약, 키와 값을 모두 반환받을 경우엔 `items()`라는 메서드를 사용합니다.

생성자

리스트와 딕셔너리는 모두 파이썬에서 기본적으로 제공하는 객체입니다. 이 컨테이너 객체는 각각의 이름으로 생성자를 갖고, 서로 다른 타입의 컬렉션 타입을 형변환 할 수 있습니다.(리스트: `list()`, 딕셔너리: `dict()`)

아래 코드는 생성자를 이용해 서로 다른 컬렉션 타입을 변환하는 예시입니다.

```
a = [[1, 2], [3, 4], [5, 6]]
b = {"a":1, "b":2, "c":3}

print(list(b.keys())) # ['a', 'b', 'c']
print(list(b.values())) # [1, 2, 3]
print(list(b.items())) # [('a', 1), ('b', 2), ('c', 3)]
print(dict(a)) # {1: 2, 3: 4, 5: 6}
```

우선, a 변수엔 리스트 안에 리스트를 담는 2차원 리스트 형태로 초기화합니다. b 변수엔 딕셔너리를 선언하고 초기화합니다. 리스트 생성자(`list()`) 안에 딕셔너리의 키, 값, 혹은, 키와 값 모두를 반환하는 메서드를 인자로 넘겨주면, 리스트로 변환되어 값을 반환합니다. 단, `items()` 메서드로 변환할 경우엔 키와 값을 '튜플'이란 자료형으로 만들어 반환합니다. 튜플에 대한 내용은 다음 장에서 정리해드리도록 하겠습니다. :)

한 가지 주의할 점이 있습니다. 딕셔너리 생성자(`dict()`)를 통해 컬렉션 자료형을 변환하는 경우에, 위 코드처럼 키, 값 형태로 나타낼 수 있어야 딕셔너리로 변환될 수 있습니다. 예를 들어, `a = [1, 2, 3, 4, 5]` 형태로 리스트가 선언된 상태에서 형 변환을 진행하려고 하면 `TypeError`라는 에러를 발생시킵니다.

서로 다른 컬렉션을 변환시키는 작업은 파이썬 코드 작성 시 자주 사용되는 코드입니다. 각각의 이름으로 생성자를 호출하여, 형 변환을 할 수 있다는 점 기억해주세요! :)

정렬 알고리즘(Sorting Algorithm)

우리가 프로그래밍에 대해 이야기할 때, 빠지지 않고 이야기하는 것이 있습니다. 바로 제목에 나와있는 알고리즘(Algorithm)인데요, 알고리즘은 어떤 문제를 해결하기 위해 정형화된 코드 구문입니다. 이미 많은 문제를 해결하기 위해 알고리즘이 구성되어 있습니다.

그렇다면, 알고리즘은 왜 공부해야 할까요? 다들 잘 아시겠지만, 컴퓨터는 굉장히 빠른 연산 속도를 자랑합니다. 또한, 메모리 기술의 비약적인 발전을 통해, 우리는 프로그램 코드 작성시 많은 메모리 공간을 사용할 수 있습니다. 하지만 그만큼 우리가 프로그램 상에서 다루는 연산은 굉장히 큰 수를 다루기도 하며, 굉장히 많은 양의 데이터를 처리하기도 합니다. 이러한 경우엔, 아무리 빠른 연산 속도를 자랑하는 컴퓨터라고 해도 부담이 아닐 수가 없습니다.

알고리즘은 여러 가지 프로그래밍 파트 중에서도 상당히 난이도가 높은 분야에 속합니다. 하지만, 굉장히 굉장히 중요합니다!! 우리는 이번 절에서 정렬이라는 기초 알고리즘에 대해서 알아볼 것입니다. 정렬 알고리즘을 통해서 알고리즘이 왜 필요하고, 어떻게 구성할 수 있는지 생각해보는 시간이 되었으면 좋겠습니다. :)

정렬은 이름에서 느껴지듯이, 여러 요소를 어떤 기준에 따라서 재배열하는 것을 이야기합니다. 예를 들어, 우리가 흔히 사용하는 웹, 앱 애플리케이션에서 최신순, 공감순 등의 내용을 보신 적이 있을 겁니다. 모두 어떤 기준에 맞춰서 데이터를 재배열하는 작업이라고 할 수 있습니다. 정렬 알고리즘은 상대적으로 오래된 알고리즘 분야입니다. 그렇기 때문에, 상당 부분에 대해서 해답이 나와있습니다. 또한, 파이썬에서는 이미 정렬을 위한 메서드와 함수가 존재합니다. (`e.g.`, `sort()` 메서드)

하지만, 모든 경우를 기본으로 제공하는 메서드로 해결할 수 없습니다. 또한, 알고리즘의 목적 자체가 문제 해결과 직결되는 부분이기 때문에, 알고리즘을 공부하는 것이 프로그래밍이란 넓은 분야를 이해하는 데에 많은 도움이 됩니다. 이번 절에서는 많은 종류의 정렬 알고리즘 중에서, 상대적으로 구현하기 쉬운 삽입 정렬 알고리즘에 대해서 알아보겠습니다.

삽입 정렬(Insertion Sort)

삽입 정렬은 이름에서 느껴지듯이, 정렬이 필요한 집합 안에서 기준에 따라 자신의 위치를 찾아 삽입하는 정렬 방법입니다. 구현하기 간단하며, 상대적으로 적은 수의 요소를 갖는 리스트를 정렬하기에 용이합니다. 기본적으로 우리는 아래 코드와 같이 랜덤 하게 정렬되어 있는 다섯 개의 정수 값을 오름 차순으로 정렬하고 싶다고 가정하겠습니다.

```
num_list = [1, 5, 4, 3, 2]
```

정렬 코드를 작성하기 전에, 삽입 정렬이 어떻게 이루어지는지 먼저 살펴보도록 하겠습니다. 삽입 정렬은 반복에 따라 각각의 부분으로 나눠서 정렬하는 방법입니다. 반복이 진행될 때마다 해당 인덱스에 해당하는 위치까지 정렬이 진행되며, 새롭게 정렬이 필요한 값과 선행된 정렬 부분의 가장 마지막 요소 값을 비교하여 정렬이 필요한 값이 더 작다면, 정렬을 진행해나갑니다.

결과적으로, 이전 요소와 삽입할 요소를 비교하면 됩니다. 이전 요소가 삽입할 요소보다 큰 경우 이전 요소를 다음 위치로 이동합니다. 첫 번째 요소는 정렬이 되어있다고 가정하기 때문에, 이전 요소와 비교하기 위해서 인덱스는 1부터 시작하게 됩니다.

1.

```
key = 5 # 첫 번째 인덱스를 가준으로 시작합니다.
키를 통해 앞의 요소와 비교해 나갑니다.

가장 먼저, 1과 5를 비교합니다. 1 < 5이므로, 정렬이 필요하지 않습니다.

        결과: [1 5 4 3 2]
```

2

```
key = 4 # 두 번째 인덱스의 값을 저장합니다.
5 > 4 # 4가 5보다 작은 값을 나타내므로, 더 앞의 요소와 비교합니다.
1 < 4 # 4는 1보다 크기 때문에, 더 비교하지 않고 4를 첫 번째 인덱스에 삽입합니다.

        결과: [1 4 5 3 2]
```

3

```
key = 3 # 세 번째 인덱스의 값을 저장합니다.
5 > 3 # 3이 5보다 작은 값을 나타내므로, 앞의 요소와 비교합니다.
4 > 3 # 3이 4보다 작은 값을 나타내므로, 앞의 요소와 비교합니다.
1 < 3 # 4가 5보다 작은 값을 나타내므로, 더 비교하지 않고 3을 첫 번째 인덱스에 삽입합니다.
        결과: [1 3 4 5 2]
```

4

```
key = 2 # 네 번째 인덱스의 값을 저장합니다.
5 > 2 # 2가 5보다 작은 값을 나타내므로, 앞의 요소와 비교합니다.
4 > 2 # 2가 4보다 작은 값을 나타내므로, 앞의 요소와 비교합니다.
3 > 2 # 2가 3보다 작은 값을 나타내므로, 앞의 요소와 비교합니다.
1 < 3 # 4가 5보다 작은 값을 나타내므로, 더 비교하지 않고 3을 첫 번째 인덱스에 삽입합니다.
        결과: [1 2 3 4 5]
```

위 내용을 실제 파이썬 코드로 작성하면 아래와 같이 작성할 수 있습니다.

```
num_list = [1, 5, 4, 3, 2]

for i in range(1, len(num_list)):
    j = i - 1 # 삽입할 요소보다 앞의 인덱스
    key = num_list[i] # 삽입할 값
    while num_list[j] > key and j >= 0: # 반복문 조건 비교
        num_list[j+1], j = num_list[j], j - 1 # 값을 대입
    num_list[j+1] = key # 요소 삽입

for i in num_list:
    print(i, end= ' ') # 1, 2, 3, 4, 5
```

우리가 코드를 작성하는 데 익숙해져서, 바로 코드를 작성할 수 있다면 좋겠지만, 머리 속에 내용을 바로 코드로 표현하는 것은 많이 어려운 작업입니다. 그렇기 때문에, 코드를 작성하기 전에, 그림을 그리거나 코드의 흐름을 하나하나 글로 작성해보는 것이 도움이 될 수 있습니다.

우리가 알고리즘을 구성하고, 문제 풀이를 위해선 기본적으로 해당 프로그래밍 언어에 대한 이해뿐만 아니라, 문법과 사용할 수 있는 함수, 객체의 메서드 등을 두루두루 알고 있어야 합니다. 예를 들어, 리스트의 메서드를 기억하고 있어야만 리스트를 이용한 알고리즘을 빠르고, 효과적으로 구성할 수 있습니다.

프로그래밍을 함에 있어서 알고리즘은 절대 멀어질 수 없는 부분입니다. 기초적인 알고리즘부터, 상당한 난이도를 갖는 다양한 알고리즘이 존재합니다. 이미 많은 사이트에선 알고리즘 관련 문제와 풀잇법을 제공하고 있습니다. 목표를 잡고, 꾸준히 알고리즘 문제 풀이에 도전해보세요! :)

List Comprehension

파이썬에서는 간결한 방법으로 원하는 리스트를 만들 수 있는, Comprehension이라는 문법이 존재합니다. 조금 더 깊게 이야기하면, 특정 조건(지정된 표현식)에 맞게 새로운 컬렉션을 생성하는 것을 말합니다. 기본적인 선언 방법은 “변수명 = [표현식 for 요소 in 컬렉션 [if 조건식]]”입니다.

아래 코드는 1부터 10 까지의 정수를 리스트에 할당하는 두 가지 방법입니다.

```
# 1번 코드
num1to10 = []
for i in range(1, 11):
    num1to10.append(i)

# 2번 코드
num1to10 = [x for x in range(1, 11)]
```

먼저, 1번 코드는 우리가 잘 아는 문법 형태입니다. 빈 리스트를 먼저 선언하고, 반복문과 range() 함수를 통해서 값을 리스트에 할당하고 있습니다. 2번 코드가 문법이 조금 독특한데요, 간단히 살펴보도록 하겠습니다.

우선 표현식이라고 불리는 부분에 변수 혹은, 다양한 연산 코드를 작성합니다. 위 코드 예시에서는 x가 쓰인 부분입니다. 표현식 다음에 나오는 부분은 우리가 일반적으로 알고 있는 반복문 코드를 원하는 형태로 작성합니다. 반복문 안에 어떤 조건을 담고 싶을 경우에는 조건문도 함께 작성할 수 있습니다.

아래 코드는 1부터 10까지의 수 중에서, 짝수의 값만을 제곱하여 리스트를 만들어 반환하는 예시입니다.

```
even1to10 = [x ** 2 for x in range(1, 11) if x % 2 == 0]
print(even1to10) # [4, 16, 36, 64, 100]
```

이 밖에도 파이썬에서는 상대적으로 간결하면서도 독특한? 문법을 많이 사용할 수 있습니다. 지금 당장은 어색하고 이해하기 힘들더라도, 코드를 작성하다 보면 자연스럽게 익숙해질 수 있는 부분이므로 걱정하지 않으셔도 됩니다!

이번 장에서는, 컨테이너 객체 중 리스트와 딕셔너리에 대해서 알아보았습니다. 이번 장에 내용은 다양한 내용이 새롭게 나왔고, 많은 부분이 처음 프로그래밍을 공부하는 분들에겐 꽤나 난이도 있는 내용이라 생각합니다. (물론, 저라고 쉽다는 것은 절대 아닙니다!!)

제가 생각하기에 프로그래밍이 어려운 이유는, 유독 공부를 하면 할수록 더 많은 것들을 새롭게 공부해야 한다는 점이 가장 큰 이유이지 않을까 생각합니다. 정해진 답은 없지만, 결국 꾸준한 공부와 최대한 많은 코드를 직접 작성해보는 것, 중요한 내용을 글로 정리해두는 것 등이 수반되어야 하는 점에는 이견이 없을 거라고 생각합니다. 조금 당연한 말이죠..?^^
힘들어도 엄청난 노력으로 얻은 결과는 배신하지 않을 것입니다. 긴 글 읽으시느라 고생하셨습니다. 파이팅입니다! :)