LLM 활용 인공지능 서비스 개발자 양성과정

도봉 SeSAC 캠퍼스 X Saltlux

강사 최동혁

파이썬: 리스트와 튜플

리스트란 무엇인가?

- 가장 자주 보는 자료형
- 리스트 자료형은 자유도와 활용도가 매우 높아 파이썬 개발자들에게 자주 사용되는 자료형이다.
- 사실 인간의 인지 과정에서 자연스럽게 이해되는 자료구조다.
- 리스트 자료형이 갖고 있는 네 가지 큰 특징은 다음과 같다:
  - 반복 가능(iterable)
  - 원소 자료형 무관
  - 변경 가능(mutable)
  - 유익한 메소드

리스트 - 반복 가능(iterable)

- 리스트는 반복 가능한(iterable) 자료형이다.
- 수학에서 배운 배열(array)을 일반화한 것으로 간주하면 이해하기 쉽다.
- 반복 가능한 자료구조로는 또 뭐가 있을까?

```
my_first_list = [0, 1, 2, 3, 4, 5]
print(type(my_first_list))
print(len(my_first_list))
for item in my_first_list:
    print(item, end=" ")
```

리스트 - 반복 가능(iterable): 요소 1개 접근하기

- 리스트와 같은 iterable 자료형들은 내부 각 원소에 대한 접근 및 전달이 가능한 것이 특징이다.
- 인덱싱과 슬라이싱은 iterable 객체 내부 원소(들)에 대한 접근 방식을 의미한다.
- 인덱싱(indexing): 원소 한 개에 대한 접근

```
my_first_list = [0, 1, 2, 3, 4, 5]
print(my_first_list[0]) # 맨 처음 원소 인덱싱
print(my_first_list[-1]) # 마지막 원소 인덱싱
print(my_first_list[-3]) # 뒤에서 세번째 원소 인덱싱
print(my_first_list[6]) # 인덱싱 에러 (범위 밖)
```

리스트 - 반복 가능(iterable): 요소 N개 접근하기 (1)

- 리스트와 같은 iterable 자료형들은 내부 각 원소에 대한 접근 및 전달이 가능한 것이 특징이다.
- 인덱싱과 슬라이싱은 iterable 객체 내부 원소(들)에 대한 접근 방식을 의미한다.
- 슬라이싱(slicing): 원소 N 개에 대한 접근

```
my_first_list = [0, 1, 2, 3, 4, 5]
print(my_first_list[0:3]) # 0 이상 3 미만 인덱스의 원소들 슬라이싱
print(type(my_first_list[0:3])) # 슬라이싱의 결과는 리스트
print(my_first_list[2:3]) # 2 이상 3 미만 (즉, 두번째 원소 only)
print(type(my_first_list[2:3])) # 슬라이싱의 결과는 리스트
print(my_first_list[:])      # 전체 원소
print(type(my_first_list[:])) # 슬라이싱의 결과는 리스트
```

리스트 - 반복 가능(iterable): 요소 N개 접근하기 (2)

- 슬라이싱(slicing): 원소 N 개에 대한 접근.
- 사실 온전한 슬라이싱은 두 개의 콜론을 사용하여 표현한다.
- [start:end:step]형태에서 start은 슬라이싱 시작 인덱스, end는 슬라이싱 마지막+1 인덱스, step은 건너뛰는 크기를 의미한다. (양수는 오른쪽 방향, 음수는 왼쪽 방향)

```
my_first_list = [0, 1, 2, 3, 4, 5]
print(my_first_list[2:5])      # 전체 원소
print(my_first_list[2:5:1])  # 위와 동치 (하나씩 건너뛰며 순회)
print(my_first_list[2:5:2])
```

리스트 - 변경 가능(mutable)

- 파이썬에서 객체는 변경이 가능한(mutable) 자료형과 아닌(immutable) 것이 있다.
- list, dict, set의 경우는 mutable 자료형에 속하고 나머지는 immutable하다고 생각하면 된다.
- mutable 객체의 장점은 내부 원소의 수정과 삭제가 자유롭다는 것이다.

```
my_first_list = [0, 1, 2, 3, 4, 5]
print(my_first_list)
my_first_list[0] = "Start" # 0번째 원소 재할당
print(my_first_list) # 수정된 리스트 출력
my_first_list[3:] = "0" # 3번째 이후 구간은 "0"  원소 하나로 재할당
print(my_first_list)
del(my_first_list[0]) # 빌트인 함수 del을 활용한   0번째 원소 제거
print(my_first_list)
```

리스트 - 변경 가능(mutable): immutable 과 구분하기

- immutable 자료형에 속하는 str(string; 문자열) 자료형은 내부 원소 값의 조회는 되지만, 수정을 시도하려고 하면 TypeError가 발생함을 알 수 있다

```
string_example = "Hello"
print(string_example[0])
print(string_example[2:])
```

```
string_example[0] = 'h' # 자료형 오류
```

리스트 - 변경 가능(mutable): immutable 과 구분하기

- 형 변환 (Type Casting)을 통해 mutable 자료형으로 변경한 뒤 다시 원래 자료형으로 되돌릴 수 있다

```
list_example = list(string_example) # 리스트 내장 함수를 활용하여 형 변환
list_example[0] = 'h' # 리스트 0번째 값 수정
string_example = "".join(list_example) # 문자열 자료형의 join 메소드를 활용 (list -> str)
```

리스트 - 원소 자료형 무관

- 리스트 내부 원소들의 자료형은 일관되지 않아도 된다.
- 즉, 원소가 어떠한 자료형을 가지는지 신경을 쓰지 않고 자유자재로 다룰 수 있다.

```
my_second_list = ["a", None, 3, 4.01, [5]]
print(my_second_list)
for value in my_second_list:
    print(value, end=" ")
print(my_second_list[-1])
```

리스트 - 유익한 메소드 제공

- 리스트에는 유익한 메소드(클래스 함수)를 다양하게 제공하고 있다.
- 이들을 적재적소로 잘 활용하는 것이 개발자로서는 무척 중요하다.
- 무엇이든 개발 도중에 궁금한 것이 발생하면 빌트인 함수인 help를 사용하거나 구글링을 하자.
- 본 수업에서 다뤄볼 메소드들은 총 아홉가지이며 모두 빈번히 사용되므로 사용법을 잘 숙지하자.
- JupyterHub에서 계속

1. `append`
2. `pop`
3. `count`
4. `index`
5. `remove`
6. `sort`
7. `reverse`
8. `insert`
9. `extend`

튜플 (Tuple) 자료형

- 튜플(tuple)은 몇 가지 점을 제외하곤 리스트와 거의 비슷하며 리스트와 다른 점은 다음과 같다.
- 튜플 자료형이 갖고 있는 네 가지 큰 특징은 다음과 같다:
    1. 변경 불가능(immutable)
    2. 반복 가능(iterable)
    3. 원소 자료형 무관
    4. 유익한 메소드

튜플은 어떻게 만들까?

- 튜플은 다음과 같이 여러 가지 방법으로 생성할 수 있다.

```
t1 = ()
t2 = (1,)
t3 = (1, 2, 3)
t4 = 1, 2, 3
t5 = ('a', 'b', ('ab', 'cd'))
```

튜플은 어떻게 만들까?

- 튜플에서는 리스트와 다른 2가지 차이점이 있다.
  - t2 = (1,)처럼 단지 1개의 요소만을 가질 때는 요소 뒤에 쉼표(,)를 반드시 붙여야 한다는 것
  - t4 = 1, 2, 3처럼 소괄호(())를 생략해도 된다는 점
- 얼핏 보면 튜플과 리스트는 비슷한 역할을 하지만, 프로그래밍을 할 때 튜플과 리스트는 구별해서 사용하는 것이 좋다.
- 튜플과 리스트의 가장 큰 차이는 요솟값을 변화시킬 수 있는가 여부이다. 즉, 리스트의 요솟값은 변화가 가능하고 튜플의 요솟값은 변화가 불가능하다.
- 튜플과 리스트의 사용 예시:
  - 실행되는 동안 요솟값이 항상 변하지 않기를 바랄 때 -> 튜플
  - 수시로 그 값을 변화 시켜야 할 때 -> 리스트
- 실제 프로그램에서는 값이 변경되는 형태의 변수가 훨씬 많기 때문에 평균적으로 튜플보다 리스트를 더 많이 사용한다.

튜플 - 변경 불가능(immutable)

- 튜플의 요솟값은 한 번 정하면 지우거나 변경할 수 없다.

1. 튜플 요솟값을 삭제하려 할 때

```
t1 = (1, 2, 'a', 'b')
## del t1[0]  ## This will raise an error
```

튜플의 요소를 리스트처럼 del 함수로 지우려고 한 예이다. 튜플은 요솟값을 지울 수 없다는 오류 메시지를 확인할 수 있다.

2. 튜플 요솟값을 변경하려 할 때

```
t1 = (1, 2, 'a', 'b')
## t1[0] = 'c'  ## This will raise an error
```

튜플의 요솟값을 변경하려고 해도 오류가 발생하는 것을 확인할 수 있다.

튜플 - 반복 가능(iterable)

- 튜플은 리스트와 마찬가지로 반복 가능한(iterable) 자료형이다.

```
my_first_tuple = (0, 1, 2, 3, 4, 5)
print(type(my_first_tuple))
print(len(my_first_tuple))
for item in my_first_tuple:
    print(item, end=" ")
```

인덱싱과 슬라이싱은 리스트와 동일하게 작동한다

```
print("\n"+str(my_first_tuple[3]))
print(my_first_tuple[2:4])
print(my_first_tuple[-1:-4:-2])
```

튜플 - 원소 자료형 무관

- 튜플 내부 원소들의 자료형 역시 리스트와 마찬가지로 일관되지 않아도 된다.
- 즉, 원소가 어떠한 자료형을 가지는지 신경을 쓰지 않고 자유자재로 다룰 수 있다.

```
my_tuple = (0, 1, "2")
my_second_tuple = (3, 4.0)
my_tuple = my_tuple.__add__(my_second_tuple)
print(my_tuple)
```

튜플 - 유익한 메소드 제공 및 그 외

- 튜플에도 리스트와 마찬가지로 유익한 메소드(클래스 함수)를 제공하고 있다.
- count, index

```
my_tuple = (0, 1, "2")
print(my_tuple.count(0))
print(my_tuple.index("2"))
print(my_tuple.index(2))
```

튜플 - 유익한 메소드 제공 및 그 외

- 다만, 튜플은 immutable하므로 원소 값의 수정 및 삭제와 관련된 메소드는 존재하지 않는다
- 내장 함수 len, max, min을 통해 튜플의 원소 개수, 최댓값, 최솟값을 구할 수 있다

```
my_first_tuple = (0, 1, 2, 3, 4, 5)
print(len(my_first_tuple))
print(max(my_first_tuple))
print(min(my_first_tuple))
```

튜플 - 유익한 메소드 제공 및 그 외

- 튜플에서도 + 과 * 연산자를 지원한다.
- +는 튜플의 add 메소드 역할과 동일하다.
- *은 “튜플 자기 자신을 몇 번 반복하여 연장할 것인가”를 수행하는 연산자이다.

```
tuple_A = (1, 2, 1, 3)
tuple_B = ("a", "B", 10.0)
print(tuple_A + tuple_B)
print(tuple_A * 3)
print((tuple_A * 3).count(1))
```