인덱싱, 슬라이싱 및 함수

박범진

Department of Statistics
University of Seoul

인덱싱과 슬라이싱 I I

인덱싱(Indexing)과 슬라이싱(slicing)

- 인덱싱과 슬라이싱은 객체(리스트, 튜플 등)의 원소 (데이터)에 접근하는 것을 의미함.
- 단일 원소를 추출하는 것을 인덱싱이라 하며 일정 구간의 원소를 추출하는 것을 슬라이싱 혹은 subsetting이라 부름.

인덱싱과 슬라이싱 II

리스트와 튜플의 인덱싱 및 슬라이싱

- 리스트와 튜플의 인덱싱과 슬라이싱은 동일하므로 같이 설명함.
- 리스트와 튜플에 사용되는 인덱싱과 슬라이싱은 Numpy와 Pandas의 객체에서의 인덱싱과 거의 동일하므로 필히 숙지해야함.

인덱싱과 슬라이싱 III

- 리스트와 튜플에서의 인덱싱 및 슬라이싱은 대괄호 ([])를 통해 수행됨.
- 인덱싱에는 양의 정수와 음의 정수를 이용한 방법이 존재함.
- 양의 정수를 이용한 인덱싱은 입력한 정수에 해당하는 위치의 원소를 반환해줌.

```
# Positive integer indexing
>>> x = [1, 2, 3, 4]
>>> x[0]
1
>>> x[1]
2
```

인덱싱과 슬라이싱 IV

- 주의해야할 점은 Python의 원소 위치는 0부터 시작함.
- 즉, 리스트 (혹은 튜플)의 첫번째 원소는 x[0], 두번째 원소는 x[1]로 접근.
- 음의 정수를 이용한 인덱싱은 리스트의 마지막 원소에서부터 인덱싱을 함.
- 다시말해, x[-1]은 x의 마지막 원소, x[-2]는 마지막에서 두번째 원소를 의미함.

```
>>> y = (1, 2, 3, 4)
>>> y[-1]
4
>>> y[-2]
3
```

인덱싱과 슬라이싱 V

- 슬라이싱은 연속적인 원소를 추출.
- 슬라이싱은 x[start:end:stride] 형식으로 수행됨.
- start는 변수 x로부터 원소를 추출할 시작 위치, end는 끝나는 위치를 입력하게 되고 stride는 시작 위치부터 끝나는 위치까지 위치의 증가량을 의미함.
- 만약, 1, 3, 5번째 원소를 추출하고 싶다면 x[0:5:2]와 같이 작성.
- start와 end 그리고 stride는 생략이 가능하고 start를 생략하면 0으로 처리되고, end를 생략하면 리스트(혹은 튜플)의 길이로 처리 그리고 stride를 생략하면 1로 자동으로 처리됨.

```
>>> x = [2, 3, 1, 5, 4]
>>> x[0:5:2]
[2, 1, 4]
>>> x[1:4]
[3, 1, 5]
>>> x[3:]
[5, 4]
```

인덱싱과 슬라이싱 VI

- 인덱싱과 슬라이싱은 매우 유사하지만 변수에서 하나의 원소를 추출할 때 주의가 필요함.
- 다음 두가지 코드는 같은 변수에서 같은 원소 한개를 추출하였지만 추출된 변수의 형태가 다름.

```
>>> x[0]
2
>>> x[0:1]
[2]
```

- 첫번째 코드(인덱싱)은 변수에서 해당 원소 하나를 추출하였고 추출된 형태가 정수형인 것을 알 수 있음.
- 하지만, 두번째 코드(슬라이싱)은 똑같은 원소 2가 추출되었지만 리스트 구조가 유지되어 추출된 것을 알 수 있음.
- 인덱싱 혹은 슬라이싱 후, 해당 형태에 맞춰서 프로그램을 작성해야 오류를 피할 수 있음.

인덱싱과 슬라이싱 VII

딕셔너리에서 값 추출하기

- 딕셔너리는 key와 그에 해당하는 value로 짝지어진 객체였음.
- 딕셔너리 객체의 구조에서 알 수 있듯이 딕셔너리에서 값을 추출하는 것은 key를 통해 수행됨.
- 리스트와 튜플에서와 마찬가지로 대괄호에 key를 입력해서 key에 해당하는 값을 추출.

```
>>> y = {"first␣name" : ["Beomjin", "Sion", "Sangjun"],
         "last␣name" : ["Park", "Park", "Moon"]}
>>> y["first␣name"]
['Beomjin', 'Sion', 'Sangjun']
>>> y["last␣name"]
['Park', 'Park', 'Moon']
```

인덱싱과 슬라이싱 VIII

- 앞선 예제에서 알 수 있듯이 딕셔너리의 key에 해당하는 값이 리스트 등 다양한 객체일 수 있음.
- 따라서 key를 통해 딕셔너리에서 값을 추출하고 해당 값에서 다시 한번 인덱싱 혹은 슬라이싱 하는 것도 가능함.

```
>>> y["first␣name"][0]
'Beomjin'
>>> y["first␣name"][1:]
['Sion', 'Sangjun']
```

수정과 삭제 I

원소의 수정

- 변할수 있는 (mutable) 객체는 객체의 원소를 수정 및 삭제할 수 있음.
- 대표적인 mutable 객체로는 리스트와 딕셔너리가 있음.
- 반대로 immutable 객체 (예: 튜플)는 원소를 수정하거나 삭제하는 것이 불가능함.
- mutable 객체에서 원소를 수정하는 방법은 인덱싱 혹은 슬라이싱하는 코드에 "="를 통해 할당하면 해당 위치의 원소들이 할당하는 값으로 수정됨.

```
>>> x = [1, 5, 3, 4]
>>> x[1] = 2
>>> x
[1, 2, 3, 4]
>>> y = {"name" : [3, 2, 1]}
>>> y["name"] = [1, 2, 3]
>>> y
{'name' : [1, 2, 3]}
```

수정과 삭제 II

```
>>> x[1:3] = [7, 8]
>>> x
[1, 7, 8, 4]
>>> y["name"][0] = 10
>>> y
{'name' : [10, 2, 3]}
```

- 원소를 삭제하는 방법은 del 함수를 이용함.

```
>>> del x[1]
>>> x
[1, 8, 4]
>>> del x[1:]
>>> x
[1]
>>> y = {"first␣name" : ["Beomjin", "Sion", "Sangjun"],
         "last␣name" : ["Park", "Park", "Moon"]}
>>> del y["first␣name"]
>>> y
{'last␣name': ['Park', 'Park', 'Moon']}
```

객체의 함수 I

- 앞서 배운 자료형은 다양한 기능을 수행하는 함수(function) 및 메소드(method)를 적용할 수 있음.
- 함수와 메소드는 추후에 다룰 예정이며 이번 장에서는 함수와 메소드를 특별히 구분하지 않음.
- 각 객체에 해당하는 함수와 메소드들은 매우 다양하며 다 소개하지 않고 자주 사용되는 몇가지만을 소개함.

객체의 함수 II

리스트에 대한 함수

- len() 함수
  - len() 함수는 리스트의 길이를 반환해줌.
  - 또한 len() 함수는 리스트 뿐만 아니라 문자열의 길이, 튜플의 길이, 그리고 딕셔너리의 길이등을 반환해줌.

```
>>> x = [1, 3, 2]
>>> len(x)
3
>>> len("Park Beomjin")
12
```

- append() 메소드
  - append() 메소드는 리스트의 마지막에 함수에 입력되는 값이 추가됨.
  - append() 메소드는 리스트 x에 대해 x.append()로 실행됨.

```
>>> x = [1, 2, 3]
>>> x.append(4)
>>> x
[1, 2, 3, 4]
```

객체의 함수 III

- sort() 메소드
  - sort() 메소드는 리스트의 요소들을 순서대로 정렬해줌.
  - sort() 메소드내에 reverse 인자(argument)에 True를 입력하면 내림차순, False 혹은 reverse 인자를 입력하지 않는다면 오름차순으로 정렬됨.
  - 리스트의 sort() 메소드는 따로 값을 반환하지 않고 해당 객체 내에서 수행됨.

```
>>> x = [2, 3, 1, 5, 4]
>>> x.sort()
>>> x
[1, 2, 3, 4, 5]

>>> x.sort(reverse = True)
>>> x
[5, 4, 3, 2, 1]
```

객체의 함수 IV

- count() 메소드
  - count() 메소드는 입력되는 값이 리스트에서 몇개가 존재하는지 계산해줌.

```
>>> y = 'banana'
>>> y.count('a')
3
>>> x = [1, 1, 2, 3, 3, 3]
>>> x.count(1)
2
```

- 튜플에서도 len() 함수 및 sort() 함수 및 count() 메소드는 리스트에서와 동일하게 작동함.
- 하지만, append() 및 sort() 등의 메소드는 존재하지 않음.

객체의 함수 V

딕셔너리에 대한 함수

- keys() 메소드
  - 딕셔너리의 keys() 메소드는 해당 딕셔너리에 존재하는 key 들을 반환해줌.
  - keys() 메소드로 반환된 값은 리스트가 아닌 dict_keys라는 객체로 만약 리스트로 변환하고 싶을 때에는 list() 함수를 적용.
  - dict_keys 객체는 반복문등에서 직접 사용이 가능하지만 편의상 list() 함수를 적용해서 사용하는 경우도 많음.

```
>>> y = {"first␣name" : ["Beomjin", "Sion", "Sangjun"
         "last␣name" : ["Park", "Park", "Moon"]}
>>> y.keys()
dict_keys(['first␣name', 'last␣name'])
>>> list(y.keys())
['first␣name', 'last␣name']
```

객체의 함수 VI

- values() 메소드
  - values() 메소드는 해당 딕셔너리의 value를 반환해줌.
  - keys() 메소드와 유사하게 반환된 값은 dict_values 객체임.
  - list() 함수를 적용해서 리스트로 변환 가능함.

```
>>> y.values()
dict_values([['Beomjin', 'Sion', 'Sangjun'],
             ['Park', 'Park', 'Moon']])
>>> list(y.values())
[['Beomjin', 'Sion', 'Sangjun'],
 ['Park', 'Park', 'Moon']]
```

객체의 함수 VII

- items() 메소드
  - items() 메소드는 딕셔너리의 key와 value의 쌍을 튜플로 묶어 반환해줌.

```
>>> y.items()
dict_items([('first␣name', ['Beomjin', 'Sion', 'Sangj
            ('last␣name', ['Park', 'Park', 'Moon'])])
>>> list(y.items())
[('first␣name', ['Beomjin', 'Sion', 'Sangjun']),
 ('last␣name', ['Park', 'Park', 'Moon'])]
```

객체의 함수 VIII

- get() 메소드
  - get() 메소드는 key를 통해서 key에 해당하는 value를 반환해주는 메소드.

```
>>> y.get("first␣name")
['Beomjin', 'Sion', 'Sangjun']

>>> y.get("last␣name")
['Park', 'Park', 'Moon']
```