딥베이직

20161115 김단비

3. 파이 채우기: 리스트, 튜플, 딕셔너리, 셋

2장에서 부울, 정수, 부동소수점수, 문자열에 대한 기본 데이터 타입을 살펴봤다.
이들 데이터 타입이 원자라면 이번 장에서 다룰 자료구조(data structure)는 분자다.
대부분의 프로그래밍 과정은 데이터를 잘게 나누고 이들을 붙여서 특정한 형태로 만든다

3.1 리스트와 튜플

이전 장에서 문자열은 문자의 시퀀스라는 걸 배웠다. 리스트는 모든 것의 시퀀스임
파이썬에는 두 가지 다른 시퀀스 구조가 있는데, 튜플과 리스트임

파이썬은 왜 리스트와 튜플 두 가지를 포함하고 있을까?

튜플은 불변(immutable), 리스트는 변경 가능(mutable)하니까.
튜플은 항목을 할당하고 이를 바꿀 수 없음
리스트는 항목을 할당하고 자유롭게 수정하거나 삭제 가능

3.2 리스트
리스트는 데이터를 순차적으로 파악하는 데 유용
리스트의 현재 위치에서 새로운 요소를 추가하거나 삭제 혹은 기존 요소를 덮어쓸 수 있음
동일한 값이 여러 번 나타날 수 있음!
3.2.1 리스트 생성하기: [ ] 또는 list( )
리스트는 콤마(,)로 구분하고 대괄호([])로 둘러싸여 있음

In [2]:

```
empty_list = [ ]
weekdays = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday'] #리스트의 순서를 가
big_birds = ['emu', 'ostrich', 'cassowary']
first_names = ['Graham', 'John', 'Terry', 'Terry', 'Michael'] #값이 유일할 필요가 없다는
```

NOTE_요소들이 순서는 상관없이 유일한 값으로 유지될 필요가 있다면 리스트보다 셋(Set)을 사용하는 것이 더 좋음! 좀 이따 나와요

또한 list() 함수로 빈 리스트 할당 가능

In [1]:

```
another_empty_list = list()
another_empty_list
```

Out[1]:

```
[]
```

NOTE_4.6절 컴프리헨션에서는 리스트를 생성하는 다른 방법인 리스트 컴프리헨션을 설명한대요 다음에 와서 들으세요

3.2.2 다른 데이터 타입을 리스트로 변환하기: list( )

In [3]:

```
list('cat')
```

Out[3]:

```
['c', 'a', 't']
```

In [5]:

```
a_tuple = ('ready', 'fire', 'aim')
list(a_tuple)
```

Out[5]:

```
['ready', 'fire', 'aim']
```

In [6]:

```
birthday = '1/6/1952'
birthday.split('/')
```

Out[6]:

```
['1', '6', '1952']
```

In [7]:

```
splitme = 'a/b//c/d///e'
splitme.split('/')
```

Out[7]:

```
['a', 'b', '', 'c', 'd', '', '', 'e']
```

In [8]:

```
splitme = 'a/b//c/d///e'
splitme.split('//')
```

Out[8]:

```
['a/b', 'c/d', '/e']
```

3.2.3 [offset]으로 항목 얻기

In [9]:

```
marxes = ['Groucho', 'Chico', 'Harpo']
marxes[0]
```

Out[9]:

```
'Groucho'
```

In [10]:

```
marxes[1]
```

Out[10]:

```
'Chico'
```

In [11]:

```
marxes[2]
```

Out[11]:

```
'Harpo'
```

In [12]:

```
marxes[-1]
```

Out[12]:

```
'Harpo'
```

In [13]:

```
marxes[-2]
```

Out[13]:

```
'Chico'
```

In [14]:

```
marxes[-3]
```

Out[14]:

```
'Groucho'
```

리스트의 오프셋은 값을 할당한 위치에 맞게 입력하지 않으면 예외(에러)가 발생함

In [15]:

```
marxes[5]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call
 last)
<ipython-input-15-3e37954ddf51> in <module>()
----> 1 marxes[5]

IndexError: list index out of range
```

3.2.4 리스트의 리스트

리스트는 리스트 뿐만 아니라 다른 타입의 요소도 포함할 수 있음

In [16]:

```
small_birds = ['hummingbird', 'finch']
extinct_birds = ['dodo', 'passenger pigeon', 'Norwegian Blue']
carol_birds = [3, 'French hens', 2, 'turtledoves']
all_birds = [small_birds, extinct_birds, 'macaw', carol_birds]
```

In [17]:

```
all_birds
```

Out[17]:

```
[['hummingbird', 'finch'],
 ['dodo', 'passenger pigeon', 'Norwegian Blue'],
 'macaw',
 [3, 'French hens', 2, 'turtledoves']]
```

In [18]:

```
all_birds[0]
```

Out[18]:

```
['hummingbird', 'finch']
```

첫 번째 항목은 리스트. all_birds 리스트의 첫 번째 항목인 small_birds 리스트임

small_birds 리스트의 두 번째 항목을 다음과 같이 두 인덱스를 사용해서 추출 가능

In [20]:

```
all_birds[0][1]
```

Out[20]:

```
'finch'
```

[0]은 all_birds의 첫 번째 항목을 가리키고, [1]은 앞에서 가리킨 항목(small_birds)의 두 번째 항목을 가리킴

3.2.5 [offset]으로 항목 바꾸기

오프셋으로 항목을 얻어서 바꿀 수 있음

In [21]:

```
marxes = ['Groucho', 'Chico', 'Harpo']
marxes[2] = 'Wanda'
marxes
```

Out[21]:

```
['Groucho', 'Chico', 'Wanda']
```

3.2.6 슬라이스로 항목 추출하기

슬라이스를 사용해서 리스트의 서브시퀀스를 추출할 수 있음

In [24]:

```
marxes = ['Groucho', 'Chico', 'Harpo']
marxes[0:2]
```

Out[24]:

```
['Groucho', 'Chico']
```

리스트의 슬라이스 또한 리스트!
문자열과 마찬가지로 슬라이스에 스텝 사용 가능

In [25]:

```
marxes[::2]
```

Out[25]:

```
['Groucho', 'Harpo']
```

In [26]:

```
marxes[::-2]
```

Out[26]:

```
['Harpo', 'Groucho']
```

In [27]:

```
marxes[::-1]
```

Out[27]:

```
['Harpo', 'Chico', 'Groucho']
```

3.2.7 리스트의 끝에 항목 추가하기: append( )

In [28]:

```
marxes.append('Zeppo')
marxes
```

Out[28]:

```
['Groucho', 'Chico', 'Harpo', 'Zeppo']
```

3.2.8 리스트 병합하기: extend( ) 또는 +=

In [29]:

```
marxes = ['Groucho', 'Chico', 'Harpo', 'Zeppo']
others = ['Gummo', 'Karl']
marxes.extend(others)
marxes
```

Out[29]:

```
['Groucho', 'Chico', 'Harpo', 'Zeppo', 'Gummo', 'Karl']
```

+=로도 병합할 수 있음

In [30]:

```
marxes = ['Groucho', 'Chico', 'Harpo', 'Zeppo']
others = ['Gummo', 'Karl']
marxes += others
marxes
```

Out[30]:

```
['Groucho', 'Chico', 'Harpo', 'Zeppo', 'Gummo', 'Karl']
```

append()를 사용하면 항목을 병합하지 않고 others가 하나의 리스트로 추가됨

In [31]:

```
marxes = ['Groucho', 'Chico', 'Harpo', 'Zeppo']
others = ['Gummo', 'Karl']
marxes.append(others)
marxes
```

Out[31]:

```
['Groucho', 'Chico', 'Harpo', 'Zeppo', ['Gummo', 'Karl']]
```

marxes에는 4개의 문자열과 두 문자열의 리스트가 존재하게 됨

3.2.9 오프셋과 insert( )로 항목 추가하기

append() 함수는 단지 리스트의 끝에 항목을 추가하지만, insert()함수는 원하는 위치에 항목을 추가할 수 있음
오프셋 0은 시작 지점에 항목을 삽입하고, 리스트의 끝을 넘는 오프셋은 append()처럼 끝에 항목 추가

In [35]:

```
marxes = ['Groucho', 'Chico', 'Harpo', 'Zeppo']
marxes.insert(3, 'Gummo')
marxes
```

Out[35]:

```
['Groucho', 'Chico', 'Harpo', 'Gummo', 'Zeppo']
```

In [36]:

```
marxes.insert(10, 'Karl')
marxes
```

Out[36]:

```
['Groucho', 'Chico', 'Harpo', 'Gummo', 'Zeppo', 'Karl']
```

3.2.10 오프셋으로 항목 삭제하기: del

In [37]:

```
del marxes[-1]
marxes
```

Out[37]:

```
['Groucho', 'Chico', 'Harpo', 'Gummo', 'Zeppo']
```

NOTE_del은 리스트 함수가 아니라 파이썬 구문. marxes[-2].del을 수행할 수 없음

3.2.11 값으로 항목 삭제하기: remove( )

리스트에서 삭제할 항목의 위치를 모르는 경우, remove()와 값으로 그 항목 삭제 가능

In [38]:

```
marxes = ['Groucho', 'Chico', 'Harpo', 'Gummo', 'Zeppo']
marxes.remove('Gummo')
marxes
```

Out[38]:

```
['Groucho', 'Chico', 'Harpo', 'Zeppo']
```

3.2.12 오프셋으로 항목을 얻은 후 삭제하기: pop( )

pop()은 리스트에서 항목을 가져오는 동시에 그 항목을 삭제함
오프셋과 함께 pop()을 호출했다면 그 오프셋의 항목이 반환됨
pop(0)은 리스트의 head(시작)을 반환
pop() 혹은 pop(-1)은 리스트의 tail(끝) 반환

In [39]:

```
marxes = ['Groucho', 'Chico', 'Harpo', 'Zeppo']
marxes.pop()
```

Out[39]:

```
'Zeppo'
```

In [40]:

```
marxes
```

Out[40]:

```
['Groucho', 'Chico', 'Harpo']
```

In [41]:

```
marxes.pop(1)
```

Out[41]:

```
'Chico'
```

In [42]:

```
marxes
```

Out[42]:

```
['Groucho', 'Harpo']
```

3.2.13 값으로 항목 오프셋 찾기: index( )

항목 값의 리스트 오픈셋을 알고 싶다면 index()를 사용하면 된다.

In [43]:

```
marxes = ['Groucho', 'Chico', 'Harpo', 'Zeppo']
marxes.index('Chico')
```

Out[43]:

```
1
```

3.2.14 존재여부 확인하기: in

리스트에서 어떤 값의 존재를 확인하려면 in을 사용

In [44]:

```
marxes = ['Groucho', 'Chico', 'Harpo', 'Zeppo']
'Groucho' in marxes
```

Out[44]:

```
True
```

리스트에 같은 값이 여러 개 존재할 수 있음. 리스트에 값이 적어도 하나 존재하면 in은 True 반환

In [45]:

```
words = ['a', 'deer', 'a' 'female', 'deer']
'deer' in words
```

Out[45]:

```
True
```

3.2.15 값 세기: count( )

리스트에 특정 값이 얼마나 있는지 세기 위해서는 count()를 사용

In [46]:

```
marxes = ['Groucho', 'Chico', 'Harpo']
marxes.count('Harpo')
```

Out[46]:

```
1
```

3.2.16 문자열로 변환하기: join( )

2.3.10절에서 더 자세히 나와있으니 패스하려 했으나 예제 하나만 보고 갑시다

In [47]:

```
marxes = ['Groucho', 'Chico', 'Harpo']
', '.join(marxes)
```

Out[47]:

```
'Groucho, Chico, Harpo'
```

join( )은 split( )의 반대라고 생각하면 기억하기 쉬워요

In [48]:

```
friends = ['Harry', 'Hermione', 'Ron']
separator = ' * '
joined = separator.join(friends)
joined
```

Out[48]:

```
'Harry * Hermione * Ron'
```

In [49]:

```
separated = joined.split(separator)
separated
```

Out[49]:

```
['Harry', 'Hermione', 'Ron']
```

In [50]:

```
separated == friends
```

Out[50]:

```
True
```

3.2.17 정렬하기: sort( )

오프셋을 이용해서 리스트를 정렬할 때도 있지만 값을 이용해서 리스트를 정렬할 때도 있음
파이썬은 두 가지 함수를 제공함

- sort()는 리스트 자체를 내부적으로 정렬
- sorted()는 리스트의 정렬된 복사본을 반환

리스트의 항목이 숫자인 경우, 기본적으로 오름차순으로 정렬
문자열인 경우, 알파벳 순으로 정렬

In [51]:

```
marxes = ['Groucho', 'Chico', 'Harpo']
sorted_marxes = sorted(marxes)
sorted_marxes
```

Out[51]:

```
['Chico', 'Groucho', 'Harpo']
```

In [52]:

```
marxes
```

Out[52]:

```
['Groucho', 'Chico', 'Harpo']
```

In [53]:

```
marxes.sort()
```

In [54]:

```
marxes
```

Out[54]:

```
['Chico', 'Groucho', 'Harpo']
```

In [55]:

```
numbers = [2, 1, 4.0, 3]
numbers.sort()
numbers
```

Out[55]:

```
[1, 2, 3, 4.0]
```

In [56]:

```
numbers = [2, 1, 4.0, 3]
numbers.sort(reverse=True)
numbers
```

Out[56]:

```
[4.0, 3, 2, 1]
```

3.2.18 항목 개수 얻기: len( )

In [57]:

```
marxes = ['Groucho', 'Chico', 'Harpo']
len(marxes)
```

Out[57]:

```
3
```

3.2.19 할당: =, 복사: copy( )

한 리스트를 변수 두 곳에 할당했을 경우, 한 리스트를 변경하면 다른 리스트도 따라서 변경

In [58]:

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

Out[58]:

```
[1, 2, 3]
```

In [59]:

```
b=a
b
```

Out[59]:

```
[1, 2, 3]
```

In [60]:

```
a[0] = 'surprise'
a
```

Out[60]:

```
['surprise', 2, 3]
```

In [61]:

```
b
```

Out[61]:

```
['surprise', 2, 3]
```

b는 단지 같은 리스트 객체의 a를 참조함. 그러므로 a 혹은 b 리스트의 내용을 변경하면 두 변수 모두에 반영됨

In [62]:

```
b[0] = 'I hate surprises'
b
```

Out[62]:

```
['I hate surprises', 2, 3]
```

In [63]:

```
a
```

Out[63]:

```
['I hate surprises', 2, 3]
```

한 리스트를 새로운 리스트로 복사 가능!

- copy() 함수
- list() 변환 함수

- 슬라이스 [:]

In [64]:

```
a = [1,2,3]
b = a.copy()
c = list(a)
d = a[:]
```

b, c, d는 a의 복사본이고 자신만의 값을 가진 새로운 객체이므로 복사본 b, c, d를 바꾸더라도 원본 a에는 아무런 영향이 없음

In [65]:

```
a[0] = 'integer lists are boring'
a
```

Out[65]:

```
['integer lists are boring', 2, 3]
```

In [66]:

```
b
```

Out[66]:

```
[1, 2, 3]
```

In [67]:

```
c
```

Out[67]:

```
[1, 2, 3]
```

In [68]:

```
d
```

Out[68]:

```
[1, 2, 3]
```

3.3 튜플

리스트와 다르게 튜플은 불변, 튜플을 정의한 후에 추가, 삭제, 수정 불가

3.3.1 튜플 생성하기: ( )

In [ ]: