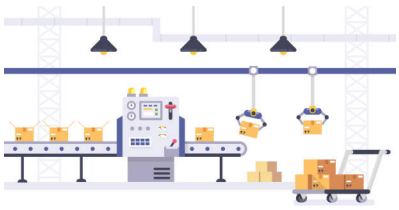
Lists, Comprehensions and Tuples

September 8, 2020



1 Resources:

- Official documentation: lists link
- Official documentation: list comprehensions link
- Official documentation: tuples link

2 Extra reading (Optional):

- Fluent Python by Luciano Ramalho -> Chapter 2
- Discusion on tuples vs lists article

In this course, we will talk about python's list and tuples.

We will look at the following :

- What we did in the previous lesson?
- What are lists?
- What problems can we solve with them?
- Comprehensions
- What are tuples?
- Difference between lists and tuples?
- Hobby Shop - Practical problem

3 Data Model - Recap

3.1 Lists

In python we have allready *build-in sequences* that are implemented in C. Sequences can be grouped by diferent criterias:

- Container or Flat sequences
  - Container: lists, tuple
  - Flat: str, bytearray
- Mutable or Immutable sequences
  - Mutable: lists, bytearray
  - Immutable: tuple, str

3.2 Core concepts

3.2.1 Example of list

[46]:
```
colors = ['blue', 'red', 'green', 'orange', 'yellow']
print(colors)
```

```
['blue', 'red', 'green', 'orange', 'yellow']
```

3.2.2 Storing multiple types

This kind of sequences can hold items of different types.

[47]:
```
multiple_types_list = [1, 'one', ['Hello', "World"], 2, 'two']
print(multiple_types_list)
```

```
[1, 'one', ['Hello', 'World'], 2, 'two']
```

3.2.3 Operators

[48]:
```
# in / not in
color = 'blue'
secound_color = 'pink'
print(color in colors)
print( secound_color not in colors)
```

```
True
True
```

[49]:
```
# +
my_shirts = ['pink', 'green']
```

```
brother_shirts = ['black', 'white']
closet = my_shirts + brother_shirts
print(closet)
```

```
['pink', 'green', 'black', 'white']
```

[50]:
```
# *
store_shirts = ['blue'] * 10
print(store_shirts)
```

```
['blue', 'blue', 'blue', 'blue', 'blue', 'blue', 'blue', 'blue', 'blue', 'blue']
```

3.2.4 Exercise:

Get the shirts from closet and store_shirts in on singel list (all_shirts)

---

[51]:
```
# Gett all the shirts in one list
all_shirts = []
```

3.2.5 Working with lists

- **getitem**

[52]:
```
first_element = all_shirts[0]
third_element = all_shirts[2]
fifth_element = all_shirts[4]
print(first_element, third_element, fifth_element)
```

```
␣
↪---------------------------------------------------------------------------

IndexError                                Traceback (most recent call␣
↪last)

<ipython-input-52-299b3046580e> in <module>
----> 1 first_element = all_shirts[0]
      2 third_element = all_shirts[2]
      3 fifth_element = all_shirts[4]
      4 print(first_element, third_element, fifth_element)
```

```
IndexError: list index out of range
```

- **Slice**

  slice(start,stop,step)

```
[ ]: # slice slice(star,stop,step)
print(all_shirts[:2])
print(all_shirts[4:])
print(all_shirts[::2])
```

3.2.6 Bonus:

```
[ ]: # Triks 1
print(all_shirts[-1])
```

```
[ ]: # Triks 2
print(all_shirts[::-1])
```

3.2.7 Usefull methods

- **len**

Get the lenght of the list

```
[ ]: # len
shirts_count = len(all_shirts)
print(shirts_count)
```

- **min** and **max**

```
[ ]: # min max
min_ = min(all_shirts)
max_ = max(all_shirts)

print(min_)
print(max_)
```

```
[ ]: shirt_sizes = [48, 49, 50, 51, 52, 53]
smallest = min(shirt_sizes)
largest = max(shirt_sizes)

print(smallest)
print(largest)
```

- **append**

Add element to the end of the list

[ ]:
```
all_shirts.append('green')
print(all_shirts)

all_shirts.append('yellow')
print(all_shirts)
```

- **insert**

Add element before the specified index

[ ]:
```
# insert -> add element before the specified index
all_shirts.insert(10, 'green')
```

3.3 Challenge:

Mimic the behaviour of append() with insert()

- **clear**

[ ]:
```
sold_shirts = all_shirts[5:10]
print(sold_shirts)

sold_shirts.clear()
print(sold_shirts)
```

- **pop**

Remove elemnt by index

[ ]:
```
print(all_shirts)

all_shirts.pop()
print(all_shirts)

all_shirts.pop(0)
print(all_shirts)
```

- **remove**

Remove element by value

[ ]:
```
print(all_shirts)

all_shirts.remove('green')
print(all_shirts)
```

3.3.1 Note:

'pop' is returning the removed value or a default remove is not returning the value and is trowing an exception if the item is not in list

---

- **extend**

Extend the list by appending all the items from the given iterable

[ ]:
```
all_shirts.extend(['white', 'black'])
print(all_shirts)
```

3.4 Homework:

Two list methods are not listed here. Using the python documentation find out the missing methods and play with then in order to figure out what they do.

---

4 Comprehensions

With list comprehensions, we can create and populate a list in a cleaner way. Most of the time when we work with lists we will use comprehensions.

[53]:
```
# example from official documentation
squares = []
for x in range(10):
    squares.append(x**2)

print(squares)
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

One of the most common use case of comprehensions is when we want to apply a certain operation to each element of a list

[54]:
```
squares = [x**2 for x in range(10)]
print(squares)
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

After the first for we can add multiple fors or ifs.

Advice: Complicated comprehensions are hard to read. Sometimes more code is better than unreadable code!

[55]:
```
# comprehensions with if
even = [x for x in range(10) if x % 2 == 0]
print(even)

# multiple for loops
unique_grid = [(x, y) for x in [1,2,3] for y in [3,1,4] if x != y]
print(unique_grid)
```

```
[0, 2, 4, 6, 8]
[(1, 3), (1, 4), (2, 3), (2, 1), (2, 4), (3, 1), (3, 4)]
```

4.1 Exercise:

Write unique_grid with regular for loops!

More complicated comprehension

[56]:
```
# list flatening
list_of_lists = [[1,2,3], [4,5,6], [7,8,9]]
[num for list_ in list_of_lists for num in list_]
```

[56]:
```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

5 Tuples

Tuples are containers and immutable sequences. Tuples are similar with lists but are used in different use cases based on the need

- **syntax**

[57]:
```
tuple_example_1 = 1, 2, 3, 4
print(tuple_example_1)

tuple_example_2 = (1, 2, 3, 4)
print(tuple_example_2)

tuple_example_3 = tuple([1, 2, 3, 4])
print(tuple_example_3)
```

```
(1, 2, 3, 4)
(1, 2, 3, 4)
(1, 2, 3, 4)
```

- **imutable**

[58]:
```
car = ('BMW', "X5", "black")
car[0] = 'Audi'
```

```
␣
↪---------------------------------------------------------------------------

TypeError                                 Traceback (most recent call␣
↪last)

<ipython-input-58-77282a25c155> in <module>
  1 car = ('BMW', "X5", "black")
----> 2 car[0] = 'Audi'

TypeError: 'tuple' object does not support item assignment
```

- **container** -> can hold multiple types

[ ]:
```
pizza = ('peperoni', ('pepperoni', 'motzzarella'), 28, 25.5)
```

- **tuple unpacking** and **tuple as records**

[ ]:
```
name, ingredients, size, price = pizza
print(f'Pizza {name} has {ingredients} it is {size}cm and costs {price}')
```

[ ]:
```
# use * to grab excess items
a,b, *rest = range(10)
print(a,b,rest)
```

[ ]:
```
a, *rest, b = range(10)
print(a,rest,b)
```

[ ]:
```
# tuple unpacking in for loop
for x, y in [(2,3),(4,5),(6,7)]:
    print(x**y)
```

5.1 Differance between lists and tuples

One of the biggest difference is that **tuples have structure, lists have order** this is a consequence of tuples being immutable.

	list	tuple	
s.__add__(s2)	•	•	s + s2 — concatenation
s.__iadd__(s2)	•		s += s2 — in-place concatenation
s.append(e)	•		append one element after last
s.clear()	•		delete all items
s.__contains__(e)	•	•	e in s
s.copy()	•		shallow copy of the list
s.count(e)	•	•	count occurrences of an element
s.__delitem__(p)	•		remove item at position p
s.extend(it)	•		append items from iterable it
s.__getitem__(p)	•	•	s[p] — get item at position
s.__getnewargs__()		•	support for optimized serialization with pickle
s.index(e)	•	•	find position of first occurrence of e
s.insert(p, e)	•		insert element e before the item at position p
s.__iter__()	•	•	get iterator
s.__len__()	•	•	len(s) — number of items
s.__mul__(n)	•	•	s * n — repeated concatenation
s.__imul__(n)	•		s *= n — in-place repeated concatenation
s.__rmul__(n)	•	•	n * s — reversed repeated concatenation[a]
s.pop(«p»)	•		remove and return last item or item at optional position p
s.remove(e)	•		remove first occurrence of element e by value
s.reverse()	•		reverse the order of the items in-place
s.__reversed__()	•		get iterator to scan items from last to first
s.__setitem__(p, e)	•		s[p] = e — put e in position p, overwriting existing item
s.sort(«key», «reverse»)	•		sort items in place with optional keyword arguments key and reverse

(Source: Fluent Python)

6 Homework:

Hobby Shop

Requirements:

- Have at least 400 articles in the shop
- Have at least four types of articles (shirt, scarf, glove, heat)
- Have at least five sizes (S M L XL XXL) for each type of article
- To be able to sell the latest article that was added to the shop

- To be able to sell any item that is in the shop
- To restock the shop with new items

The code should be uploaded on moodle:
