td3

January 26, 2023

1 TD3 Revision - basic data structure

Dans ce tp, vous apprendrez quelques structures de données de base et de méthodes de manipulation. Vous allez voir `string`, `liste`, et `disctionary`.

Si vous n'avez aucune difficulté avec ces concepts de base, vous pouvez aller directement à la partie du devoir.

1.1 1. What's in a string?

Strings are inherently an ordered sequence of characters. Which is why you can iterate over a string using a for-loop, or retrieve specific characters using a slice.

Before executing the next cell, have a guess at what the output will produce!

[ ]:

```
str_a = "I am a chimp. I love peanuts."
for char in str_a:
    print(char)

print(str_a[2:-2:2])
```

1.2 2. Split and join

Two crucial functions you should know about are `str.join()` and `str.split()`.

- `str.join()` links together a series of strings:
- `str.split()` breaks down a single string into a list of strings

[ ]:

```
print("A list of bare necessities: %s." % ", ".join(["peanut", "typewriter",␣
 ↪"peanut (important!)", "evil plans"]))

str_a = "I am a chimp. I love peanuts."
strings = str_a.split()
for s in strings:
    print(s)
```

1.2.1 Exercise 1: Sifting through many words

Implement a function `every_other_word(s)` that splits its argument string on spaces, joins every other item with an underscore and returns this transformed string. For instance:

```
>>> every_other_word("Figaro, that's a man who loves peanuts. But what about Bond? James Bond?"
'Figaro,_a_who_peanuts._what_Bond?_Bond?'
```

[ ]:

```python
def every_other_word(s):
    # put your code here
    pass

print(every_other_word("Figaro, that's a man who loves peanuts. But what about␣
 ↪Bond? James Bond?"))
```

1.3 3. List

Predict what the following lines of Python will do. Then, run the code block below to see if they match what you expect:

```python
s = [0] * 3
print(s)
s[0] += 1
print(s)

s = [''] * 3
print(s)
s[0] += 'a'
print(s)

s = [[]] * 3
print(s)
s[0] += [1]
print(s)
```

[ ]:

```python
# Explore the elements of lists. Is the output what you expect?
s = [0] * 3
print(s)
s[0] += 1
print(s)

s = [''] * 3
print(s)
s[0] += 'a'
print(s)

s = [[]] * 3
print(s)
s[0] += [1]
print(s)
```

1.4 4. Manipulating lists

Broadly speaking, there are three things you might want to do with lists:

1. add elements
2. remove elements
3. inspect elements

All of these can be done in a number of different ways. Here are a few main ones:

1. To **add** an element, you can use `my_list.append(elem)`. To add all the elements of some other sequence at once, you can try `my_list.extend(other_seq)`. Related to that, the addition operator `+` for lists corresponds to concatenation, and the multiplication operator `*` corresponds to duplication (remember how it went for strings?)
2. To **remove** elements by value, you can use `my_list.remove('some_value')`, which will delete the first occurrence of that value in your list. Alternatively, you can remove elements based on their index: `del my_list[idx]` will remove the value at index `idx`; `elem = my_list.pop(idx)` will remove the value at index `idx` and place it in the variable `elem` instead.
3. To **access** the value at a given index `idx`, we generally use indexing: `my_value = my_list[idx]`.

Try predicting what the code below will print!

[ ]:

```
chimps = (["chimp", "peanuts"] * 3)

chimps.remove("chimp")
del chimps[1]
something = chimps.pop(-2)

# what will this print?
print(something, chimps)
```

1.5 5. List slicing

Another common type of operation with lists (or *ordered* sequences in general) is to iterate over them. This is very frequently done with **slicing**:

```
>>> daily_articles = ['no peanut', 'one peanut', 'two peanuts', 'three peanuts', 'peanuts stol
>>> daily_articles[1:3]
['one peanut', 'two peanuts']
>>> daily_articles[1:-2]
['one peanut', 'two peanuts', 'three peanuts']
>>> daily_articles[::2]
['no peanuts', 'two peanuts', 'peanuts stolen from Bond']
>>> daily_articles[-2::-2]
['peanuts stolen from Bond', 'two peanuts', 'no peanuts']
>>> daily_articles[::-1]
['evil plan to destroy Paris', 'peanuts stolen from Bond', 'three peanuts', 'two peanuts', 'one
```

As you can see, `my_list[::-1]` traverses the list in reverse: it starts from the end. Another function you can use for that is `reversed()`.

1.5.1 Exercise 2: Sifting through

Using slices, write a function called `every_third(l)` that takes a list `l` as argument and returns every third element in the list.

```
>>> every_third([0, 1, 2, 3, 4, 5])
[2, 5]
```

Write a function called `first_and_last(l)` that returns a list containing only the first and last element of the argument list `l`.

```
>>> first_and_last([])
[]
>>> first_and_last([1])
[1, 1]
>>> first_and_last([1, 1])
[1, 1]
>>> first_and_last([1, 2])
[1, 2]
>>> first_and_last([1, 2, 3, 4, 5])
[1, 5]
```

[ ]:

```python
# write your code here
def every_third(l):
    # TODO

def first_and_last(l):
    # TODO
```

[ ]:

```python
# Uncomment for testing

# print(every_third([0, 1, 2, 3, 4, 5]))
# print(first_and_last([]))
# print(first_and_last([1]))
# print(first_and_last([1, 1]))
# print(first_and_last([1, 2]))
# print(first_and_last([1, 2, 3, 4, 5]))
```

1.6 6. Dictionaries

Dictionaries (the `dict` type in python) are mappings that associate keys to values.

Instead of using integers to index elements, as you would in a list, dictionaries allow you to use whatever value as a key.

As such, you can use `del my_dict[key]` to remove a certain `key`, `value` pair from a dictionary

The only two requirements for keys are that they need to be unique and hashable, i.e., immutable and composed only of immutable objects.

You can retrieve only the keys as an ordered sequence using the `dict.keys()` method. The same thing applies for values with `dict.values()`. To get pairs of keys associated to values, you can use `dict.items()`.

```
>>> d = {'chimp': 'peanut', 'Bond': 'James'}
>>> d.keys()
dict_keys(['chimp', 'Bond'])
>>> d.values()
dict_values(['peanut', 'James'])
>>> d.items()
dict_items([('chimp', 'peanut'), ('Bond', 'James')])
```

1.6.1 Exercise 3: Flip it!

Write a function that properly reverses the keys and values of a dictionary - each key (originally a value) should map to a collection of values (originally keys) that mapped to it. For example,

```
flip_dict({"CA": "US", "NY": "US", "ON": "CA"})
# => {"US": ["CA", "NY"], "CA": ["ON"]}
```

Note: there is a data structure in the `collections` module from the standard library called `defaultdict` which provides exactly this sort of functionality. You provide it a factory method for creating default values in the dictionary (in this case, a list.) You can read more about `defaultdict` and other `collections` data structures here.

[ ]:

```
import collections

def flip_dict(input_dict):
    """Reverse the keys and values of a dictionary."""
    pass

flip_dict({"CA": "US", "NY": "US", "ON": "CA"})
# should print {"US": ["CA", "NY"], "CA": ["ON"]}
```

1.6.2 If you have time

Already finished? Bravo! You can try the following exercise for fun.

Write list comprehensions to transform the input data structure into the output data structure:

```
[0, 1, 2, 3] -> [1, 3, 5, 7]  # Double and add one
['apple', 'orange', 'pear'] -> ['A', 'O', 'P']  # Capitalize first letter
['apple', 'orange', 'pear'] -> ['apple', 'pear']  # Contains a 'p'

["TA_sam", "student_poohbear", "TA_guido", "student_htiek"] -> ["sam", "guido"] # TA's names
['apple', 'orange', 'pear'] -> [('apple', 5), ('orange', 6), ('pear', 4)] # words and their le

['apple', 'orange', 'pear'] -> {'apple': 5, 'orange': 6, 'pear': 4} # words and their lengths
```

[ ]:

```
nums = [0, 1, 2, 3]
fruits = ['apple', 'orange', 'pear']
people = ["TA_sam", "student_poohbear", "TA_guido", "student_htiek"]

# An example
nums_doubled_and_incremented = [i*2 +1 for i in nums] # nums -> Double and add one
print(nums_doubled_and_incremented)

# Add your comprehensions here!
fruits_capitalized_first_letter = [] # fruits -> Capitalize first letter
fruits_cotaining_p = [] # fruits -> Contains a 'p'

people_TA_names = [] # people -> TA's names
fruits_word_and_length_tuples = [] # fruits -> words and their lengths in a list of tuples

fruits_word_to_length_dict = [] # fruits -> words and their lengths as dictionary key-value pairs
```

1.6.3 /! Submission instructions

You will need to submit this lab on Arche before 9:59am on Friday, 3rd February. Submit either a `.py` or an `.ipynb` file containing the functions you wrote for the 3 exercises and name it `td3_firstname_lastname.py` or `td3_firstname_lastname.ipynb` accordingly, where `firstname` should be your first name and `lastname` should be your last name (e.g. Jane Doe's submission should be called `td4_jane_doe.py` or `td4_jane_doe.ipynb`).