SW01: Recap of Python data structures

This notebook serves you to quickly refresh your Python knowledge.

If the expressions marked with `*ADVANCED*` are not entirely clear to you yet - don't worry! They require advanced knowledge of Python and are not central to this course.

Strings

In [1]:
```
# Assign a string to a variable.
a = "Hello world!"
print(a)
```

```
Hello world!
```

In [2]:
```
# Access elements of string by index/square brackets.
print(a[1])
```

```
e
```

In [3]:
```
# Strings are arrays and can be looped through.
for x in a:
  print(x)
```

```
H
e
l
l
o

w
o
r
l
d
!
```

In [4]:
```
# Convert the characters in a string:
# - upper(): upper case
# - lower(): lower case
# - capitalize(): capitalize the first character of the string
# - title(): capitalize the first character of each word in the string
print(a.upper())
print(a.lower())
print(a.capitalize())
print(a.title())
```

```
HELLO WORLD!
hello world!
Hello world!
Hello World!
```

In [5]:
```
# Check whether a certain phrase or character is present in a string.
text = "Machine learning is cool!"
print("cool" in text)
```

```
True
```

In [6]:
```
# The strip() method removes any whitespace from the beginning or the end.
a = "   Hello, World!   "
print(a)
print(a.strip())
```

```
   Hello, World!
Hello, World!
```

In [7]:
```
# The split() method splits the string into substrings if it finds
# instances of the separator specified, and returns a list of the substrings.
a = "Hello,   World! "
print(a)
print(a.split(","))
```

```
Hello,   World!
['Hello', '   World! ']
```

In [8]:
```
# In the above solution, you might dislike the whitespaces in the substrings.
# We can improve the solution by using the strip() method on each substring.

# Using a for loop...
ret = []
for s in a.split(","):
    ret.append(s.strip())
print(ret)

# Using a list comprehension...
print([s.strip() for s in a.split(",")])

# Using the map() function... (*ADVANCED*)
print(list(map(str.strip, a.split(","))))
```

```
['Hello', 'World!']
['Hello', 'World!']
['Hello', 'World!']
```

Lists

In [9]:
```
# Create a list of strings.
list_names = ['tom', 'jerry', 'spike', 'tyke']
print(list_names)
```

```
['tom', 'jerry', 'spike', 'tyke']
```

In [10]:
```
# Access an element by index/square brackets.
print(list_names[1])
```

```
jerry
```

In [11]:
```
# Negative indexing reverses the indexing.
print(list_names[-1])
```

```
tyke
```

In [12]:
```
# Access range of values by specifying range of indexes.
print(list_names[1:3])
```

```
['jerry', 'spike']
```

In [13]:
```
# Add an item to the end of the list.
list_names.append("mike")
print(list_names)
```

```
['tom', 'jerry', 'spike', 'tyke', 'mike']
```

In [14]:
```
# Add an item at a specified index.
idx = 0
list_names.insert(idx, "pocahontas")
print(list_names)
```

```
['pocahontas', 'tom', 'jerry', 'spike', 'tyke', 'mike']
```

In [15]:
```
# You can loop through the list items by using a for loop.
for x in list_names:
    print(x)
```

```
pocahontas
tom
jerry
spike
tyke
mike
```

In [16]:
```
# Create a new list based on the values of an existing list.
new_list = []
for x in list_names:
    if "y" in x:
        new_list.append(x)
print(new_list)
```

```
['jerry', 'tyke']
```

In [17]:
```
# List comprehension offers a short-cut (and is even faster).
new_list = [x for x in list_names if "y" in x]
print(new_list)
```

```
['jerry', 'tyke']
```

Dictionaries

In [18]:
```
# Dictionaries are used to store data values in key:value pairs.
dct = {
  "title": "The Lion King",
  "year": 1994,
  "duration": 88
}
print(dct)
```

```
{'title': 'The Lion King', 'year': 1994, 'duration': 88}
```

In [19]:
```
# Add an item to the dictionary using a new key-value pair.
dct["language"] = "English"
print(dct)
print(dct["language"])
```

```
{'title': 'The Lion King', 'year': 1994, 'duration': 88, 'language': 'English'}
English
```

In [20]:
```
# Retrieve the keys and values of the dictionary (as an iterable).
print(dct.keys())
print(dct.values())

# *ADVANCED* Technically, the keys() and values() methods return view objects
# that do not behave like a list. But one can convert them easily to a list.
print(list(dct.keys()))
print(list(dct.values()))

dict_keys(['title', 'year', 'duration', 'language'])
dict_values(['The Lion King', 1994, 88, 'English'])
['title', 'year', 'duration', 'language']
['The Lion King', 1994, 88, 'English']
```

In [21]:
```
# Attempting to access a non-existing key will raise a KeyError.
dct['actor']
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In[21], line 2
      1 # Attempting to access a non-existing key will raise a KeyError.
----> 2 dct['actor']

KeyError: 'actor'
```

In [22]:
```
# Get all the key:value pairs, as tuples in an iterable.
print(dct.items())

# In some situations, we need to convert this iterable to a list.
print(list(dct.items()))

dict_items([('title', 'The Lion King'), ('year', 1994), ('duration', 88), ('language', 'English')])
[('title', 'The Lion King'), ('year', 1994), ('duration', 88), ('language', 'English')]
```

In [23]:
```
# Create a dictionary from two lists, using zip().
keys = ["a", "b"]
values = [1, 2]
new_dict = dict(zip(keys, values))
print(new_dict)

# The zip() function takes an iterable-such as a list, tuple, set, or dictionary-as an argument.
# The function will generate a list of tuples that contain elements from each iterable.
```

```
{'a': 1, 'b': 2}
```

In [24]:
```
# Create a dictionary from two lists, using dictioanry comprehension.
keys = ['a', 'b', 'c']
values = [1, 2, 3]
dct = {k: v for k, v in zip(keys, values)}
```

In [24]:
```
# *ADVANCED* We used the zip() function to combine two (or more) lists to
# generate a sequence of tuples where the i-th tuple contains the i-th
# element from each of the input lists. Note that the zip() function will
# stop generating tuples when the shortest input iterable is exhausted.
print(zip(range(7), "abcd"))

# Well, the zip() function returns a generator. To actually see the tuples,
# we need to convert the generator to a list.
list(zip(range(7), "abcd"))
```

```
<zip object at 0x126803c40>
```

Out[24]:
```
[(0, 'a'), (1, 'b'), (2, 'c'), (3, 'd')]
```

Sets

In [25]:
```
# Sets are containers that do not allow duplicate elements.
# Sets are unordered, so the elements may appear in a random order.
a = {1, 1, 2, 3, 3, 4, 5, 5, 5, 5}
print(a)
```

```
{1, 2, 3, 4, 5}
```

In [26]:
```
# Sets are particularly useful to remove duplicates from a list.
b = [1, 1, 2, 3, 3, 4, 5, 5, 5, 5]
print(set(b))
```

```
{1, 2, 3, 4, 5}
```