06. Methods on known data types

September 30, 2025

1 6. Methods on known data types

When we discussed Python data types, we (purposely) missed a big part of their behavior: their methods. Methods are functions associated with objects. They are functions, which means they accept parameters and return values. But, they are associated with objects, which means we need an object *on* which we call them. The general syntax for a method call is:

```
obj.method(arguments)
```

In order to see all methods supported by an object, you can call:

```
dir(obj)
# or
dir(cls)
```

In order to see the documentation for a method, you can call:

```
help(obj.method)
# or
help(cls.method)
```

We will discuss next some commonly used methods for strings, lists and tuples.

1.1 6.1 String methods

String objects support many methods. We will present here some of the most commonly used string methods, grouped by the type of operation they perform.

Finding substrings:

```
[1]: greeting = 'hello world'
     greeting.find('wo')  # index of the first occurence of the substring
```

```
[1]: 6
```

```
[2]: greeting.startswith('h')  # checks if string starts with argument
```

```
[2]: True
```

```
[3]: greeting.endswith(' ')  # checks if string ends with argument
```

```
[3]: False
```

```
[4]: greeting.count('o')  # counts the number of occurrences of substring
```

```
[4]: 2
```

Splitting and joining:

```
[5]: greeting = 'hello world'
     words = greeting.split(' ')
     print(words)
```

```
['hello', 'world']
```

```
[6]: new_greeting = ', '.join(words)
     print(new_greeting)
```

```
hello, world
```

Verifying the string nature:

```
[7]: 'HELLO'.isupper()
```

```
[7]: True
```

```
[8]: 'hello'.islower()
```

```
[8]: True
```

```
[9]: 'Hello'.isalpha()
```

```
[9]: True
```

```
[10]: 'Hello123'.isalnum()
```

```
[10]: True
```

```
[11]: '\n\r\t '.isspace()
```

```
[11]: True
```

```
[12]: '2052'.isdigit()
```

```
[12]: True
```

Removing leading/trailing characters (by default whitespaces):

```
[13]: greeting = ' \thello\n'
      greeting.strip()  # strip any whitespace (space, newline, tab) from the␣
       ↪beginning and end of the string
```

```
[13]: 'hello'
```

[14]:
```python
greeting.lstrip()  # left strip, strips only at the beginning
```

[14]:
```
'hello\n'
```

[15]:
```python
greeting.rstrip()  # right string, strips only at the end
```

[15]:
```
' \thello'
```

[16]:
```python
greeting.strip("\n")  # if an argument is passed, strips only characters in␣
 ↪argument
```

[16]:
```
' \thello'
```

Changing characters:

[17]:
```python
greeting = 'hello WoRLD'
greeting.replace('o', 'ooo')
```

[17]:
```
'hellooo WoooRLD'
```

[18]:
```python
greeting.upper()
```

[18]:
```
'HELLO WORLD'
```

[19]:
```python
greeting.lower()
```

[19]:
```
'hello world'
```

[20]:
```python
greeting.capitalize()
```

[20]:
```
'Hello world'
```

Aligning text:

[21]:
```python
greeting = 'hello world'
greeting.rjust(20)
```

[21]:
```
'         hello world'
```

[22]:
```python
greeting.ljust(20, '_')
```

[22]:
```
'hello world_________'
```

[23]:
```python
greeting.center(20, '-')
```

[23]:
```
'----hello world-----'
```

1.2 Exercises 6.1

1. Write a function `get_words` which receives a string as a parameter and does the following:
   - removes punctuation marks (`,`, `.`, `!` and `?`)

- transforms everything to lowercase
- splits the string by space
- returns the list of words

Call the function on a sentence like `One warm, sunny day Jessica and Lilly went to the Zoo. When they arrived, they visited the monkeys.`

2. Write a function called `validate_password` that validates a password based on certain criteria:
   - the password has at least 8 characters
   - the password contains at least one uppercase letter
   - the password contains at least one lowercase letter
   - the password contains at least one digit.

   If all criteria are met, return `True`; otherwise, return `False`.

1.3 6.2. List methods

`a.append(x)` * add an item to the end of the list * equivalent to `a[len(a):] = [x]`

[1]:
```
odd_numbers = list(range(1, 20, 2))
odd_numbers
```

[1]:
```
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19]
```

[2]:
```
odd_numbers.append(200)
odd_numbers
```

[2]:
```
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 200]
```

`a.extend(L)`
* extend the list by appending all the items in the given list * equivalent to `a[len(a):] = L`

[25]:
```
odd_numbers.extend([0, 0, 0])
odd_numbers
```

[25]:
```
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 200, 0, 0, 0]
```

`a.insert(i, x)`
* insert an item at a given position * `a.insert(0, x)` inserts at the front of the list * `a.insert(len(a), x)` is equivalent to `a.append(x)`

[26]:
```
odd_numbers.insert(1, 'apple')
odd_numbers
```

[26]:
```
[1, 'apple', 3, 5, 7, 9, 11, 13, 15, 17, 19, 200, 0, 0, 0]
```

`a.remove(x)` * remove the first item from the list whose value is x * raises an error if no such item exists

[27]:
```
odd_numbers.remove(13)
odd_numbers
```

[27]:
```
[1, 'apple', 3, 5, 7, 9, 11, 15, 17, 19, 200, 0, 0, 0]
```

`a.pop()` / `a.pop(i)` * remove the item at the given position in the list, and return it * if no index is specified, `a.pop()` removes and returns the last item in the list

[28]:
```
second_element = odd_numbers.pop(1)
second_element
```

[28]:
```
'apple'
```

[29]:
```
last_element = odd_numbers.pop()
last_element
```

[29]:
```
0
```

[30]:
```
odd_numbers
```

[30]:
```
[1, 3, 5, 7, 9, 11, 15, 17, 19, 200, 0, 0]
```

`a.sort()` * sort the items of the list, in place.

[31]:
```
odd_numbers.sort()
odd_numbers
```

[31]:
```
[0, 0, 1, 3, 5, 7, 9, 11, 15, 17, 19, 200]
```

`a.reverse()` * reverse the elements of the list, in place

[32]:
```
odd_numbers.reverse()
odd_numbers
```

[32]:
```
[200, 19, 17, 15, 11, 9, 7, 5, 3, 1, 0, 0]
```

`a.index(x)` * return the index in the list of the first item whose value is `x` * it raises an error if there is no such item

[33]:
```
odd_numbers.index(19)
```

[33]:
```
1
```

`a.count(x)` * return the number of times `x` appears in the list

[34]:
```
odd_numbers.count(0)
```

[34]:
```
2
```

`a.clear()` * removes all items from `a` * same as `del a[:]`

[35]:
```
odd_numbers.clear()
odd_numbers
```

```
[35]: []
```

`a.copy()` * creates a shallow copy of `a` * same as `a[:]`

```
[36]: odd_numbers.copy()
```

```
[36]: []
```

1.4 Exercises 6.2

1. Write a function that takes a list of strings and an integer `min_length` (optional) as parameters and returns the list of strings longer than `min_length`. By default (when `min_length` not given), it should return the original list.

   E.g.

   ```
   filter_strings(["hello", "", "hi", "bye"], min_length=2)  # returns ['hello', 'bye']
   filter_strings(["hello", "", "hi", "bye"])  # returns ["hello", "", "hi", "bye"]
   ```

   ```
   words = ["hello", "", "hi", "bye"]
   min_length = 2
   long_words = []
   ```

2. The `remove` method removes the first occurrence of a value in a list. Write a function that removes all occurrences of a certain value.

3. [optional] Write a function `strip_all` that receives two parameters: a list of strings `strings` (required) and a string `chars` (optional), and returns a list containing all items in `strings` with leading and trailing characters in `chars` removed. If `chars` is not given, it should strip whitespaces around strings.

   E.g.

   ```
   strip_all(["\nAnna", "Jane ", "  Mike\n"])  # returns ["Anna", "Jane", "Mike"]
   strip_all(["Anna;", "Jane,", ",Mike"], ",;")  # returns ["Anna", "Jane", "Mike"]
   ```

1.5 6.3. Tuple methods

Because tuples are the immutable counterpart of lists, they support only *read* operations.

`t.count(x)` * return the number of times `x` appears in the tuple

```
[37]: my_tuple = (100, 20, 30, 100, 0)
      my_tuple.count(100)
```

```
[37]: 2
```

`t.index(x)` * return the index in the tuple of the first item whose value is `x` * it raises an error if there is no such item

```
[38]: my_tuple.index(20)
```

```
[38]: 1
```