.APPEND: method for adding an item to the end of the *list*
BREAK: clause used to break out of a loop prematurely, execution will continue after the entire loop structure (*while* and *for* loops)
CLASS = a definition
CONDITIONALS: if | elif | else
CONTINUE: clause used to shortcut a loop and start it again as if it had reached the end of its body of code (*while* and *for* loops)
DICT METHODS (cfr.)
EXPRESSION = a unit of evaluation; an expression is any combination of literals, identifiers, and operators. Generally, this means *anything that returns a value is an expression*
FOR LOOP: iterates over a sequence and the body of the loop is executed for each element of the sequence and until the sequence is exhausted
.FORMAT() is a method of the string object Ex.1 print('Hello world {}'.format(x)) Ex. 2 print(f'Hello world {x}')
FUNCTION CALLS: all function calls return a value
ISINSTANCE() : for debugging by type Ex. x=42.0 y=isinstance(x, int) -> False
JSON.DUMPS per convertire un dizionario x in JSON: y = json.dumps(x)
LIST COMPREHENSION = a list created based on another list or iterator
LIST METHODS (cfr.)
LOOPS: while | for
METHOD = a function that is associated with a class.
OBJECT = instance of a class. An object is created by calling the class as if it were a function, and the *constructor* is used to initialize the object.
RANGE(): this function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.
Sintax: range(start, stop, step)
STATEMENT = a unit of execution; a line of code
STRING METHODS (cfr.): returns new values; they do not change the original string
TUPLE METHODS (cfr.)
TRY-EXCEPT with sys.exc_info() : import sys <- informs about the kind of error
WHILE LOOP: tests a conditional expression; the body of the loop is executed while the condition remains true

# Python Collections (Arrays)

There are four collection data types in the Python programming language:

- **List** is a collection which is ordered and changeable. Allows duplicate members.
- **Tuple** is a collection which is ordered and unchangeable. Allows duplicate members.
- **Set** is a collection which is unordered and unindexed. No duplicate members.
- **Dictionary** is a collection which is unordered, changeable and indexed. No duplicate members.

| **Dict Methods** | Description |
|---|---|
| clear() | Removes all the elements from the dictionary |
| copy() | Returns a copy of the dictionary |
| fromkeys() | Returns a dictionary with the specified keys and values |
| get() | Returns the value of the specified key |
| items() | Returns a list containing the a tuple for each key value pair |
| keys() | Returns a list containing the dictionary's keys |
| pop() | Removes the element with the specified key |
| popitem() | Removes the last inserted key-value pair |
| setdefault() | Returns the value of the specified key. If the key does not exist: insert the key, with the specified value |
| update() | Updates the dictionary with the specified key-value pairs |
| values() | Returns a list of all the values in the dictionary |

| **FILE Methods** | Description |
|---|---|
| `open()` function | `"r"` - Read - Default value. Opens a file for reading, error if the file does not exist<br>`"a"` - Append - Opens a file for **appending**, creates the file if it does not exist<br>`"w"` - Write - Opens a file for **writing**, creates the file if it does not exist<br>`"x"` - Create - Creates the specified file, returns an error if the file exists |
| `.close()` | Close the file when you have finished with it |
| `.read()` | Return the file content |
| `.readline()` | Return one line |
| `.writelines()` | copy each line of a file in the new file (using a for/loop) |
| `os.remove()` function | `import os` - To delete a file |

| **LIST Methods** | Description |
|---|---|
| append() | Adds an element at the end of the list |
| clear() | Removes all the elements from the list |
| copy() | Returns a copy of the list |
| count() | Returns the number of elements with the specified value |
| extend() | Add the elements of a list (or any iterable), to the end of the current list |
| index() | Returns the index of the first element with the specified value |
| insert() | Adds an element at the specified position |
| pop() | Removes the element at the specified position |
| remove() | Removes the item with the specified value |
| reverse() | Reverses the order of the list |
| sort() | Sorts the list |

| **STRING Methods** | Description |
|---|---|
| .capitalize() | Converts the first character to upper case - *Ex. Hello World -> Hello world* |
| casefold() | Converts string into lower case; stronger than lower() - *Ex. Hällo World -> hallo world* |
| center() | Returns a centered string |
| count() | Returns the number of times a specified value occurs in a string |
| encode() | Returns an encoded version of the string |
| endswith() | Returns true if the string ends with the specified value |
| expandtabs() | Sets the tab size of the string |
| find() | Searches the string for a specified value and returns the position of where it was found |
| format() | Formats specified values in a string |
| format_map() | Formats specified values in a string - *Ex. print('{}'.format(7*6)) /-> 42* |
| index() | Searches the string for a specified value and returns the position of where it was found |
| isalnum() | Returns True if all characters in the string are alphanumeric |
| isalpha() | Returns True if all characters in the string are in the alphabet |
| isdecimal() | Returns True if all characters in the string are decimals |
| isdigit() | Returns True if all characters in the string are digits |
| isidentifier() | Returns True if the string is an identifier |
| islower() | Returns True if all characters in the string are lower case |
| isnumeric() | Returns True if all characters in the string are numeric |
| isprintable() | Returns True if all characters in the string are printable |
| isspace() | Returns True if all characters in the string are whitespaces |
| istitle() | Returns True if the string follows the rules of a title |
| isupper() | Returns True if all characters in the string are upper case |
| join() | Joins the elements of an iterable to the end of the string - *Ex. s='Hello World' s2=':'.join(s) -> Hello:world* |
| ljust() | Returns a left justified version of the string |
| lower() | Converts a string into lower case - *Ex. Hello World -> hello world* |
| lstrip() | Returns a left trim version of the string |
| maketrans() | Returns a translation table to be used in translations |
| partition() | Returns a tuple where the string is parted into three parts |
| replace() | Returns a string where a specified value is replaced with a specified value |
| rfind() | Searches the string for a specified value and returns the last position of where it was found |
| rindex() | Searches the string for a specified value and returns the last position of where it was found |
| rjust() | Returns a right justified version of the string |
| rpartition() | Returns a tuple where the string is parted into three parts |
| rsplit() | Splits the string at the specified separator, and returns a **list** - *Syntax: string.rsplit(separator, max)* |
| rstrip() | Returns a right trim version of the string |
| split() | Splits the string at the specified separator, and returns a list - *Ex. Hello World ->['Hello', 'World']* |
| splitlines() | Splits the string at line breaks and returns a list |
| startswith() | Returns true if the string starts with the specified value |
| strip() | Returns a trimmed version of the string |
| swapcase() | Swaps cases, lower case becomes upper case and vice versa - *Ex. Hello World -> hELLO wORLD* |
| title() | Converts the first character of each word to upper case - *Ex. hello world -> Hello World* |
| translate() | Returns a translated string |
| upper() | Converts a string into upper case |
| zfill() | Fills the string with a specified number of 0 values at the beginning |

| **TUPLE Methods** | Description |
|---|---|
| count() | Returns the number of times a specified value occurs in a tuple |
| index() | Searches the tuple for a specified value and returns the position of where it was found |