Python Additional Methods and Functions Documentation

1. String Methods

`str.capitalize()`

- **Description**: Capitalizes the first character of the string.
- **Usage**: `string.capitalize()`

`str.casefold()`

- **Description**: Returns a casefolded version of the string, suitable for caseless matching.
- **Usage**: `string.casefold()`

`str.center(width[, fillchar])`

- **Description**: Centers the string in a field of a given width, optionally filling with the specified character.
- **Usage**: `string.center(10, '-')`

`str.count(sub[, start[, end]])`

- **Description**: Counts non-overlapping occurrences of a substring within a string.
- **Usage**: `string.count('sub')`

`str.endswith(suffix[, start[, end]])`

- **Description**: Checks if the string ends with the specified suffix.
- **Usage**: `string.endswith('.txt')`

`str.find(sub[, start[, end]])`

- **Description**: Finds the lowest index where the substring is found in the string.
- **Usage**: `string.find('sub')`

`str.format(*args, **kwargs)`

- **Description**: Formats the string using placeholders.
- **Usage**: `"Hello, {}!".format('world')`

`str.index(sub[, start[, end]])`

- **Description**: Similar to `find()`, but raises a `ValueError` if the substring is not found.
- **Usage**: `string.index('sub')`

`str.isalnum()`

- **Description**: Checks if the string is alphanumeric.
- **Usage**: `string.isalnum()`

`str.isalpha()`

- **Description**: Checks if the string is alphabetic.
- **Usage**: `string.isalpha()`

`str.isdecimal()`

- **Description**: Checks if the string is a decimal number.
- **Usage**: `string.isdecimal()`

`str.isdigit()`

- **Description**: Checks if the string contains only digits.
- **Usage**: `string.isdigit()`

`str.islower()`

- **Description**: Checks if all cased characters in the string are lowercase.
- **Usage**: `string.islower()`

`str.isnumeric()`

- **Description**: Checks if the string contains only numeric characters.
- **Usage**: `string.isnumeric()`

`str.isspace()`

- **Description**: Checks if the string contains only whitespace characters.
- **Usage**: `string.isspace()`

`str.istitle()`

- **Description**: Checks if the string is titlecased.
- **Usage**: `string.istitle()`

`str.isupper()`

- **Description**: Checks if all cased characters in the string are uppercase.
- **Usage**: `string.isupper()`

`str.join(iterable)`

- **Description**: Concatenates the elements of an iterable into a single string with the string as a separator.
- **Usage**: `','.join(['a', 'b', 'c'])`

`str.ljust(width[, fillchar])`

- **Description**: Left-justifies the string in a field of a given width.
- **Usage**: `string.ljust(10, '-')`

`str.lower()`

- **Description**: Converts all cased characters to lowercase.
- **Usage**: `string.lower()`

`str.lstrip([chars])`

- **Description**: Removes leading characters (space by default).
- **Usage**: `string.lstrip(' ')`

`str.partition(sep)`

- **Description**: Splits the string at the first occurrence of sep and returns a 3-tuple.
- **Usage**: `string.partition(' ')`

`str.replace(old, new[, count])`

- **Description**: Replaces all occurrences of the substring old with new.
- **Usage**: `string.replace('old', 'new')`

`str.rfind(sub[, start[, end]])`

- **Description**: Finds the highest index where the substring is found in the string.
- **Usage**: `string.rfind('sub')`

`str.rindex(sub[, start[, end]])`

- **Description**: Like `rfind()` but raises `ValueError` when the substring is not found.
- **Usage**: `string.rindex('sub')`

`str.rjust(width[, fillchar])`

- **Description**: Right-justifies the string in a field of a given width.
- **Usage**: `string.rjust(10, '-')`

`str.rpartition(sep)`

- **Description**: Splits the string at the last occurrence of sep, returns a 3-tuple.
- **Usage**: `string.rpartition(' ')`

`str.rsplit(sep=None, maxsplit=-1)`

- **Description**: Splits the string from the right at sep.
- **Usage**: `string.rsplit(' ', 1)`

`str.rstrip([chars])`

- **Description**: Removes trailing characters (space by default).
- **Usage**: `string.rstrip(' ')`

`str.split(sep=None, maxsplit=-1)`

- **Description**: Splits the string at sep, returning a list.
- **Usage**: `string.split(' ')`

`str.splitlines([keepends])`

- **Description**: Splits the string at line breaks.
- **Usage**: `string.splitlines()`

`str.startswith(prefix[, start[, end]])`

- **Description**: Checks if the string starts with the specified prefix.
- **Usage**: `string.startswith('prefix')`

`str.strip([chars])`

- **Description**: Removes leading and trailing characters (space by default).
- **Usage**: `string.strip(' ')`

`str.swapcase()`

- **Description**: Converts uppercase characters to lowercase and vice versa.
- **Usage**: `string.swapcase()`

`str.title()`

- **Description**: Converts the string to title case.
- **Usage**: `string.title()`

`str.upper()`

- **Description**: Converts all cased characters to uppercase.
- **Usage**: `string.upper()`

`str.zfill(width)`

- **Description**: Pads the string on the left with zeros to fill the given width.
- **Usage**: `string.zfill(10)`

---

2. List Methods

`list.append(x)`

- **Description**: Adds an item to the end of the list.
- **Usage**: `list.append(10)`

`list.clear()`

- **Description**: Removes all items from the list.
- **Usage**: `list.clear()`

`list.copy()`

- **Description**: Returns a shallow copy of the list.
- **Usage**: `new_list = list.copy()`

`list.count(x)`

- **Description**: Returns the number of occurrences of `x` in the list.
- **Usage**: `list.count(10)`

`list.extend(iterable)`

- **Description**: Extends the list by appending all elements from the iterable.
- **Usage**: `list.extend([1, 2, 3])`

`list.index(x[, start[, end]])`

- **Description**: Returns the index of the first occurrence of `x`.
- **Usage**: `list.index(10)`

`list.insert(i, x)`

- **Description**: Inserts an item at a given position.
- **Usage**: `list.insert(1, 10)`

`list.pop([i])`

- **Description**: Removes and returns the item at the given position.
- **Usage**: `list.pop(1)`

`list.remove(x)`

- **Description**: Removes the first item from the list that has a value of `x`.
- **Usage**: `list.remove(10)`

`list.reverse()`

- **Description**: Reverses the elements of the list in place.
- **Usage**: `list.reverse()`

`list.sort(key=None, reverse=False)`

- **Description**: Sorts the items of the list in place.
- **Usage**: `list.sort()`

---

3. Set Methods

`set.add(x)`

- **Description**: Adds an element to the set.
- **Usage**: `set.add(10)`

`set.clear()`

- **Description**: Removes all elements from the set.
- **Usage**: `set.clear()`

`set.copy()`

- **Description**: Returns a shallow copy of the set.
- **Usage**: `new_set = set.copy()`

`set.difference(*others)`

- **Description**: Returns the difference of the set and other sets as a new set.
- **Usage**: `set.difference(other_set)`

`set.difference_update(*others)`

- **Description**

: Removes all elements of another set from this set.

- **Usage**: `set.difference_update(other_set)`

`set.discard(x)`

- **Description**: Removes an element from the set if it is a member.
- **Usage**: `set.discard(10)`

`set.intersection(*others)`

- **Description**: Returns the intersection of the set and other sets as a new set.
- **Usage**: `set.intersection(other_set)`

`set.intersection_update(*others)`

- **Description**: Updates the set with the intersection of itself and another.
- **Usage**: `set.intersection_update(other_set)`

`set.isdisjoint(other)`

- **Description**: Returns True if the set has no elements in common with other.
- **Usage**: `set.isdisjoint(other_set)`

`set.issubset(other)`

- **Description**: Returns True if the set is a subset of another.
- **Usage**: `set.issubset(other_set)`

`set.issuperset(other)`

- **Description**: Returns True if the set is a superset of another.
- **Usage**: `set.issuperset(other_set)`

`set.pop()`

- **Description**: Removes and returns an arbitrary set element.
- **Usage**: `set.pop()`

`set.remove(x)`

- **Description**: Removes an element from the set; it must be a member.
- **Usage**: `set.remove(10)`

`set.symmetric_difference(other)`

- **Description**: Returns the symmetric difference of the set and another as a new set.
- **Usage**: `set.symmetric_difference(other_set)`

`set.symmetric_difference_update(other)`

- **Description**: Updates the set with the symmetric difference of itself and another.
- **Usage**: `set.symmetric_difference_update(other_set)`

`set.union(*others)`

- **Description**: Returns the union of the set and other sets as a new set.
- **Usage**: `set.union(other_set)`

`set.update(*others)`

- **Description**: Updates the set, adding elements from all others.
- **Usage**: `set.update(other_set)`

---

4. Dictionary Methods

`dict.clear()`

- **Description**: Removes all items from the dictionary.
- **Usage**: `dict.clear()`

`dict.copy()`

- **Description**: Returns a shallow copy of the dictionary.
- **Usage**: `new_dict = dict.copy()`

`dict.fromkeys(iterable, value=None)`

- **Description**: Creates a new dictionary with keys from iterable and values set to value.
- **Usage**: `dict.fromkeys(['a', 'b'], 0)`

`dict.get(key[, default])`

- **Description**: Returns the value for the specified key if key is in dictionary.
- **Usage**: `dict.get('key', default)`

`dict.items()`

- **Description**: Returns a view object that displays a list of a dictionary's key-value tuple pairs.
- **Usage**: `dict.items()`

`dict.keys()`

- **Description**: Returns a view object that displays a list of all the keys in the dictionary.
- **Usage**: `dict.keys()`

`dict.pop(key[, default])`

- **Description**: Removes the key and returns its value.
- **Usage**: `dict.pop('key')`

`dict.popitem()`

- **Description**: Removes and returns an arbitrary key-value pair as a tuple.
- **Usage**: `dict.popitem()`

`dict.setdefault(key[, default])`

- **Description**: Returns the value of the specified key. If the key does not exist, inserts the key with the specified value.
- **Usage**: `dict.setdefault('key', default)`

`dict.update([other])`

- **Description**: Updates the dictionary with the key/value pairs from other.
- **Usage**: `dict.update(other_dict)`

`dict.values()`

- **Description**: Returns a view object that displays a list of all the values in the dictionary.
- **Usage**: `dict.values()`

---

5. Built-in Functions

`enumerate(iterable, start=0)`

- **Description**: Adds a counter to an iterable and returns it as an enumerate object.
- **Usage**: `enumerate(list)`

`zip(*iterables)`

- **Description**: Aggregates elements from each of the iterables.
- **Usage**: `zip(list1, list2)`

`map(function, iterable, ...)`

- **Description**: Applies a function to every item of an iterable and returns a list of the results.
- **Usage**: `map(func, list)`

`filter(function, iterable)`

- **Description**: Constructs an iterator from elements of iterable for which function returns true.
- **Usage**: `filter(func, list)`

`any(iterable)`

- **Description**: Returns `True` if any element of the iterable is true.
- **Usage**: `any(list)`

`all(iterable)`

- **Description**: Returns `True` if all elements of the iterable are true.
- **Usage**: `all(list)`

`next(iterator[, default])`

- **Description**: Retrieves the next item from the iterator.
- **Usage**: `next(iter(list))`

`iter(o[, sentinel])`

- **Description**: Returns an iterator object.
- **Usage**: `iter(list)`
