[Python Data Structure] (CheatSheet)

Lists

- **Create a list**: `my_list = [1, 2, 3, 4, 5]`
- **Append to a list**: `my_list.append(6)`
- **Extend a list with another list**: `my_list.extend([7, 8, 9])`
- **Insert into a list at a specific position**: `my_list.insert(0, 0)`
- **Remove an item from a list**: `my_list.remove(0)`
- **Pop an item from the list**: `last_item = my_list.pop()`
- **Find index of the first matching item**: `index = my_list.index(3)`
- **Count occurrences of an item**: `count = my_list.count(3)`
- **Reverse a list**: `my_list.reverse()`
- **Sort a list**: `my_list.sort()`
- **List comprehension (create a list based on existing lists)**: `squared = [x**2 for x in my_list]`
- **Slice a list**: `sub_list = my_list[2:5]`
- **Clear a list**: `my_list.clear()`
- **Copy a list**: `my_list_copy = my_list.copy()`
- **Concatenate two lists**: `concatenated_list = my_list + another_list`
- **List of lists (2D List)**: `matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]`
- **Flatten a list of lists**: `flattened = [item for sublist in matrix for item in sublist]`
- **List slicing with step**: `sliced = my_list[::2]`
- **Enumerate for index and value**: `for index, value in enumerate(my_list): print(index, value)`
- **Sort a list of dictionaries by a key**: `sorted_list = sorted(list_of_dicts, key=lambda d: d['key'])`
- **Filter a list with a condition**: `filtered = [x for x in my_list if x > 0]`
- **Map a function to a list**: `mapped = list(map(lambda x: x**2, my_list))`
- **Zip lists to create pairs of elements**: `zipped = list(zip(list1, list2))`
- **Unzip pairs into two lists**: `list1, list2 = zip(*zipped)`
- **Check if list is sorted**: `is_sorted = all(my_list[i] <= my_list[i + 1] for i in range(len(my_list) - 1))`

Dictionaries

- **Create a dictionary**: `my_dict = {'key': 'value', 'number': 42}`
- **Add or update a key-value pair**: `my_dict['new_key'] = 'new_value'`
- **Get a value for a key**: `value = my_dict.get('key')`

By: Waleed Mousa

- **Remove a key-value pair**: removed_value = my_dict.pop('key')
- **Get keys as a list**: keys_list = list(my_dict.keys())
- **Get values as a list**: values_list = list(my_dict.values())
- **Check if a key exists**: exists = 'number' in my_dict
- **Dictionary comprehension**: squared_dict = {k: v**2 for k, v in my_dict.items() if isinstance(v, int)}
- **Merge two dictionaries**: merged_dict = {**dict1, **dict2}
- **Iterate through key-value pairs**: for key, value in my_dict.items(): print(key, value)
- **Clear a dictionary**: my_dict.clear()
- **Copy a dictionary**: my_dict_copy = my_dict.copy()
- **Get length of a dictionary**: length = len(my_dict)
- **Remove and return an arbitrary (key, value) pair**: key, value = my_dict.popitem()
- **Dictionary from two lists**: new_dict = dict(zip(list_of_keys, list_of_values))
- **Nested dictionaries**: nested_dict = {'dictA': {'key1': 'value1'}, 'dictB': {'key2': 'value2'}}
- **Iterate through keys and values**: for key, value in my_dict.items(): print(key, value)
- **Dictionary of lists**: dict_of_lists = {'list1': [1, 2, 3], 'list2': [4, 5, 6]}
- **Access nested dictionary items**: nested_item = nested_dict['dictA']['key1']
- **Sort a dictionary by its values**: sorted_dict = dict(sorted(my_dict.items(), key=lambda item: item[1]))
- **Filter items in a dictionary**: filtered_dict = {k: v for k, v in my_dict.items() if 'condition' in v}
- **Dictionary from two lists (using zip)**: dict_from_lists = dict(zip(list_keys, list_values))
- **Use defaultdict for missing keys**: from collections import defaultdict; my_defaultdict = defaultdict(int)
- **Update a dictionary with another dictionary**: my_dict.update(other_dict)
- **Convert dictionary keys & values to lists**: keys, values = list(my_dict.keys()), list(my_dict.values())

Sets

- **Create a set**: my_set = {1, 2, 3, 4, 5}
- **Add an element to a set**: my_set.add(6)
- **Remove an element from a set**: my_set.remove(6)

- **Discard an element (no error if not found)**: `my_set.discard(5)`
- **Pop an element from the set**: `element = my_set.pop()`
- **Clear a set**: `my_set.clear()`
- **Union of two sets**: `union_set = my_set | another_set`
- **Intersection of two sets**: `intersection_set = my_set & another_set`
- **Difference between two sets**: `difference_set = my_set - another_set`
- **Symmetric difference between two sets**: `symmetric_difference_set = my_set ^ another_set`
- **Check if a set is a subset of another set**: `is_subset = my_set.issubset(another_set)`
- **Check if a set is a superset of another set**: `is_superset = my_set.issuperset(another_set)`
- **Set comprehension**: `squared_set = {x**2 for x in my_set}`
- **Copy a set**: `my_set_copy = my_set.copy()`
- **Check for membership in a set**: `exists = 4 in my_set`
- **Convert list to set to remove duplicates**: `unique_items = set(my_list)`
- **Set operations with assignment**: `my_set |= another_set # Union`
- **Find set difference in place**: `my_set -= another_set`
- **Intersection update**: `my_set &= another_set`
- **Symmetric difference update**: `my_set ^= another_set`
- **Subset and superset checks**: `is_subset = my_set <= another_set; is_superset = my_set >= another_set`
- **FrozenSet (Immutable version of Set)**: `frozen_set = frozenset([1, 2, 3])`
- **Convert set to list**: `my_list = list(my_set)`
- **Perform set operations without modifying the original sets**: `union_set = my_set.union(another_set)`
- **Create a set with set comprehension**: `set_comp = {x for x in range(10) if x % 2 == 0}`

Tuples

- **Create a tuple**: `my_tuple = (1, 2, 3, 4, 5)`
- **Access tuple item**: `item = my_tuple[1]`
- **Slicing a tuple**: `sub_tuple = my_tuple[1:3]`
- **Count occurrences of an item**: `count = my_tuple.count(3)`
- **Find index of the first matching item**: `index = my_tuple.index(3)`
- **Unpack a tuple**: `a, b, c, d, e = my_tuple`
- **Tuple with a single item**: `single_item_tuple = (1,)`
- **Concatenate tuples**: `concatenated_tuple = my_tuple + another_tuple`
- **Check for membership in a tuple**: `exists = 3 in my_tuple`
- **Iterate through a tuple**: `for item in my_tuple: print(item)`

- **Convert list to tuple**: `my_tuple = tuple(my_list)`
- **Nested tuple**: `nested_tuple = (1, (2, 3), 4)`
- **Tuple length**: `length = len(my_tuple)`
- **Convert tuple to list**: `my_list = list(my_tuple)`
- **Immutable nature (trying to change a value throws an error)**: `# my_tuple[2] = 10 # TypeError`

Strings

- **Concatenate strings**: `concatenated = 'Hello' + ' ' + 'World'`
- **Access substring**: `sub = my_string[1:5]`
- **Convert to uppercase**: `upper_case = my_string.upper()`
- **Convert to lowercase**: `lower_case = my_string.lower()`
- **Find position of a substring**: `pos = my_string.find('sub')`
- **Replace a substring**: `replaced = my_string.replace('old', 'new')`
- **Split a string into a list**: `split_list = my_string.split(',')`
- **Join a list into a string**: `joined_string = ','.join(my_list)`
- **Strip leading and trailing whitespaces**: `stripped = my_string.strip()`
- **Check if string starts with a substring**: `starts_with = my_string.startswith('He')`
- **Check if string ends with a substring**: `ends_with = my_string.endswith('ld')`
- **String formatting with f-strings**: `formatted = f"Value: {value}"`
- **Check if all characters are alphanumeric**: `is_alnum = my_string.isalnum()`
- **Reverse a string**: `reversed_string = my_string[::-1]`
- **Length of a string**: `length = len(my_string)`
- **Count occurrences of a character or substring**: `count = my_string.count('a')`
- **Check if string contains a substring**: `contains = 'substr' in my_string`
- **Strip whitespace (or other characters) from ends**: `stripped = my_string.strip()`
- **Capitalize the first letter**: `capitalized = my_string.capitalize()`
- **Title case (first letter of each word capitalized)**: `title_case = my_string.title()`
- **Swap case**: `swapped = my_string.swapcase()`
- **Slice a string**: `sliced = my_string[2:5]`
- **Iterate over a string**: `for char in my_string: print(char)`
- **String formatting with placeholders**: `formatted = "Hello, {}. You are {}.".format('John', 30)`
- **String alignment and padding**: `padded = my_string.center(20, '*')`

- **Zfill - Pad a numeric string on the left with zeros**: `zfilled = my_string.zfill(5)`
- **String to list of characters**: `char_list = list(my_string)`
- **Regular expression operations**: `import re; matched = re.findall(r'\d+', my_string)`
- **Encode and decode strings (e.g., UTF-8)**: `encoded = my_string.encode('utf-8'); decoded = encoded.decode('utf-8')`
- **Remove digits from a string**: `no_digits = ''.join([i for i in my_string if not i.isdigit()])`
- **Find and replace using regular expressions**: `replaced = re.sub(r'[a-z]+', 'replacement', my_string)`
- **String slicing with steps**: `step_slice = my_string[::2]`
- **Convert between strings and bytes**: `byte_str = my_string.encode(); str_from_bytes = byte_str.decode()`
- **Multiline strings and stripping leading whitespace**: `multiline = '''\n line1\n line2\n'''.strip()`