# Common Operations in Python Data Structures

## Arrays (Lists)

In Python, lists are commonly used to represent arrays. While Python has an `array` module, lists are more versatile.

|  |  |  |
| --- | --- | --- |
| Operation | Example | Description/Time Complexity |
| Accessing an element | `my\_list[index]` | (O(1)) - Accesses the element at the specified index. |
| Appending an element | `my\_list.append(item)` | (O(1) amortized) - Adds an element to the end of the list. |
| Inserting an element | `my\_list.insert(index, item)` | (O(n)) - Inserts an element at a specific index. |
| Removing an element (by value) | `my\_list.remove(item)` | (O(n)) - Removes the first occurrence of an element. |
| Removing an element (by index) | `my\_list.pop(index)` | (O(n) in general, O(1) for the last element) - Removes and returns the element at the specified index. |
| Finding the length | `len(my\_list)` | (O(1)) - Returns the number of elements in the list. |
| Slicing | `my\_list[start:end:step]` | (O(k), where k is the size of the slice) - Creates a new list containing a portion of the original list. |
| Iterating | `for item in my\_list:` | (O(n)) - Iterates through the elements of the list. |
| Checking for membership | `item in my\_list` | (O(n)) - Checks if an element exists in the list. |
| Sorting | `my\_list.sort()` | (in-place, O(n log n)) or `sorted(my\_list)` (returns a new sorted list, O(n log n)) - Sorts the list. |
| Reversing | `my\_list.reverse()` | (in-place, O(n)) or `reversed(my\_list)` (returns an iterator, O(n) to iterate) - Reverses the order of elements. |
| Concatenation | `list1 + list2` | (O(n+m) where n and m are the lengths of lists ) - create a new list containing all elements. |

## Strings

Strings are sequences of characters and are immutable.

|  |  |  |
| --- | --- | --- |
| Operation | Example | Description/Time Complexity |
| Accessing a character | `my\_string[index]` | (O(1)) - Accesses the character at the specified index. |
| Slicing | `my\_string[start:end:step]` | (O(k), where k is the size of the slice) - Creates a new string containing a portion of the original string. |
| Concatenation | `string1 + string2` | (O(n+m), where n and m are string lengths) - Creates a new string by joining two strings. |
| Finding the length | `len(my\_string)` | (O(1)) - Returns the number of characters in the string. |
| Iterating | `for char in my\_string:` | (O(n)) - Iterates through the characters of the string. |
| Checking for substring | `substring in my\_string` | (O(nm) in worst case, but often faster with optimized algorithms like KMP) - Checks if a substring exists in the string. |
| Converting case | `my\_string.lower()` | (O(n)), `my\_string.upper()` (O(n)) - Converts the string to lowercase or uppercase. |
| Replacing substrings | `my\_string.replace(old, new)` | (O(n)) - Creates a new string with replaced substrings. |
| Splitting | `my\_string.split(separator)` | (O(n)) - Creates a list of strings by splitting the string at the separator. |
| Joining | `separator.join(list\_of\_strings)` | (O(n)) - Creates a string by joining elements of a list with a separator. |
| Finding index of substring | `string.find(substring)` | (O(nm)) - Returns the index of the first occurrence of the substring or -1 if it is not found. |
| Stripping whitespace | `my\_string.strip()` | (O(n))- returns a new string with leading and trailing whitespaces removed. |

## Dictionaries

Dictionaries store key-value pairs. Keys must be immutable (e.g., strings, numbers, tuples).

|  |  |  |
| --- | --- | --- |
| Operation | Example | Description/Time Complexity |
| Accessing a value | `my\_dict[key]` | (O(1) average, O(n) worst case) - Accesses the value associated with the key. |
| Adding or updating a key-value pair | `my\_dict[key] = value` | (O(1) average, O(n) worst case) - Adds a new key-value pair or updates the value for an existing key. |
| Removing a key-value pair | `del my\_dict[key]` | (O(1) average, O(n) worst case) - Removes the key-value pair. |
| Checking for key existence | `key in my\_dict` | (O(1) average, O(n) worst case) - Checks if a key exists in the dictionary. |
| Getting all keys | `my\_dict.keys()` | (O(1) - returns a view object) - Returns a view object of all keys. |
| Getting all values | `my\_dict.values()` | (O(1) - returns a view object) - Returns a view object of all values. |
| Getting all key-value pairs | `my\_dict.items()` | (O(1) - returns a view object) - Returns a view object of all key-value pairs as tuples. |
| Iterating | `for key, value in my\_dict.items():` | (O(n)) - Iterates through the key-value pairs. |
| Finding the length (number of key-value pairs) | `len(my\_dict)` | (O(1)) - Returns the number of key-value pairs. |
| Removing and return a value | `my\_dict.pop(key)` | (O(1) average, O(n) worst-case) - Returns value associated with the key. Key value pair also removed. Raises KeyError if Key not found. |
| Get a value, or default | `my\_dict.get(key, default\_value)` | (O(1) average, O(n) worst case) - returns the value associated with key, or default\_value if the key isn't present. |