Cheatsheets / **Learn Python 3**

Lists

Lists

In Python, lists are ordered collections of items that allow for easy use of a set of data.
List values are placed in between square brackets `[ ]`, separated by commas. It is good practice to put a space between the comma and the next value. The values in a list do not need to be unique (the same value can be repeated).
Empty lists do not contain any values within the square brackets.

```
primes = [2, 3, 5, 7, 11]
print(primes)

empty_list = []
```

Adding Lists Together

In Python, lists can be added to each other using the plus symbol `+`. As shown in the code block, this will result in a new list containing the same items in the same order with the first list's items coming first.
Note: This will not work for adding one item at a time (use `.append()` method). In order to add one item, create a new list with a single value and then use the plus symbol to add the list.

```
items = ['cake', 'cookie', 'bread']
total_items = items + ['biscuit', 'tart']
print(total_items)
# Result: ['cake', 'cookie', 'bread',
'biscuit', 'tart']
```

Python Lists: Data Types

In Python, lists are a versatile data type that can contain multiple different data types within the same square brackets. The possible data types within a list include numbers, strings, other objects, and even other lists.

```
numbers = [1, 2, 3, 4, 10]
names = ['Jenny', 'Sam', 'Alexis']
mixed = ['Jenny', 1, 2]
list_of_lists = [['a', 1], ['b', 2]]
```

List Method `.append()`

In Python, you can add values to the end of a list using the `.append()` method. This will place the object passed in as a new element at the very end of the list. Printing the list afterwards will visually show the appended value. This `.append()` method is *not* to be confused with returning an entirely new list with the passed object.

```
orders = ['daisies', 'periwinkle']
orders.append('tulips')
print(orders)
# Result: ['daisies', 'periwinkle',
'tulips']
```

Zero-Indexing

In Python, list index begins at zero and ends at the length of the list minus one. For example, in this list, `'Andy'` is found at index `2`.

```
names = ['Roger', 'Rafael', 'Andy',
'Novak']
```

List Indices

Python list elements are ordered by *index*, a number referring to their placement in the list. List indices start at 0 and increment by one.
To access a list element by index, square bracket notation is used: `list[index]`.

```
berries = ["blueberry", "cranberry",
"raspberry"]

berries[0]   # "blueberry"
berries[2]   # "raspberry"
```

Negative List Indices

Negative indices for lists in Python can be used to reference elements in relation to the end of a list. This can be used to access single list elements or as part of defining a list range. For instance:

- To select the last element, `my_list[-1]`.
- To select the last three elements, `my_list[-3:]`.
- To select everything except the last two elements, `my_list[:-2]`.

```
soups = ['minestrone', 'lentil', 'pho',
'laksa']
soups[-1]   # 'laksa'
soups[-3:]  # 'lentil', 'pho', 'laksa'
soups[:-2]  # 'minestrone', 'lentil'
```

Modifying 2D Lists

In order to modify elements in a 2D list, an index for the sublist and the index for the element of the sublist need to be provided. The format for this is `list[sublist_index][element_in_sublist_index] = new_value` .

```
# A 2D list of names and hobbies
class_name_hobbies = [["Jenny",
"Breakdancing"], ["Alexus",
"Photography"], ["Grace", "Soccer"]]

# The sublist of Jenny is at index 0. The
hobby is at index 1 of the sublist.
class_name_hobbies[0][1] = "Meditation"
print(class_name_hobbies)

# Output
# [["Jenny", "Meditation"], ["Alexus",
"Photography"], ["Grace", "Soccer"]]
```

Accessing 2D Lists

In order to access elements in a 2D list, an index for the sublist and the index for the element of the sublist both need to be provided. The format for this is `list[sublist_index][element_in_sublist_index]` .

```
# 2D list of people's heights
heights = [["Noelle", 61], ["Ali", 70],
["Sam", 67]]
# Access the sublist at index 0, and then
access the 1st index of that sublist.
noelles_height = heights[0][1]
print(noelles_height)

# Output
# 61
```

List Method `.remove()`

The `.remove()` method in Python is used to remove an element from a list by passing in the value of the element to be removed as an argument. In the case where two or more elements in the list have the same value, the first occurrence of the element is removed.

```
# Create a list
shopping_line = ["Cole", "Kip", "Chris",
"Sylvana", "Chris"]

# Removes the first occurance of "Chris"
shopping_line.remove("Chris")
print(shopping_line)

# Output
# ["Cole", "Kip", "Sylvana", "Chris"]
```

Tuples

Tuples are one of the built-in data structures in Python. Tuples are immutable, meaning we can't modify a tuple's elements after creating one, and they do not require an extra memory block like lists. Because of this, tuples are great to work with if you are working with data that won't need to be changed in your code. Some of the built-in methods and functions to be used with tuples are: `len()`, `max()`, `min()`, `.index()` and `.count()`.

```
my_tuple = ('abc', 123, 'def', 456, 789,
'ghi')

len(my_tuple) # returns length of tuple
max(my_tuple) # returns maximum value of
tuple
min(my_tuple) # returns minimum value of
tuple
my_tuple.index(123) # returns the
position of the value 123
my_tuple.count('abc') # returns the
number of occurrences of the value 'abc'
```

List Method `.count()`

The `.count()` Python list method searches a list for whatever search term it receives as an argument, then returns the number of matching entries found.

```
backpack = ['pencil', 'pen', 'notebook',
'textbook', 'pen', 'highlighter', 'pen']
numPen = backpack.count('pen')

print(numPen)
# Output: 3
```

Determining List Length with `len()`

The Python `len()` function can be used to determine the number of items found in the list it accepts as an argument.

```
knapsack = [2, 4, 3, 7, 10]
size = len(knapsack)
print(size)
# Output: 5
```

List Method `.sort()`

The `.sort()` Python list method will sort the contents of whatever list it is called on. Numerical lists will be sorted in ascending order, and lists of Strings will be sorted into alphabetical order. It modifies the original list, and has no return value.

```
exampleList = [4, 2, 1, 3]
exampleList.sort()
print(exampleList)
# Output: [1, 2, 3, 4]
```

List Slicing

A *slice*, or sub-list of Python list elements can be selected from a list using a colon-separated starting and ending point.
The syntax pattern is `myList[START_NUMBER:END_NUMBER]`. The slice will include the `START_NUMBER` index, and everything until but excluding the `END_NUMBER` item.
When slicing a list, a new list is returned, so if the slice is saved and then altered, the original list remains the same.

```
tools = ['pen', 'hammer', 'lever']
tools_slice = tools[1:3] # ['hammer',
'lever']
tools_slice[0] = 'nail'

# Original list is unaltered:
print(tools) # ['pen', 'hammer', 'lever']
```

`sorted()` Function

The Python `sorted()` function accepts a list as an argument, and will return a new, sorted list containing the same elements as the original. Numerical lists will be sorted in ascending order, and lists of Strings will be sorted into alphabetical order. It does not modify the original, unsorted list.

```
unsortedList = [4, 2, 1, 3]
sortedList = sorted(unsortedList)
print(sortedList)
# Output: [1, 2, 3, 4]
```

List Method `.insert()`

The Python list method `.insert()` allows us to add an element to a specific index in a list.
It takes in two inputs:

- The index that you want to insert into.
- The element that you want to insert at the specified index.

```
# Here is a list representing a line of people at a store
store_line = ["Karla", "Maxium", "Martim", "Isabella"]

# Here is how to insert "Vikor" after "Maxium" and before "Martim"
store_line.insert(2, "Vikor")

print(store_line)
# Output: ['Karla', 'Maxium', 'Vikor', 'Martim', 'Isabella']
```

List Method `.pop()`

The `.pop()` method allows us to remove an element from a list while also returning it. It accepts one optional input which is the index of the element to remove. If no index is provided, then the last element in the list will be removed and returned.

```
cs_topics = ["Python", "Data Structures", "Balloon Making", "Algorithms", "Clowns 101"]

# Pop the last element
removed_element = cs_topics.pop()

print(cs_topics)
print(removed_element)

# Output:
# ['Python', 'Data Structures', 'Balloon Making', 'Algorithms']
# 'Clowns 101'

# Pop the element "Baloon Making"
cs_topics.pop(2)
print(cs_topics)

# Output:
# ['Python', 'Data Structures', 'Algorithms']
```

Print Share