Python List Cheat Sheet

A Python list is a collection whose items can be of any type. A comma separates the items. Square braces (‘[‘ and ‘]’) delimit a list. A list is **mutable** (i.e., it can be changed in place).

`list()` or `[]`: creates new empty list and returns it.
`list(iterable)`: creates a list containing the items in `iterable` and returns it.

```
alist[pos] = val: Assigns val as the item at index pos of alist in place.
alist[pos]: If not the left side of an assignment, returns the item at index pos of alist.
alist[start:end:step]: If not the left side of an assignment, returns the list created from
alist starting with the item at index start, counting by step until (but not including) the
item at index end. Does not modify alist.
alist[start:end] = iterable: Assigns iterable to the slice of alist on the right side
of the assignment.
alist.append(val): Appends (adds) val as the last item of alist.
alist.extend(iterable): Extends alist by appending to it each item in iterable.
alist.index(val): Returns the index of the first occurrence of val in alist.
alist.insert(index, val): Inserts val into alist just before the item at index index.
alist.pop(): Removes the last item of alist and also returns the item removed.
alist.reverse(): Reverses the order of the items of alist.
alist.remove(val): Removes the first item in alist whose value equals val.
alist.sort(), alist.sort(reverse=True): Sorts the items in alist in ascending
order, if reverse is False (the default), or in descending order, if reverse is True.
max(alist): Returns the maximum (largest) item in alist.
min(alist): Return the minimum (smallest) item in alist.
alist + blist: Returns a new list created by concatenating alist and blist.
num * alist: Returns a new list created by concatenating num copies of alist into one list.
val in alist: Returns True, if some item in alist equals val; and False, otherwise.
alist < blist, alist <= blist, alist > blist, alist >= blist, 
alist == blist, alist != blist: compares items of the two lists in order; the
relationship between the first two items that differ determines the result.
len(alist): Returns the length (number of items) in alist.
del alist[pos]: Deletes the item at index pos.
String methods for working with lists: `join(...)`, `split(...)`

`astr.join(iterable_of_string)`: returns a new string obtained by joining (concatenating) the items in `iterable_of_string` with `astr` as a separator.

`astr.split()`, `astr.split( astr_sep )`: returns the list of words (strings) of `astr` delimited by whitespace, by default, or by `astr_sep`, if given.

### Python Tuple Cheat Sheet

Like a list, a Python tuple is a collection whose items can be of any type. Also like a list, a comma separates the items. But unlike a list, a tuple is immutable. Also, surrounding parentheses are not needed. Some useful tuple operations follow.

- `tuple()`: creates a new empty tuple and returns it.
- `tuple(iterable)`: creates tuple from the items in `iterable` and returns it.

- `atuple[pos]`: Returns the item at index `pos` of `atuple`. (Not allowed as the left side of an assignment)

- `atuple[start:end:step]`: Returns a new tuple created from `atuple` starting with the item at index `start`, counting by `step` until (but not including) the item at index `end`.

- `max(atuple)`: Returns the maximum (largest) item in `atuple`.
- `min(atuple)`: Returns the minimum (smallest) item in `atuple`.

- `atuple + btuple`: Returns a new tuple created by concatenating `atuple` and `btuple`.

- `num * atuple`: Returns a new tuple created by concatenating `num` copies of `alist` into one list.

- `val in atuple`: Returns `True`, if some item in `atuple` equals `val`; and `False`, otherwise.

- `atuple < btuple`, `atuple <= btuple`, `atuple > btuple`, `atuple >= btuple`, `atuple == btuple`, `atuple != btuple`: compares items of the two tuples in order; the relationship between the first two items that differ determines the result.

- `len(atuple)`: Returns the length of (number of items in) `atuple`.

### Iteration over a collection (list, tuple, string, range)

- `for x in iterable:`
  - `suite`

Repeatedly execute `suite` for each item in `iterable`; before each iteration, assign `x` the next value in `iterable`. Here, `iterable` stands for any collection (e.g., tuple, list, string, range).