Python References

Operator precedence (summarized):

Operator	Description
`if – else`	Conditional expression
`or`	Boolean OR
`and`	Boolean AND
`not x`	Boolean NOT
`in, not in, is, is not, <, <=, >, >=, !=, ==`	Comparisons, incl. membership tests and identity tests
`+, -`	Addition and subtraction
`*, @, /, //, %`	Multiplication, matrix multiplication, division, remainder
`+x, -x`	Positive, negative
`**`	Exponentiation
`x[index], x[index:index], x(arguments...), x.attribute`	Subscription, slicing, call, attribute reference
`(expressions...), [expressions...], {key: value...}, {expressions...}`	Binding or tuple display, list display, dictionary display, set display

Functions:

- **`len(s)`**
  - Return the length (the number of items) of an object. The argument may be a sequence (such as a string, bytes, tuple, list, or range) or a collection (such as a dictionary, set, or frozen set).
- **`print(*objects, sep=' ', end='\n', file=sys.stdout, flush=False)`**
  - Print objects to the text stream file, separated by *sep* and followed by *end*. *sep*, *end*, *file* and *flush,* if present, must be given as keyword arguments.
  - All non-keyword arguments are converted to strings like `str()` does and written to the stream, separated by *sep* and followed by *end*. Both *sep* and *end* must be strings; they can also be `None`, which means to use the default values. If no objects are given, `print()` will just write *end*.
  - The *file* argument must be an object with a `write(string)` method; if it is not present or `None`, `sys.stdout` will be used. Since printed arguments are converted to text strings, `print()` cannot be used with binary mode file objects. For these, use *`file.write(...)`* instead.
  - Whether output is buffered is usually determined by *file*, but if the *flush* keyword argument is `True`, the stream is forcibly flushed.
  - Changed in version 3.3: Added the *flush* keyword argument.

- **`class range(stop)`¶**
- **`class range(start, stop[, step])`**
  - The arguments to the range constructor must be integers (either built-in int or any object that implements the `__index__` special method). If the step argument is omitted, it defaults to 1. If the start argument is omitted, it defaults to 0. If step is zero, `ValueError` is raised.
  - For a positive step, the contents of a range r are determined by the formula `r[i] = start + step*i` where `i >= 0` and `r[i] < stop`.
  - For a negative step, the contents of the range are still determined by the formula `r[i] = start + step*i`, but the constraints are `i >= 0` and `r[i] > stop`.
  - A range object will be empty if `r[0]` does not meet the value constraint. Ranges do support negative indices, but these are interpreted as indexing from the end of the sequence determined by the positive indices.
  - Ranges containing absolute values larger than `sys.maxsize` are permitted but some features (such as `len()`) may raise `OverflowError`.

Mutable sequence types

Operation	Result	
`s[i] = x`	item i of s is replaced by x	
`s[i:j] = t`	slice of s from i to j is replaced by the contents of the iterable t	
`del s[i:j]`	same as s[i:j] = []	
`s[i:j:k] = t`	the elements of s[i:j:k] are replaced by those of t	(1)
`del s[i:j:k]`	removes the elements of s[i:j:k]from the list	
`s.append(x)`	appends x to the end of the sequence (same as s[len(s):len(s)] = [x])	
`s.clear()`	removes all items from s (same as del s[:])	(5)
`s.copy()`	creates a shallow copy of s (same as s[:])	(5)
`s.extend(t)` or `s += t`	extends s with the contents of t (for the most part the same ass[len(s):len(s)] = t)	
`s *= n`	updates s with its contents repeated n times	(6)
`s.insert(i, x)`	inserts x into s at the index given by i(same as s[i:i] = [x])	
`s.pop([i])`	retrieves the item at i and also removes it from s	(2)
`s.remove(x)`	remove the first item from s where s[i] == x	(3)
`s.reverse()`	reverses the items of s in place	(4)

Notes:

1. `t` must have the same length as the slice it is replacing.
2. The optional argument i defaults to -1, so that by default the last item is removed and returned.
3. remove raises ValueError when x is not found in s.
4. The reverse() method modifies the sequence in place for economy of space when reversing a large sequence. To remind users that it operates by side effect, it does not return the reversed sequence.
5. clear() and copy() are included for consistency with the interfaces of mutable containers that don't support slicing operations (such as dict and set). New in version 3.3: `clear()` and `copy()` methods.
6. The value n is an integer, or an object implementing `__index__()`. Zero and negative values of `n` clear the sequence. Items in the sequence are not copied; they are referenced multiple times, as explained for s * n under Common Sequence Operations.