Information, Computation, Communication

Learning Python

Lists

Agenda

- Definition
- Accessing list elements by index or by slicing
- List operations:
  - Concatenation and repetition, membership
- Examples: Traversing lists
  - for <> in <>
  - for <> in range <>
- List comprehension
- **Homework**
  - List methods: growing, searching, sorting and reversing
  - Modifying lists using slicing

Next topic: **Nested** loops and lists

© kras99 / Adobe Stock

What Are Lists?

- Lists are **ordered** collections of arbitrary objects: numbers, strings, and even other lists!
  - Lists are **mutable** (i.e., their elements can be changed)

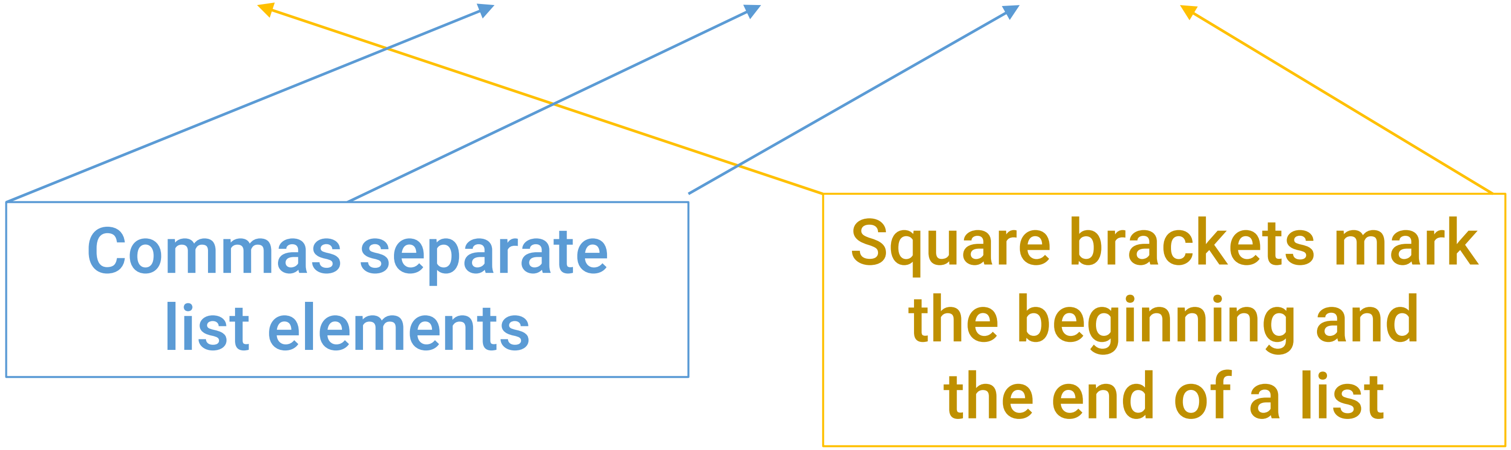
- List syntax

```
# Create a list called my_example_list and
# assign arbitrary elements to it. For example:
my_example_list = [True, "ICC", -99.5, 0]
print(my_example_list)
# [True, 'ICC', -99.5, 0]
```

Creating Lists

- Previous example, creating a list of four elements:

```
my_example_list = [True, "ICC", -99.5, 0]
```



Commas separate list elements

Square brackets mark the beginning and the end of a list

- Creating an **empty** list:

```
my_empty_list = []
```

Accessing List Elements

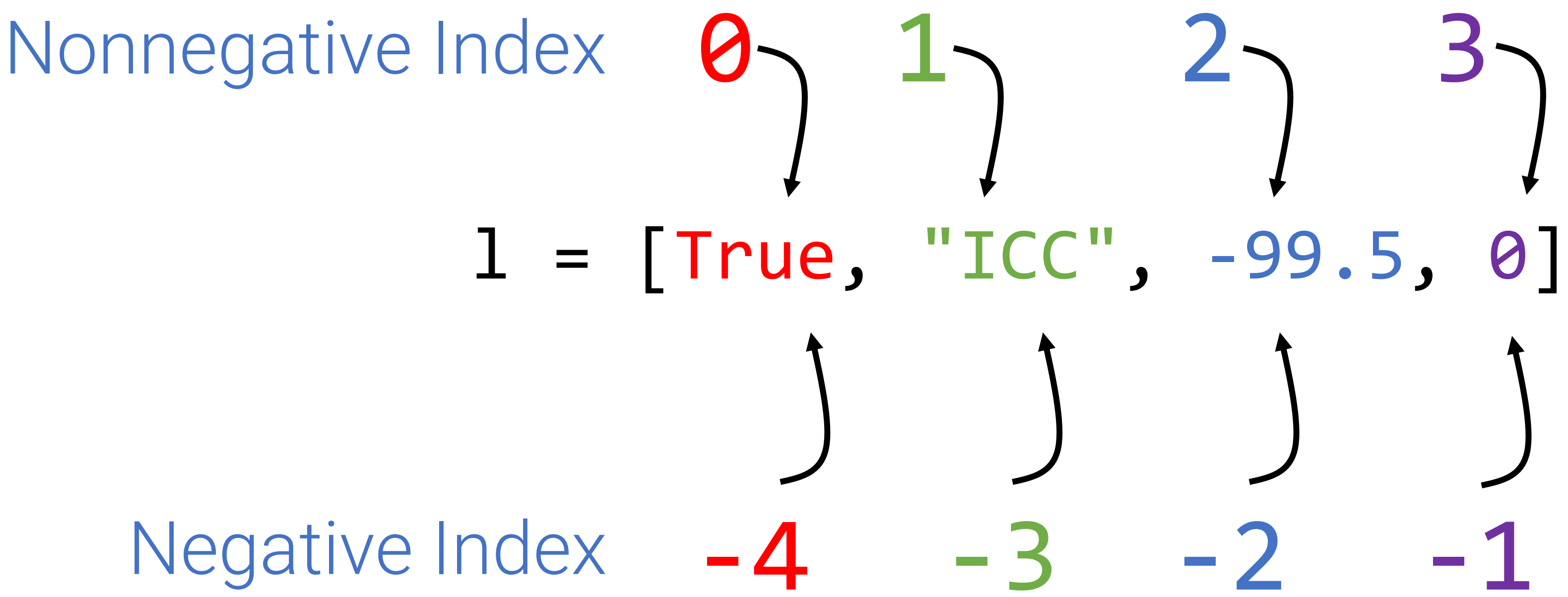
© kras99 / Adobe Stock

Accessing List Elements

- There are two ways to **access** list elements for reading them or modifying them
- (1) Accessing by the **index** of the element
  - Index is the element **position (offset from the beginning)** in the list
  - Returns the element at the given index
- (2) Accessing by **slicing**
  - Similar to accessing by index, except that we specify a range of indices between **start** and **stop-1**
  - Can return more than a single element, i.e., a "slice" of a list

Accessing List Elements by Index

- List elements are ordered by their position (index) in the list
  - Indices may be nonnegative (most common) but also negative



Nonnegative Index 0 1 2 3

```
l = [True, "ICC", -99.5, 0]
```

Negative Index -4 -3 -2 -1

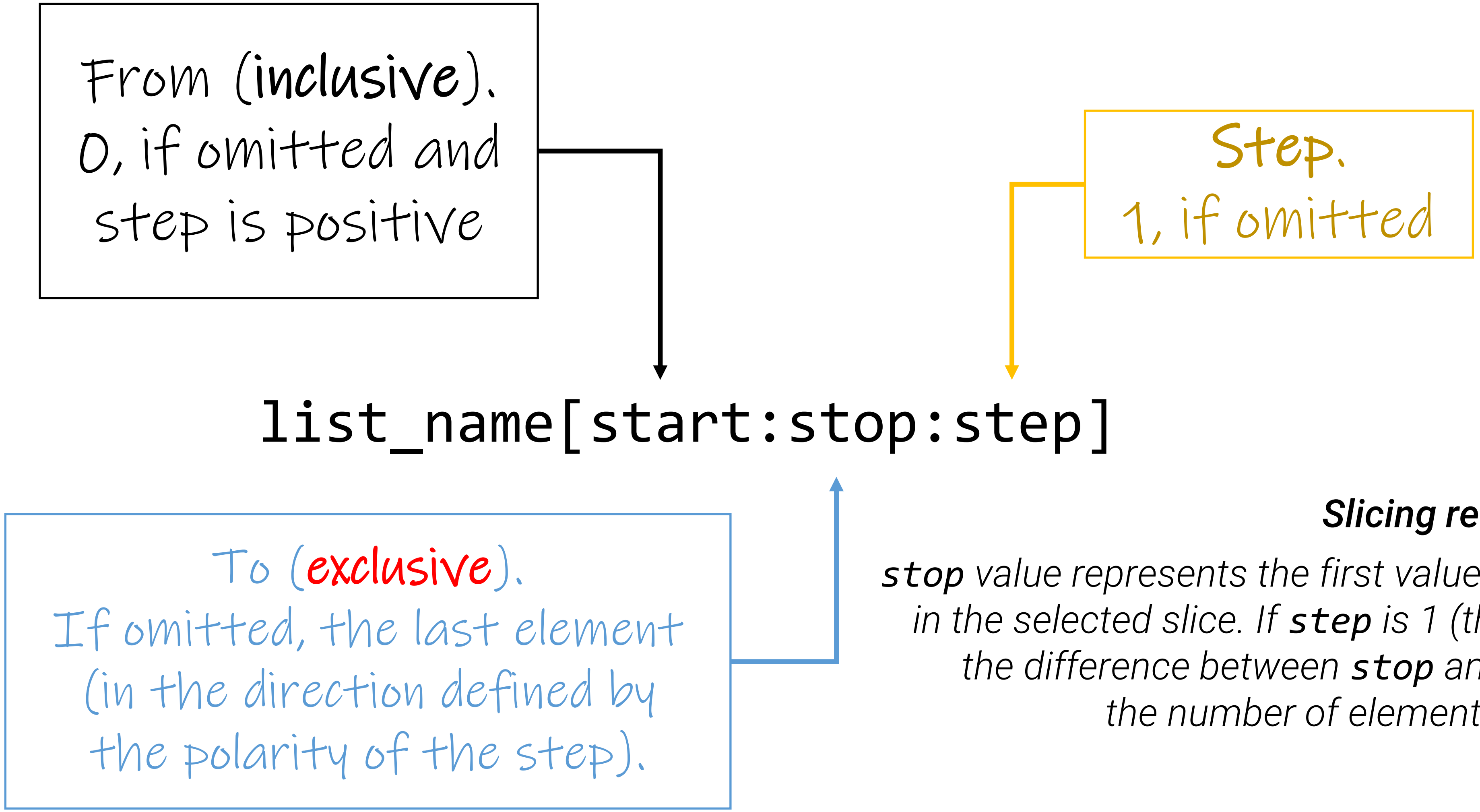
Examples: Accessing List Elements by Index

EXAMPLES

- The index of an element determines its position in the list

```
l = [True, "ICC", -99.5, 0]

print(l[1])          # print an element
# ICC

l[2] += 1            # modify an element
print(l[2])
# -98.5

print(l[-4] - l[-1]) # compute
# 1
```

Accessing List Elements by Slicing



From (**inclusive**). 0, if omitted and step is positive

Step. 1, if omitted

```
list_name[start:stop:step]
```

To (**exclusive**). If omitted, the last element (in the direction defined by the polarity of the step).

Slicing returns a list

*`stop` value represents the first value that is **not** in the selected slice. If `step` is 1 (the default), the difference between `stop` and `start` is the number of elements selected.*

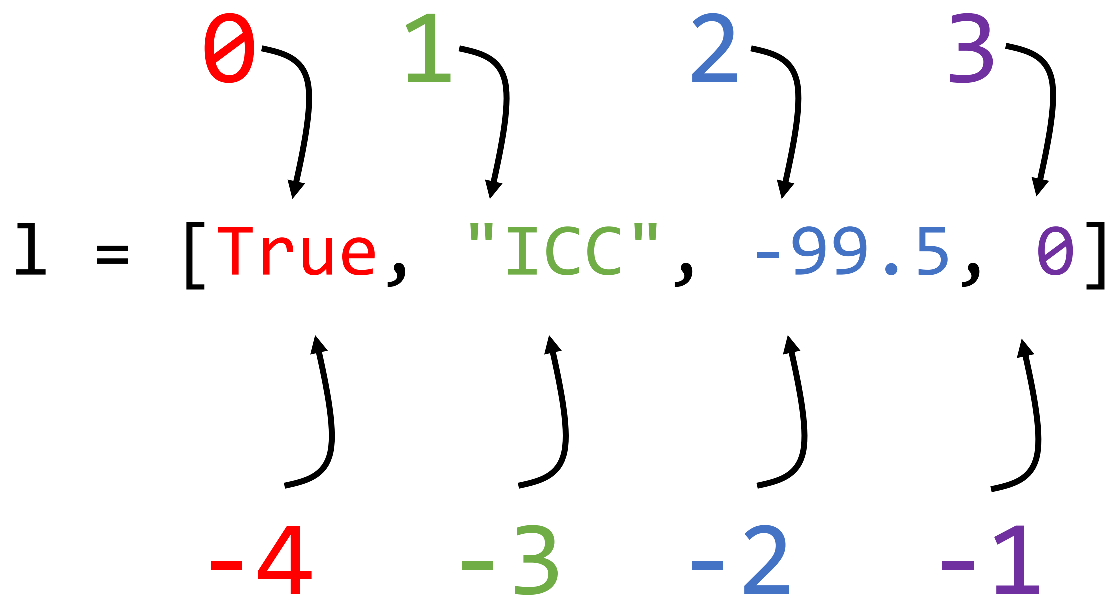
Syntax for List Slicing

```
# positive step; start < stop
x[low:high:step]
# [x[low], x[low+step], x[low+2step], …, x[high-1]]
# if (high-low)%step != 0, the endpoint is lower than high-1

# negative step; start > stop, reverse order of traversal
x[high:low:step]
# [x[high], x[high-step], x[high-2step]…, x[low+1]]
# if (high-low)%step != 0, the endpoint is higher than low+1
```

EXAMPLES

Examples: Accessing List Elements by Slicing

```
l[:]
# [True, 'ICC', -99.5, 0]
l[2:4]
# [-99.5, 0]
l[1::2]
# ['ICC', 0]
l[-1]
# 0
l[::-1]
# [0, -99.5, 'ICC', True]
l[1::-1]
# ['ICC', True]
```



```
l = [True, "ICC", -99.5, 0]
```

List Operations

- Concatenation
- Repetition
- Membership Check

© kras99 / Adobe Stock

Concatenation and Repetition

```
a = [0, 1.1, 2.2]
b = ['O', 'K', '!']
# Concatenation using the addition operator
a + b
# [0, 1.1, 2.2, 'O', 'K', '!’]

# Repetition, using the multiplication operator
b * 2
# ['O', 'K', '!', 'O', 'K', '!']
```

List Membership Check

```
a = [0, 1.1, 2.2]
b = ['O', 'K', '!']
# Membership check
0 in a
# True
2 in b
# False
'!' in b
# True
```

Examples

© kras99 / Adobe Stock

Example 1: Traversing a List (for < > in < >)

Write a piece of code that traverses a list, counts all strings in it, and prints out the count

Example: my_list = [5, 'song', 'cello', 60.4, 'theater', 'scene', -6.20, True]
Expected result: 4

EXAMPLES

Example 1: Traversing a List (for < > in < >)

Example: my_list = [5, 'song', 'cello', 60.4, 'theater', 'scene', -6.20, True]
Expected result: 4

```
my_list = [5, 'song', 'cello', 60.4,
           'theater',  'scene', -6.20, True]
# Start counting
n_strings = 0
for i in my_list:           # Traverse the list
    if type(i) is str:      # True if element i is a string
        n_strings +=1       # Update the count
print(n_strings)
```

Example 2: Traversing a List (for <> in range<>)

Write a piece of code that traverses one list and returns another list, which contains every element at an even index of the original list.

Example: in_list = [43, -32, -94, -10, -18, 33, -59]
Expected result: out_list = [43, -94, -18, -59]

Hint 1: Python has a built-in function* `len()` *which returns the number of items in an object (e.g., characters in a string, elements in a list, etc.)

Hint 2: There is a method called* `append()`*, to insert an element at the end of a list (e.g.,* `out_list.append(new_element)`*)

Example 2: Traversing a List (for <> in range<>)

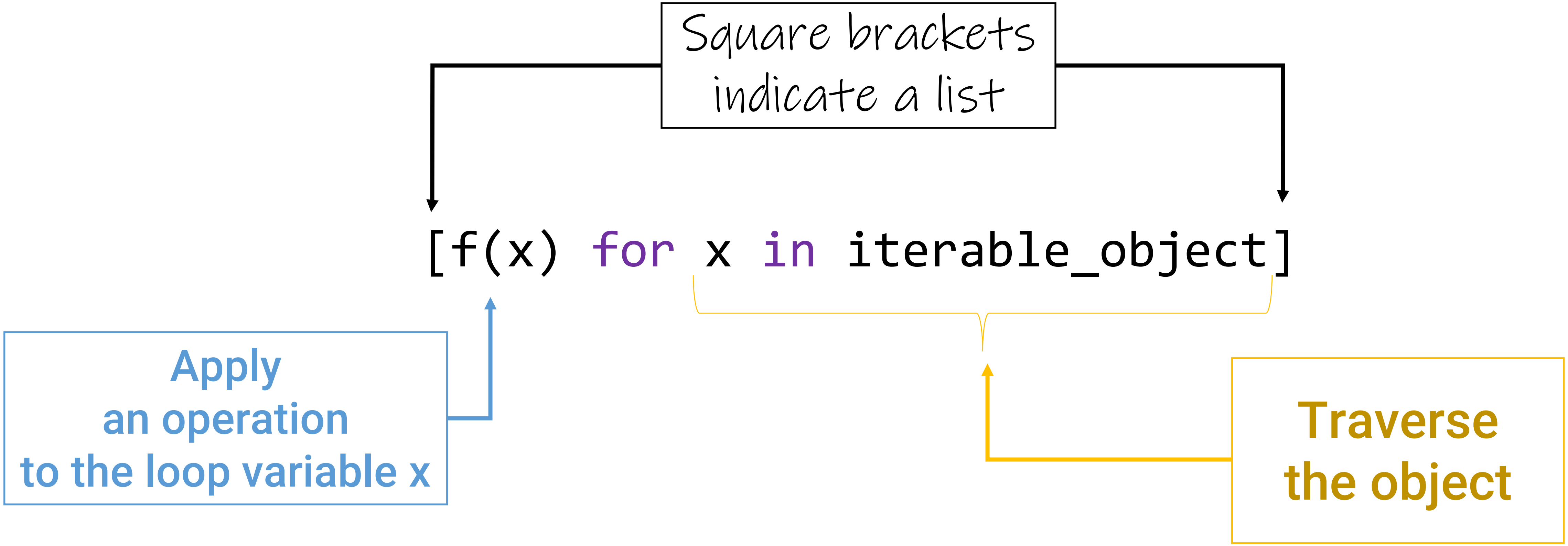
```
in_list = [43, -32, -94, -10, -18, 33, -59]
out_list = []  # Create an empty list to fill in

# Traverse the list
for i in range(len(in_list)):
     # Consider only elements at even indices
     if i % 2 == 0:
         out_list.append(in_list[i])
print(out_list)
# [43, -94, -18, -59]
```

List Comprehension ☘

© kras99 / Adobe Stock

List Comprehension

List comprehension is an easy way to **build a new list** by applying an expression to the items of an iterable object (i.e., a string, a list, …).



Square brackets indicate a list

```
[f(x) for x in iterable_object]
```

Apply an operation to the loop variable x

Traverse the object

Examples: List Comprehension

EXAMPLES

```
# range(start, stop, step)
[x**2 for x in range(1, 6, 2)]
# [1, 9, 25]

[c*4 for c in 'SPAM']
# ['SSSS', 'PPPP', 'AAAA', 'MMMM']

[c.lower() for c in 'SWEET STRAWBERRIES']
# ['s', 'w', 'e', 'e', 't', ' ', 's', 't', 'r', 'a', 'w', 'b',
'e', 'r', 'r', 'i', 'e', 's']
```

Homework

Read the remaining slides,
do the examples,
learn…
and ask for help if needed!

© kras99 / Adobe Stock

List Methods

For the summary on all list methods, click here

© kras99 / Adobe Stock

List Methods: Growing

Note: These methods modify the original list!

```
a = [0, 1.1, 2.2]

# Appending (argument is a new element)
a.append(3.3) # [0, 1.1, 2.2, 3.3]

# Extending (argument must be a list or a string)
a.extend([4.4, 5.5]) # [0, 1.1, 2.2, 3.3, 4.4, 5.5]

# Inserting (1st arg. = index where the inserted el. will be)
# (2nd argument = the element to insert)
a.insert(4, 0) # [0, 1.1, 2.2, 3.3, 0, 4.4, 5.5]
```

For the summary on all list methods, click here

List Methods: Searching and Counting

```
a = [0, 1.1, 2.2]

# Searching for an element
a.index(1.1)     # 1
a.index(3.3)     # ValueError: 3.3 is not in list

a = a * 2        # [0, 1.1, 2.2, 0, 1.1, 2.2]
# Count the number of occurences
a.count(0)       # 2
a.count(3.3)     # 0
```

For the summary on all list methods, click here

List Methods: Sorting and Reversing Order

Note: These methods modify the original list!

```
a = [0, 99, 3, 11, -5]
# Sorting
a.sort()  # increasing order of value
# [-5, 0, 3, 11, 99]
a.sort(reverse = True)  # decreasing order of value
# [99, 11, 3, 0, -5]

# Reversing order of elements
a = [0, 99, 3, 11, -5]
a.reverse()
# [-5, 11, 3, 99, 0]
```

For the summary on all list methods, click here

Modifying Lists Using Slicing

© kras99 / Adobe Stock

Modifying Lists Using Slicing

Replacing list elements by slicing is a combination of two steps:

1. **Deletion.**
   The slice you specify to the left of the assignment is deleted.
   If you specify an empty slice, nothing will be deleted.
2. **Insertion.**
   The new items to the right of the assignment operator are inserted into the list left to place of the old (deleted) slice.

*The number of inserted items does **not** have to match the number of deleted items!*

Examples: Modifying Lists Using Slicing

EXAMPLES

```
crepes = ['eggs', 'milk', 'flour', 'sugar']
len(crepes)       # 4

crepes[1:2] = []  # ['eggs', ~~'milk', ~~'flour', 'sugar']
len(crepes)       # 3
crepes            # ['eggs', 'flour', 'sugar’]

crepes[2:3] = ['milk', 'water', 'sugar']
len(crepes)       # 5
crepes            # ['eggs', 'flour', 'milk', 'water', 'sugar']
```

Summary of List methods

list.append(x)

Add an item to the end of the list; equivalent to a[len(a):] = [x].

list.extend(L)

Append all the items in the given list to extend the list; this is equivalent to a[len(a):] = L.

list.insert(i, x)

Insert an item at a given position. The first argument is the index the inserted element will have in the new list, so a.insert(0, x) inserts at the front of the list, and a.insert(len(a), x) inserts at the end of the list (because len(a) here is the length of the list before inserting x).

list.remove(x)

Remove the first item from the list whose value is x. If there is no such item, it is an error.

list.pop([i])

Remove the item from the list in the given position and return it. If no index is specified, a.pop() removes and returns the last item in the list. (The square brackets around the i in the method signature denote that the parameter is optional, not that you should type square brackets at that position. You will see this notation frequently in the Python Library Reference.)

list.index(x)

Return the index in the first item list whose value is x. If there is no such item, it is an error.

list.count(x)

Return the number of times x appears in the list.

list.sort()

Sort the items of the list, in place.

list.reverse()

Reverse the elements of the list, in place.

Next topic:
Nested Loops and Lists