Love It or List It

What are lists?

A **list** is a **data structure**, or collection of data values.

Lists allow us to store multiple values in one variable.

```
sea_creature_one = 'shark'
sea_creature_two = 'cuttlefish'
sea_creature_three = 'squid'
sea_creature_four = 'mantis shrimp'
sea_creature_five = 'anemone'
```

```
sea_creatures = ['shark', 'cuttlefish', 'squid', 'mantis shrimp', 'anemone']
```

Each value in a list is referred to as an **element**, or **item**, in the list.

What can be stored in a list?

Anything.

You can have a list with just one type of element...

```
fruits = ['apple', 'orange', 'banana'] # List of strings
```

a list with multiple types of elements...

```
hogwash = [2, 'foo', True, 3.14, None]
```

a list containing other lists...

```
alphabet_soup = ['a', 'b', ['cc', 'dd', ['eee', 'fff']], 'g', 'h']
```

or even an empty list.

```
cookie_jar = []
```

When should you use a list?

When you want to work with many related values.

Lists allow you to:

- Keep data together that belong together
- Condense your code

```
trial_one = 5.5
trial_two = 5.7
trial_three = 5.3

trials = [5.5, 5.7, 5.3]
```

- Perform the same operations on multiple values at once

```
>>> grades = [7, 9, 8]
>>> for i in range(len(grades)):
>>>   grades[i] = grades[i] + 1
>>> grades
[8, 10, 9]
```

How to create a list

Lists are defined by enclosing comma-separated elements within square brackets `[]`.

```
fruits = ['orange', 'apple', 'pear', 'banana', 'kiwi', 'apple', 'banana']
```

```
vec = [-4, -2, 0, 2, 4]
```

```
matrix = [
  [1, 2, 3, 4],
  [5, 6, 7, 8],
  [9, 10, 11, 12],
] # Contains nested lists
```

```
a = [-1, 1, 66.25, 333, 333, 1234.5]
```

```
browser_history = [] # Empty list
```

How to access an element in a list

Each element in a list has a position, or **index.** The first element of a list has an index of 0.

To access an element in a list, write the name of the list, followed by the bracket operator, `[]`. Put the index of the element you want to access within the brackets.

```
>>> fruits = ['orange', 'apple', 'pear', 'banana', 'kiwi', 'apple', 'banana']
>>> fruits[1]
'apple'
```

```
>>> vec = [-4, -2, 0, 2, 4]
>>> vec[-1] # Negative indices count backward from the end of the list
4
>>> vec[-3]
0
```

```
>>> def middle_index(list):
>>>   return len(list) // 2
>>> a = [-1, 1, 66.25, 333, 333, 1234.5]
>>> a[middle_index(a)] # Index can be any expression that evaluates to an integer (yay composition!)
```

How to access an element in a list

Use multiple bracket operators and indices to access elements within nested lists.

```
>>> matrix = [
  [1, 2, 3, 4],
  [5, 6, 7, 8],
  [9, 10, 11, 12],
]
>>> matrix[0]
[1, 2, 3, 4]
>>> matrix[0][2]
3
```

```
>>> alphabet_soup = ['a', 'b', ['cc', 'dd', ['eee', 'fff']], 'g', 'h']
>>> alphabet_soup[2]
['cc', 'dd', ['eee', 'fff']]
>>> alphabet_soup[2][2]
['eee', 'fff']
>>> alphabet_soup[2][2][0]
'eee'
```

What happens when you try to access an element that does not exist?

```
>>> vec = [-4, -2, 0, 2, 4]
>>> vec[len(vec)]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

Off-by-one errors when you base the index on the length of the list are very common because the indices start at 0!

How to assign an element of a list

In an assignment statement, the element to be assigned is identified on the left side using the bracket operator.

```
>>> alphabet_soup = ['a', 'b', ['cc', 'dd', ['eee', 'fff']], 'g', 'h']
>>> alphabet_soup[0] = 'z'
>>> alphabet_soup
['z', 'b', ['cc', 'dd', ['eee', 'fff']], 'g', 'h']
```

```
>>> fruits = ['orange', 'apple', 'pear', 'banana', 'kiwi', 'apple', 'banana']
>>> fruits[-2] = 'mango'
>>> fruits
['orange', 'apple', 'pear', 'banana', 'kiwi', 'mango', 'banana']
```

```
>>> matrix = [
  [1, 2, 3, 4],
  [5, 6, 7, 8],
  [9, 10, 11, 12],
]
>>> matrix[0][2] = 30
>>> matrix
[[1, 2, 30, 4], [5, 6, 7, 8], [9, 10, 11, 12]]
```

Can you add an element by assigning the index one above the index of the last element of the list?

```
>>> fruits = ['orange', 'apple', 'pear', 'banana', 'kiwi', 'apple', 'banana']
>>> fruits[7] = 'mango'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list assignment index out of range
```

Nope. The index must already exist to be assigned.

Instead, you can add elements using the `append` and `extend` functions...

How to check if an element is in a list

Use the `in` operator.

```
>>> fruits = ['orange', 'apple', 'pear', 'banana', 'kiwi', 'apple', 'banana']
>>> 'pear' in fruits
True
>>> 'watermelon' in fruits
False
```

```
>>> matrix = [
  [1, 2, 3, 4],
  [5, 6, 7, 8],
  [9, 10, 11, 12],
]
>>> [5, 6, 7, 8] in matrix
True
>>> [5, 6, 7] in matrix
False
```

How to check if an element is in a list

The `in` operator creates a boolean expression that evaluates to either `True` or `False`. You can use it anywhere a boolean expression is expected.

```
vec = [-4, -2, 0, 2, 4]
if -2 in vec:
  print('vec contains -2')
```

```
completed = [False, False, False]
i = 0
while False in completed:
  completed[i] = True
  i += 1
print(completed) # [True, True, True]
```

How to traverse a list using a `for` loop

There are two ways to traverse a list using a `for` loop:

1. Traverse the elements
2. Traverse the indices

How to traverse the elements in a list using a `for` loop

```
fruits = ['orange', 'apple', 'pear', 'banana', 'kiwi', 'apple', 'banana']
for fruit in fruits: # Ends with colon
  print(fruit) # Indented statements are run for each fruit in fruits
```

Let's break down this `for` loop:

- `for`: keyword that begins a `for` loop
- `fruit`: variable that stores the current element on each iteration of the `for` loop (you name it!)
  - First iteration: `fruit` equals `'orange'`
  - Second iteration: `fruit` equals `'apple'`
  - Third iteration: `fruit` equals `'pear'`
  - And so on and so forth...
- `in`: keyword that precedes the name of the list to iterate through
- `fruits`: list to iterate through

How to traverse the elements in a list using a `for` loop

```
>>> vec = [-4, -2, 0, 2, 4]
>>> for v in vec:
>>>   print(v + 1)
-3
-1
1
3
5
```

How to traverse the elements in a list using a `for` loop

```
>>> alphabet_soup = ['a', 'b', ['cc', 'dd', ['eee', 'fff']], 'g', 'h']
>>> for noodle in alphabet_soup:
>>>   print(noodle)
a
b
['cc', 'dd', ['eee', 'fff']]
g
h
```

```
>>> sea_creatures = ['shark', 'cuttlefish', 'squid', 'mantis shrimp', 'anemone']
>>> for creature in sea_creatures:
>>>   print(f'The {creature} lives in the 🌊!')
The shark lives in the 🌊!
The cuttlefish lives in the 🌊!
The squid lives in the 🌊!
The mantis shrimp lives in the 🌊!
The anemone lives in the 🌊!
```

How to traverse the elements in a list using a `for` loop

```
>>> matrix = [
  [1, 2, 3, 4],
  [5, 6, 7, 8],
  [9, 10, 11, 12],
]
>>> for row in matrix: # First iteration, row equals [1, 2, 3, 4]
>>>   for element in row: # First iteration, element equals 1
>>>     print(element)
1
2
3
4
5
6
7
8
9
10
11
12
```

How to traverse the indices of a list using a `for` loop

```
>>> fruits = ['orange', 'apple', 'pear']
>>> for fruit in fruits:
>>>   print(fruit)
orange
apple
pear
```

```
>>> fruits = ['orange', 'apple', 'pear']
>>> for i in range(len(fruits)):
>>>   print(i)
0
1
2
```

The structure of the `for` loop is the same:

- `for`: keyword that begins a `for` loop
- `i`: variable that stores the current element on each iteration of the `for` loop (you name it!)
- `in`: keyword that precedes the name of the list to iterate through

How to traverse the indices of a list using a `for` loop

If `range(len(fruits))` is the list that is being iterated through, let's look at what that list contains.

`range(stop)` is a function that takes a stop integer and returns something *like a list* that contains the integers 0 through `stop - 1`.

For instance, `range(3)` returns something like a list that contains the elements `0`, `1`, and `2`.

`range(2)` contains the elements `0` and `1`.

`range(5)` contains the elements `0`, `1`, `2`, `3`, and `4`.

`len(list)` is a function that takes a list and returns its length.

```
>>> fruits = ['orange', 'apple', 'pear']
>>> len(fruits)
3
```

So, `range(len(fruits))` gives you something like a list that contains all of the indices in `fruits`.

`len(fruits)` is `3`. `range(3)` contains `0`, `1`, and `2`.

How to traverse the indices of a list using a `for` loop

Importantly, the list-like something that `range` returns is *iterable*, meaning you can iterate through it as you would a list. That is why you can use it in a `for` loop after the `in` keyword:

```
>>> fruits = ['orange', 'apple', 'pear']
>>> for i in range(len(fruits)):
>>>   print(i)
0
1
2
```

When should you traverse the indices of a list using a `for` loop (rather than its elements)?

1. To assign an element in the list within an iteration of the `for` loop
2. To access elements in terms of the current element's index within an iteration of the `for` loop

When should you traverse the indices of a list using a `for` loop (rather than its elements)?

1. To assign an element in the list within an iteration of the `for` loop

```
>>> fruits = ['orange', 'apple', 'pear']
>>> for fruit in fruits:
>>>   # Want to assign each fruit to something else...
>>>   # But no way to do this with this for loop!
```

```
>>> fruits = ['orange', 'apple', 'pear']
>>> for i in range(len(fruits)):
>>>   # Ah, now we can use i to access each element in fruits
>>>   fruits[i] = fruits[i] + 's'
>>> fruits
['oranges', 'apples', 'pears']
```

When should you traverse the indices of a list using a `for` loop (rather than its elements)?

2. To access elements in terms of the current element's index within an iteration of the `for` loop

```
>>> numbers = [1, 3, 2, 4]
>>> for number in numbers:
>>>   # Want to check if numbers is sorted by comparing each number to the next...
>>>   # But, no way to get the next number with this for loop!
```

```
>>> numbers = [1, 3, 2, 4]
>>> for i in range(len(numbers) - 1):
>>>   print(f'Comparing {numbers[i]} to {numbers[i + 1]}...')
>>>   if numbers[i] > numbers[i + 1]:
>>>     print('Numbers are not sorted!')
>>>     break # Same behavior as in a while loop
Comparing 1 to 3...
Comparing 3 to 2...
Numbers are not sorted!
```

Why `range(len(numbers) - 1)`?

Traversing lists: What happens when you loop over an empty list?

```
>>> cookie_jar = []
>>> for cookie in cookie_jar:
>>>   print('Cookie!')
```

Zilch.

Traversing lists: How many elements does a list with nested lists contain?

```
>>> matrix = [
  [1, 2, 3, 4],
  [5, 6, 7, 8],
  [9, 10, 11, 12],
]
>>> for i in range(len(matrix)):
>>>   print(i)
```

Traversing lists: How many elements does a list with nested lists contain?

```
>>> matrix = [
  [1, 2, 3, 4],
  [5, 6, 7, 8],
  [9, 10, 11, 12],
]
>>> for i in range(len(matrix)):
>>>   print(i)
0
1
2
```

Each nested list counts as only one element.

How to concatenate lists

You can combine lists using the `+` operator (just like with strings!).

```
>>> my_favorite_numbers = [7, 25]
>>> your_favorite_numbers = [99, 1001]
>>> our_favorite_numbers = my_favorite_numbers + your_favorite_numbers
>>> our_favorite_numbers
[7, 25, 99, 1001]
```

```
>>> triangles = ['equilateral', 'isosceles', 'scalene']
>>> quadrilaterals = ['square', 'rectangle', 'rhombus']
>>> polygons = triangles + quadrilaterals
>>> polygons
['equilateral', 'isosceles', 'scalene', 'square', 'rectangle', 'rhombus']
```

```
>>> numbers = [0, 1, 2, 3]
>>> words = ['foo', 'bar']
>>> booleans = [True, False, False, True]
>>> combo = numbers + words + booleans
>>> combo
[0, 1, 2, 3, 'foo', 'bar', True, False, False, True]
```