www.careerist.com

# Lists

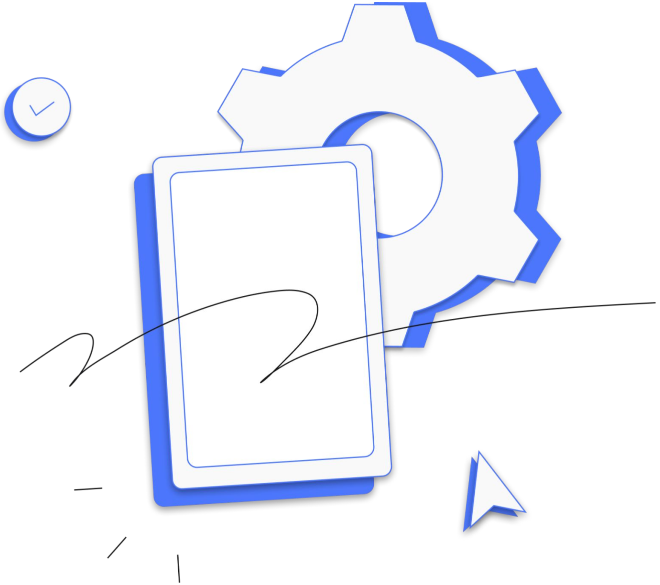
careerist

L6

# In this lesson you will learn

careerist

- What are lists?
- When should you use a list?
- How do you declare a list?
- Some properties of lists:
  - Zero-based Indexing
  - Mutable (what does this even mean?)
  - Ordered
  - Can store items of any type (even other lists)
- Some of the operations you can perform on a list, like retrieving, adding and removing elements, checking if elements exists, sorting, reversing, slicing and more!



# What are Lists?

- Lists are a type of collection in Python.
- A collection is like a file cabinet in memory, a place to keep elements together.
- Lists are the simplest collection type.

## When should you use a List?

- You need to store multiple items in a group.
- You need to iterate over the items and do something with each one of them.

## Are there other collection types in Python?

- Tuples
- Dictionaries
- Sets
- Named Tuples ☠️
- Arrays ☠️

# Properties of Lists: Zero-Based Indexing

- List elements are indexed automatically using Integers
- The first element always has an index of 0 (zero) and increments from there

```
# Indexes  0         1        2       3
my_list = ['Joseph', 'Kelly', 'Eric', 'Tom']

# my_list[0]: 'Joseph'
# my_list[1]: 'Kelly'
# my_list[2]: 'Eric'
# my_list[3]: 'Tom'

# Indexing is always consistent,
# regardless of how the contents change.

del my_list[1]

# One could expect the list to end up like this:

# my_list[0]: 'Joseph'
# my_list[2]: 'Eric'
# my_list[3]: 'Tom'
```

```
# BUT NO, list now looks like this:

# my_list[0]: 'Joseph'
# my_list[1]: 'Eric'
# my_list[2]: 'Tom'
```

# Properties of Lists: Mutability

**Lists are Mutable**, meaning that themselves (and their contents) can be changed after creation

```
# Create a list.
my_list = [1, "Joseph", True]

# Remove Index 1 of the list.
del my_list[1]

# Change the value of an element.
my_list[1] = "Eric" # Changes index 1 from “Joseph” to “Eric”.

# Add more elements at the end.
my_list.append("Kelly")
```

# Properties of Lists: Ordered

**List Elements** are always kept in the list in the order in which they were inserted

```
# Create an empty list.
my_list = []

# Add a few items in a particular order.
my_list.append("Joseph")    # Index 0
my_list.append("Kelly")     # Index 1
my_list.append("Eric")      # Index 2

print(my_list[0]) # Joseph
print(my_list[1]) # Kelly
print(my_list[2]) # Eric
```

# Properties of Lists: Heterogeneity

Heterogeneity: It’s a big ugly word, but all it means is that lists can store elements of any type

```
my_list = [1, 2, 3, 4, 5]
my_list = ["Joseph", "Kelly", "Eric"]
my_list = [True, True, False]
my_list = [None, None, None]

# Store different data types at the same time, even other lists and other collection types.

my_list = [
  1,                         # Integers
  2.00,                      # Floats
  True,                      # Booleans
  "Joseph",                  # Strings
  None,                      # None (Python's NULL)
  [100,  200, 300],          # Other lists
  {"Cars", "Motorcycles"},   # Sets, dictionaries, classes - Anything!
]
```

# Creating lists

- Lists are ALWAYS declared using square brackets []
- List elements are ALWAYS separated by commas

```
# You can create an empty list by using empty square brackets []
my_list = []

# You can create a pre-populated list by declaring the elements inside the brackets.
my_list = [1, 2, 3]
my_list = ['Joseph', 'Kelly', 'Eric']

# You can also use variables.
employee_1 = 'Joseph'
employee_2 = 'Kelly'
employee_3 = 'Eric'

my_list = [employee_1, employee_2, employee_3]

# REMEMBER: You can mix and match data types. Even other lists!
my_list = [1, 1.00, False, None, ['Joseph', 'Kelly']]
```

# Retrieving individual elements

The easiest way to retrieve elements from a list is using their **[index]**

```
# Indexes  0         1        2       3      4
my_list = ["Joseph", "Kelly", "Eric", "Tom", [100, 200]]

# Access individual elements directly using "positive indexing".
my_list[0]  # Joseph
my_list[3]  # Tom

# Access elements from the end of the list using "negative indexing".
my_list[-1]  # Whatever the last item is, in this case the list [100, 200]
my_list[-2]  # Whatever the second to last item is, in this case "Tom"

# Access a list element inside another list
my_list[4][0]  # 100
my_list[4][1]  # 200
```

# Practice!

```python
# Create a list that holds the following information about your location.
# - The first element should be the city name.
# - The second element should be the state or province.
# - The third element should be a list containing the maximum temperatures
#   of the last three days.

my_list = ???

# Now, fill in the print statements to display the following information:
print(???) # - The city name
print(???) # - The state or province
print(???) # - The list of temperatures

# Bonus
print(???) # - The first temperature of the list of temperatures.
```

# Practice! (solution)

```python
# Create a list that holds the following information about your location.
# - The first element should be the city name.
# - The second element should be the state or province.
# - The third element should be a list containing the maximum temperatures
#   of the last three days.

my_list = ["Houston", "Texas", [101, 104, 103]]

# Now, create print statements to display the following information:
print(my_list[0]) # - The city name: Houston
print(my_list[1]) # - The state or province: Texas
print(my_list[2]) # - The list of temperatures: [101, 104, 103]

# Bonus
print(my_list[2][0]) # - The first temperature of the list of temperatures: 101
```

# Add and Remove Elements

```
# We will start with an empty list, but you could also start with a pre-populated list.
my_list = []

# You can use "append", which adds a single item to the end of the list.
my_list.append("Joseph")  # ["Joseph"]
my_list.append("Tom")     # ["Joseph", "Tom"]
my_list.append("Kelly")   # ["Joseph", "Tom", "Kelly"]

# To remove items from the list, you use the "del" statement, with the index.
del my_list[1]   # Removes "Tom"
del my_list[-1]  # Removes "Kelly" (last)

# You can also remove by value. This will remove the first value it finds.
my_list.remove("Tom")

# Warning: Using "del" directly on the list itself removes the whole list from memory.
del my_list
```

# Practice!

```
# Create an empty list and add the following elements about your location, in this order:
# - City
# - State or Province
# - A list with the temperatures the last three days
# - Your favorite animal

my_list = ???
my_list.a???
my_list.a???
my_list.a???
my_list.a???

# Then, remove the State, without using the indexes.
my_list.r???

# Bonus: Remove the last element, using a negative index.
??? my_list[???]
```

# Practice! (solution)

```python
# Create an empty list and add the following elements about your location, in this order:
# - City
# - State or Province
# - A list with the temperatures the last three days
# - Your favorite animal

my_list = []
my_list.append("Houston")
my_list.append("Texas")
my_list.append([101, 102, 103])
my_list.append("Sloth")

# Then, remove the State, without using the indexes.
my_list.remove("Texas")

# Bonus: Remove the last element, using a negative index.
del my_list[-1]
```

# Check if an element exists

An easy way to check if an element exists in a list, is to use the “in” or “not in” statements

```python
my_list = ["Houston", "Texas", [101, 102, 103]]

print("Houston" in my_list)      # Prints True
print("Texas" in my_list)        # Prints True
print(101 in my_list)            # Prints False, why? 🤔
print(101 in my_list[2])         # Prints True

# We can use it as part of an "if" statement.
if "Houston" in my_list:
   print("Houston was found in the list")

if "Austin" not in my_list:
   print("Austin was NOT found in the list")
```

# Sort a list

There are two main ways to sort a list:

- **.sort()** to sort a list in place, which modifies the original list
- **sorted()** to sort by returning a copy, which leaves the original list unchanged

# Sort a list

```python
# Sorting in place: Use this when the original order is not important. Saves memory.
my_list = [5, 4, 7, 2, 1]
my_list.sort()

print(my_list)  # [1, 2, 4, 5, 7]

# Sorting to a new copy: Use this when the original order is important.
my_list = [5, 4, 7, 2, 1]
my_sorted_list = sorted(my_list)

print(my_list)         # [5, 4, 7, 2, 1]
print(my_sorted_list)   # [1, 2, 4, 5, 7]
```

**TIP TO REMEMBER:**

"Sort to distort", as it distorts 🥴the original list.

# Reverse a list

There are two main ways to reverse a list: reversing in place or to a new copy.

```
# Reversing in place using the "reverse" method, which reverses the original list.

my_list = [5, 4, 7, 2, 1]
my_list.reverse()

print(my_list)  # [1, 2, 7, 4, 5]

# Reversing to a new copy of the list using the "reversed" function,
# which leaves the original list unchanged, but needs to be cast to a list.

my_list = [5, 4, 7, 2, 1]
my_reversed_list = list(reversed(my_list))

print(my_list)            # [5, 4, 7, 2, 1]
print(my_reversed_list)   # [1, 2, 7, 4, 5]
```

# Practice!

```
# Create a list that contains the ingredients for a sandwich. Yum.🥪
# If you want to use an empty list and add stuff to it, or start
# with a pre-populated list is up to you.

# Some people like cheese, some people don't. ONLY add cheese to your
# list of ingredients if you really like cheese.🧀

my_list = ???

# Then, use an if/else statement to print a message that will tell us
# whether you like cheese, based on its presence in the list.



???


# Then, to make it look pretty, sort the list in alphabetical order and print it out.
# Our computer is very old and it doesn't have a lot of memory. Also, we don't care
# about the original order of the ingredients.

???
print(my_list)
```

# Practice! (solution)

careerist

```python
# Create a list that contains the ingredients for a sandwich. Yum.🥪
# If you want to use an empty list and add stuff to it, or start
# with a pre-populated list is up to you.

# Some people like cheese, some people don't. ONLY add cheese to your
# list of ingredients if you really like cheese.🧀

my_list = ["bread", "ham", "tomato", "cheese"]

# Then, use an if/else statement to print a message that will tell us
# whether you like cheese, based on its presence in the list.

if "cheese" in my_list:
   print("I love cheese")
else:
   print("I hate cheese")

# Then, to make it look pretty, sort the list in alphabetical order and print it out.
# Our computer is very old and it doesn't have a lot of memory. Also, we don't care
# about the original order of the ingredients.

my_list.sort()
print(my_list)
```

# Concatenation

- Concatenation means stitching stuff together using the + operator
- When you concatenate lists, the order of the elements is preserved

```
my_list_1 = ["Joseph", "Kelly"]
my_list_2 = ["Tom", "Bernard"]

my_concatenated_list = my_list_1 + my_list_2
print(my_concatenated_list)  # ['Joseph', 'Kelly', 'Tom', 'Bernard']

my_concatenated_list = my_list_2 + my_list_1
print(my_concatenated_list)  # ['Tom', 'Bernard', 'Joseph', 'Kelly']

# Another way to stitch lists together is to multiply them with
# the * operator, which repeats them.
my_list = [1, 2, 3]
my_repeated_list = my_list * 3

print(my_repeated_list)  # [1, 2, 3, 1, 2, 3, 1, 2, 3]
```

# List slicing

careerist

- Slicing is used to extract a contiguous portion of a list
- The original list remains unchanged

```
# Syntax: new_list = original_list[start:end:optional_step]

# Indexes: 0         1        2       3      4          5       6
my_list = ["Joseph", "Kelly", "Eric", "Tom", "Bernard", "Jack", "Josh"]

# Using the slicer without any parameters just returns the original list.
print(my_list[:])   # ["Joseph", "Kelly", "Eric", "Tom", "Bernard", "Jack", "Josh"]
print(my_list[::])  # ["Joseph", "Kelly", "Eric", "Tom", "Bernard", "Jack", "Josh"]

# Get the first two elements.
my_slice = my_list[0:2]  # ['Joseph', 'Kelly']

# Get the second, third and fourth elements.
my_slice = my_list[1:4]  # ['Kelly', 'Eric', 'Tom']
```

# List slicing

- You can use an optional step parameter to extract every N elements

```
my_list = ["Joseph", "Kelly", "Eric", "Tom", "Bernard", "Jack", "Josh"]

# Stepped slicing, get every N items.
# Python will take N steps before grabbing an element.

my_slice = my_list[::2]    # ['Joseph', 'Eric', 'Bernard', 'Josh']
my_slice = my_list[0:4:2]  # ['Joseph', 'Eric']
```

# Practice!

```
# For this exercise, you will create two lists. The first list will contain the work week days,
# the second list will contain the days of the weekend.
work_days = ???
weekend_days = ???

# Then, you need to concatenate both lists into a third new list that represents the full week.
full_week = ???

# Then, can you think of an easy way to create a new list that contains the days of two full
weeks?
fortnight = ???

# Finally, once you have two full weeks into a list, use the slicer to:
# 1. Extract week 1 into its own list. Use this list in the next two points.
week_1 = ???

# 2. Write a slicer that will return the following: ['monday', 'wednesday', 'friday', 'sunday']
print(week_1[???])
```

# Practice! (solution)

```
# For this exercise, you will create two lists. The first list will contain the work week days,
# the second list will contain the days of the weekend.
work_days = ['monday', 'tuesday', 'wednesday', 'thursday', 'friday']
weekend_days = ['saturday', 'sunday']

# Then, you need to concatenate both lists into a third new list that represents the full week.
full_week = work_days + weekend_days

# Then, can you think of an easy way to create a new list that contains the days of two full
weeks?
fortnight = full_week * 2

# Finally, once you have two full weeks into a list, use the slicer to:
# 1. Extract week 1 into its own list. Use this list in the next two points.
week_1 = fortnight[0:7]

# 2. Write a slicer that will return the following: ['monday', 'wednesday', 'friday', 'sunday']
print(week_1[::2])
```

# Aggregators

careerist

Aggregators are special functions that help us perform some basic list calculations

```
# Use min() to determine the smallest number in a list of numbers or
# the earliest alphabetical element in a list of strings. Same type only!

min([1, 2, 3])          # 1
min(['c', 'a', 'd'])    # a

# Use max() exactly the same way, but for the highest value.
max([1, 2, 3])          # 3
max(['a', 'b', 'c'])    # c

# Use sum() to add up the total in a list of numbers. Only things that evaluate to numbers!
sum([1, 2, 3])                      # 6
sum([1, 2.0, 3.5])                  # 6.5
sum([1, 2.0, 3.5, True, False])     # 7.5
```

# Helpers

Helpers can tell us some useful information about the list or its elements

```
# Use len() to find out the size of a list.
len([1, 2, 3, 4, 5])            # 5
len([[1, 2, 3], [4, 5, 6]])     # 2 - Why? 🤔

# Use my_list.index() to find the index of an element.
my_list = ["Joseph", "Kelly", "Tom"]
my_list.index("Kelly")  # 1
my_list.index("Tom")    # 2

# Use my_list.count() to find out how many times an element is in the list.
my_list = [1, 2, 3, 4, 4, 5, 6, 6, 6, 7]
my_list.count(2)  # 1
my_list.count(4)  # 2
my_list.count(6)  # 3

my_list = [True, True, False]
my_list.count(True)   # 2
my_list.count(False)  # 1
```

# Practice!

careerist

```
# A company opened in 2010 and ceased operations in 2014.
# Imagine the following list contains the number of
# employees the company for each year:

# Year:      2010   2011    2012    2013    2014
employees = [  93,   104,     89,    101,     93]

# We would like to know the following
# 1. What's the lowest number of employees the company ever had?
print(???)

# 2. What's the highest number of employees the company ever had?
print(???)

# 3. What's the total head count if all employees were different every year?
print(???)

# 4. How many years had 93 employees?
print(???)

# 5. Can you think of a way to determine how many years the company was in business?
#    Hint: If it's one list element per year, maybe you can count the number of elements.
print(???)
```

# Practice! (solution)

```python
# A company opened in 2010 and ceased operations in 2014.
# Imagine the following list contains the number of
# employees the company for each year:

# Year:      2010   2011    2012    2013    2014
employees = [  93,   104,     89,    101,     93]

# We would like to know the following
# 1. What's the lowest number of employees the company ever had?
print(min(employees))  # 89

# 2. What's the highest number of employees the company ever had?
print(max(employees))  # 104

# 3. What's the total head count if all employees were different every year?
print(sum(employees))  # 480

# 4. How many years had 93 employees?
print(employees.count(93))  # 2 (2010 and 2014)

# 5. Can you think of a way to determine how many years the company was in business?
#    Hint: If it's one list element per year, maybe you can count the number of elements.
print(len(employees))  # 5
```

# Congratulations!

careerist

Now you know how to:

- Create empty and pre-populated lists.
- Manipulate lists to add and remove elements.
- Sort, reverse and slice lists.
- Concatenate lists.
- Use Aggregators and Helpers on lists.

# Extra [self-study]: Tuples

Tuples are like lists, but with the following differences:

- **Tuples are declared using parenthesis ()** instead of square brackets [].
- **Tuples are immutable**. Once you create a tuple, you can't change it or its contents. They don't have any methods that change it, like append, remove, sort, reverse. But they have methods and functions that don't change it: min, max, sum, index, count.
- **Tuples are faster** and more memory efficient, because they don't create an extra copy of the data.
- **Why would you use a tuple?** You don't need (or want) to modify the elements later. Tuples are faster.

# Extra: Tuples

```python
# When declaring a tuple, you have to pre-populated. You can’t add elements later.
my_tuple = ("Joseph", "Kelly", 1, 2, 3, [100, 200, 300])

# This won't work
my_tuple.append("Eric")
my_tuple.remove("Joseph")

# This works just fine
my_tuple.index("Kelly")  # 1
my_tuple.count("Joseph") # 1

# ANNOYANCE: If you ever need a tuple with a single element,
# you have to include a trailing comma.
my_tuple = ("Joseph", )
```

# Extra [self-study]: Sets

Sets are like somewhat similar to lists, but with the following differences:

- **Sets are declared using curly braces {}** instead of square brackets [].
- **Sets are unordered**. No matter how elements are inserted, the storage order is unpredictable.
- **Sets are mutable, but they only allow immutable elements**. You can add or remove elements, but they have to be immutable data types: strings, numbers, tuples, booleans. No lists, dictionaries, or other sets.
- **Sets are unique**. Even if you add the same element multiple times, it will be on the set just once.
- **Why would you use a set?** You need a collection of unique elements of the allowed data types (numbers, strings, tuples, booleans, None) and you don’t care about the order.

# Extra: Sets

```python
# Sets have a variety of methods and functions that allow you to manipulate them. Here’s a few:
# Create a set.
my_set = {"Joseph", "Kelly"}
print(my_set)  # {'Kelly', 'Joseph'}

# Adding my name again doesn't make it show twice due to uniqueness.
my_set.add("Joseph")
print(my_set)  # {'Kelly', 'Joseph'}

# Remove an element using remove(). If you try to remove an element that doesn't exist,
# you get an error.
my_set.remove("Joseph")   # Removes the element.
my_set.remove("Joseph")   # Raises a KeyError, as the element doesn't exist anymore.
my_set.discard("Joseph")  # Same as remove(), but doesn't raise an error if it doesn't exist.

# ANNOYANCE: You have to use set() to create an empty set, as {} is used to declare a
dictionary.
my_empty_set = {}     # NOT what you want. This creates a dictionary.
my_empty_set = set()  # THIS is what you want. This creates an empty set.
```

# Homework

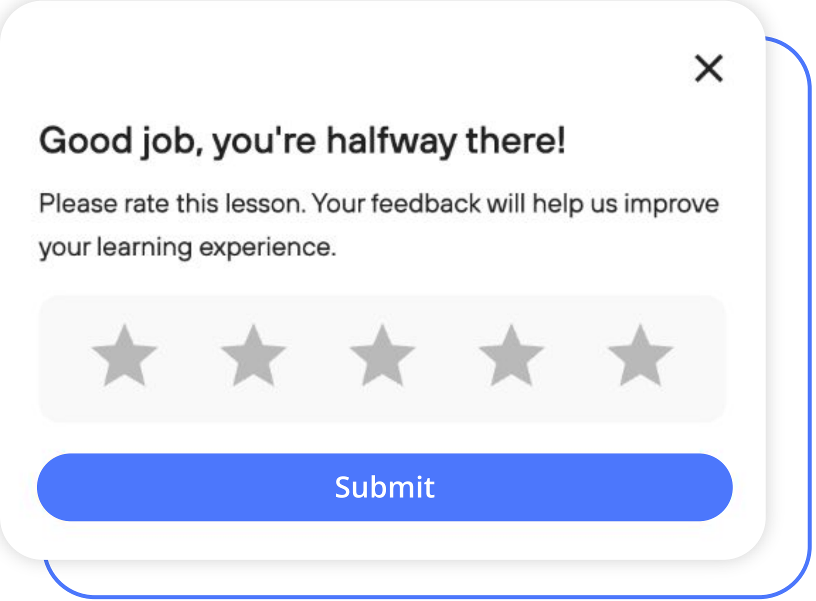
careerist

- → Pull the repository before starting your homework
- → Complete homework for Lesson 6 in your PyCharm
- → Upload the solutions to Git Gist
- → Send it to us for review through your Careerist Learning Space

# Feedback

careerist

Please take a moment to share your thoughts on this lesson in your **Careerist Learning Space**



Good job, you're halfway there!

Please rate this lesson. Your feedback will help us improve your learning experience.

# Questions

?

Take your career
to the next level!

careerist

✉ support@careerist.com

🌐 www.careerist.com

☎ 1 415 862-2563