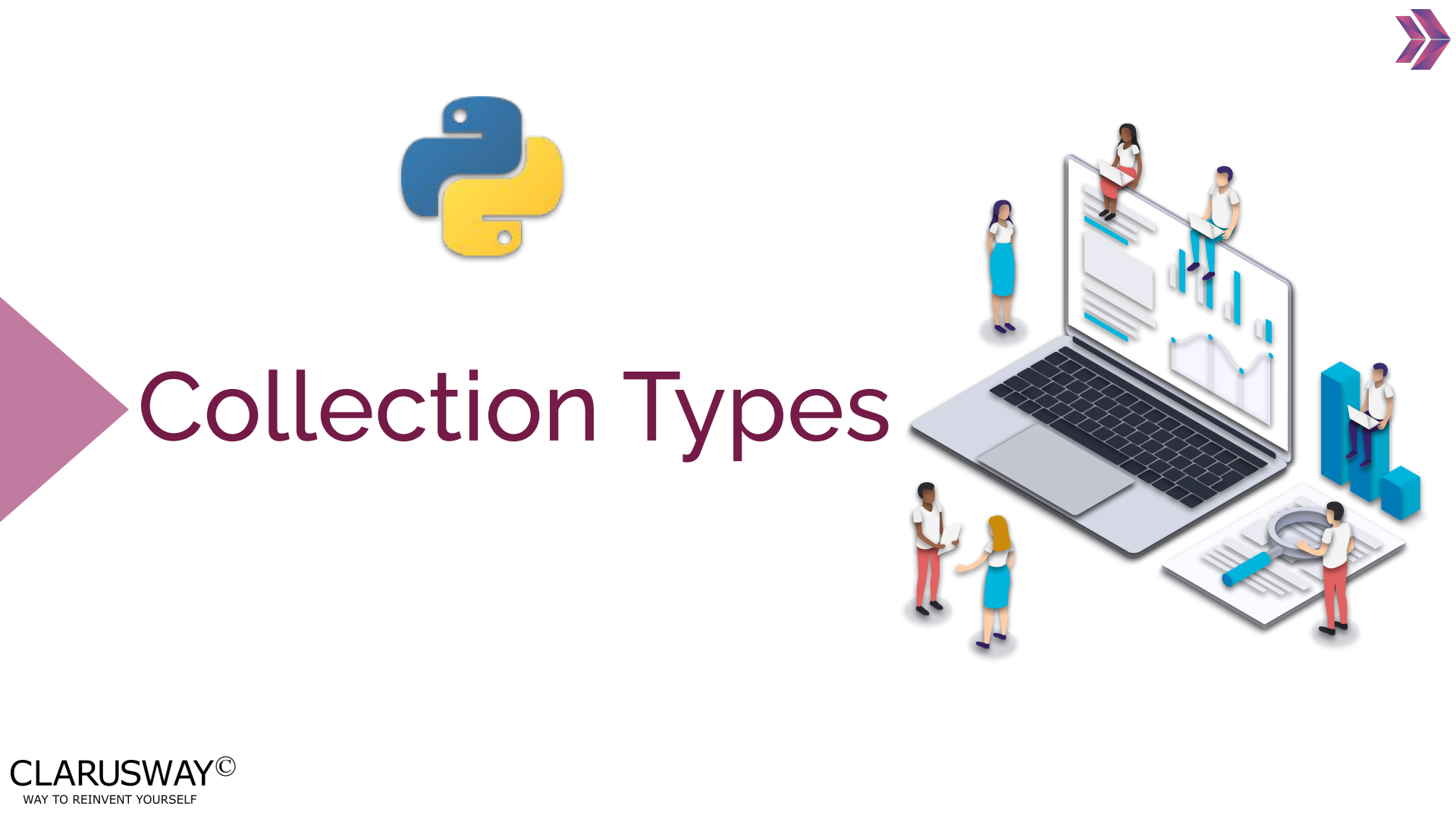


# Collection Types

# Introduction

- **What is a collection type?**
  - There are various collection types in Python. While types such as `int` and `str` hold a **single value**, collection types hold *multiple values*.
  - In your programs, you usually need to group several items to render as a single object. We use collection types of data to do this job.

# Collection Types

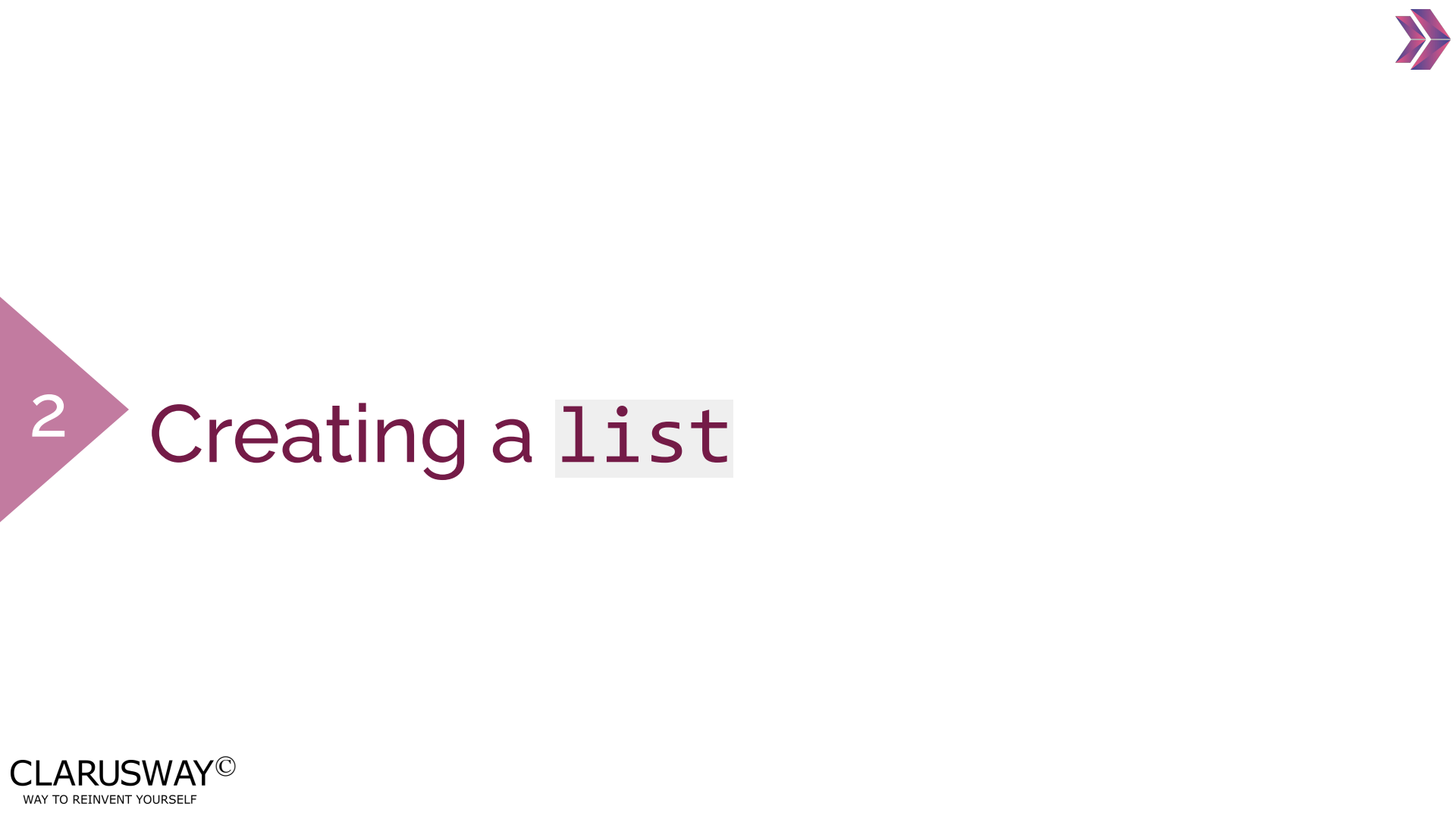
- List
- Tuple
- Dictionary
- Set

# Lists

# Table of Contents

# Introduction

- **What is a `list` ?**
  - One of the most useful collections in Python is a `list`.
  - In Python, a `list` is only an ordered collection of valid Python values.
  - The `list` type is probably the most commonly used collection type in Python.


2
Creating a list

# Creating a `list`

- A `list` can be created by enclosing values, separated by commas, in square brackets 👉 `[]`.
- Another way to create a `list` is to call the `list()` function.

- `[]`
- `list()`

# Creating a `list`

- A `list` can be created by enclosing values, separated by commas, in square brackets 👉 `[]`.
- Another way to create a `list` is to call the `list()` function.

- `[]`
- `list()`

```
list_1 = ['h', 'a', 'p', 'p', 'y']
word = 'happy'
list_2 = list(word)
print(list_1)
print(list_2)
```

What is the output? Try to figure out in your mind...

# Creating a `list`

- A `list` can be created by enclosing values, separated by commas, in square brackets 👉 `[]`.
- Another way to create a `list` is to call the `list()` function.

- `[]`
- `list()`

```
list_1 = ['h', 'a', 'p', 'p', 'y']
word = 'happy'
list_2 = list(word)
print(list_1)
print(list_2)
```

```
['h', 'a', 'p', 'p', 'y']
['h', 'a', 'p', 'p', 'y']
```

# Creating a `list` (review of pre-class)

- Here is another example of creating a `list` :

```
1  country = ['USA', 'Brasil', 'UK', 'Germany', 'Turkey', 'New Zealand']
2
3  print(country)
4
```

# Creating a `list` (review of pre-class)

Here is another example of creating a `list` :

```
1  country = ['USA', 'Brasil', 'UK', 'Germany', 'Turkey', 'New Zealand']
2
3  print(country)
4
```

```
1  ['USA', 'Brasil', 'UK', 'Germany', 'Turkey', 'New Zealand']
2
```

# Creating a `list` (review of pre-class)

Here is another example of creating a `list` :

```
1  country = ['USA', 'Brasil', 'UK', 'Germany', 'Turkey', 'New Zealand']
2
3  print(country)
4
```

```
1  ['USA', 'Brasil', 'UK', 'Germany', 'Turkey', 'New Zealand']
2
```

**Tips:**

- All the country names are printed in the same order as they were stored in the list because lists are **ordered**.

# Creating a `list` (review of pre-class)

- Here is another example of creating a `list` using `list()`.
- We can do it when we want to create a `list` from an iterable object: that is, type of object whose elements you can import individually. The `list`s are iterable like other collections and `str`ing types.

```
1  string_1 = 'I quit smoking'
2
3  new_list_1 = list(string_1)  # we created multi element list
4  print(new_list_1)
5
6  new_list_2 = [string_1]  # this is a single element list
7  print(new_list_2)
8
```

What is the output? Try to figure out in your mind...

# Creating a `list` (review of pre-class)

- Here is another example of creating a `list` using `list()`.
- We can do it when we want to create a `list` from an iterable object: that is, type of object whose elements you can import individually. The `list`s are iterable like other collections and `str`ing types.

```
1  string_1 = 'I quit smoking'
2
3  new_list_1 = list(string_1)  # we created multi element list
4  print(new_list_1)
5
6  new_list_2 = [string_1]  # this is a single element list
7  print(new_list_2)
8
```

```
1  ['I', ' ', 'q', 'u', 'i', 't', ' ', 's', 'm', 'o', 'k', 'i', 'n', 'g']
2  ['I quit smoking']
3
```

# Creating a `list`

**Tips:**

- Note that, using `list()` function, all characters of string_1 including spaces was moved into a new_list_1.
- If you noticed, **lists** can contain **more than one** of the **same** value.

```
1  string_1 = 'I quit smoking'
2
3  new_list_1 = list(string_1)  # we created multi element list
4  print(new_list_1)
5
6  new_list_2 = [string_1]  # this is a single element list
7  print(new_list_2)
8
```

```
1  ['I', ' ', 'q', 'u', 'i', 't', ' ', 's', 'm', 'o', 'k', 'i', 'n', 'g']
2  ['I quit smoking']
3
```

# Creating a `list`

**Tips:**

- Note that, using `list()` function, all characters of string_1 including spaces was moved into a new_list_1.
- If you noticed, **lists** can contain **more than one** of the **same** value.

```
1  string_1 = 'I quit smoking'
2
3  new_list_1 = list(string_1)  # we created multi element list
4  print(new_list_1)
5
6  new_list_2 = [string_1]  #  gle element list
7  print(new_list_2)
8
```

⚠ It has only one element.

⚠ It has 14 elements.

```
1  ['I', ' ', 'q', 'u', 'i', 't', ' ', 's', 'm', 'o', 'k', 'i', 'n', 'g']
2  ['I quit smoking']
3
```

# Creating a `list`

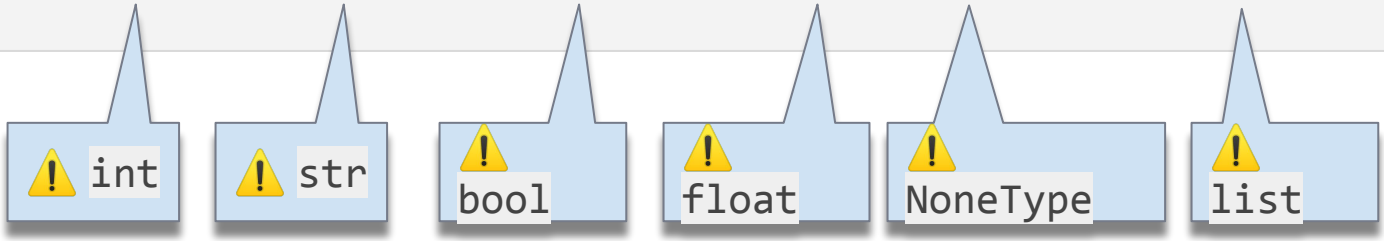
- The components of a `list` are not limited to a single data type, given that Python is a dynamic language.
- Here is an example :

```
mixed_list = [11, 'Joseph', False, 3.14, None, [1, 2, 3]]
```

# Creating a `list`

- The components of a `list` are not limited to a single data type, given that Python is a dynamic language.
- Here is an example :

```
mixed_list = [11, 'Joseph', False, 3.14, None, [1, 2, 3]]
```



# Creating a `list`

- Task:
  - Try to figure out the output of these list operations in your mind.

```
my_list = ['Joseph', 'Clarusway', 2022]
new_list1 = list(my_list) # what will be the output?
new_list2 = [my_list]  # what will be the output?

print(new_list1)
print(len(new_list1))  # what is the length of the variable?

print(new_list2)
print(len(new_list2))
```

# Creating a `list`

- The output:.

```
['Joseph', 'Clarusway', 2022]
3
[['Joseph', 'Clarusway', 2022]]
1
```

# Creating a `list`

- Task:
  - Create a list from **string** value of `"2022's hard"`.
  - Use both `list()` function and square brackets `[]`.

# Creating a `list`

- The code :

```
my_list1 = ["2022's hard"]
my_list2 = list("2022's hard")

print(my_list1)
print(my_list2)
```

Each item/char including spaces came into the list

```
["2022's hard"]
['2', '0', '2', '2', "'", 's', ' ', 'h', 'a', 'r', 'd']
```

# 3 Basic Operations with Lists

# Basic Operations with `lists`

- In most cases, we'll have to make an empty `list` to fill it later with the data you want.

How to create an empty list?

# Basic Operations with `lists`

- In most cases, we'll have to make an empty `list` to fill it later with the data you want.

```
empty_list_1 = []
empty_list_2 = list()
```

Two methods for creating an empty list.

# Basic Operations with `lists`

- We can add an element into a list using `.append()` or `.insert()` methods.
- `.append()` appends an object to the end of a `list`.

- `.append()`
- `.insert()`

# Basic Operations with `lists`

- We can add an element into a list using `.append()` or `.insert()` methods.
- `.append()` appends an object to the end of a `list`.

- `.append()`
- `.insert()`

```
numbers = [1, 4, 7]
numbers.append(9)
numbers.append('9')
print(numbers)
```

What is the output? Try to figure out in your mind...

# Basic Operations with `lists`

- We can add an element into a list using `.append()` or `.insert()` methods.
- `.append()` appends an object to the end of a `list`.

- `.append()`
- `.insert()`

```
numbers = [1, 4, 7]
numbers.append(9)
numbers.append('k')
print(numbers)
```

```
[1, 4, 7, 9, 'k']
```

# Basic Operations with `lists`

- Take a look at the example 👇

```
1  empty_list = []
2  empty_list.append(6666)
3  empty_list.append('Multiverse')
4  empty_list.append([0])
5
6  print(empty_list)
7
8
```

What is the output? Try to figure out in your mind...

# Basic Operations with `lists`

- Take a look at the example 👇

```
1  empty_list = []
2  empty_list.append(6666)
3  empty_list.append('Multiverse')
4  empty_list.append([0])
5
6  print(empty_list)
7
8
```

Output

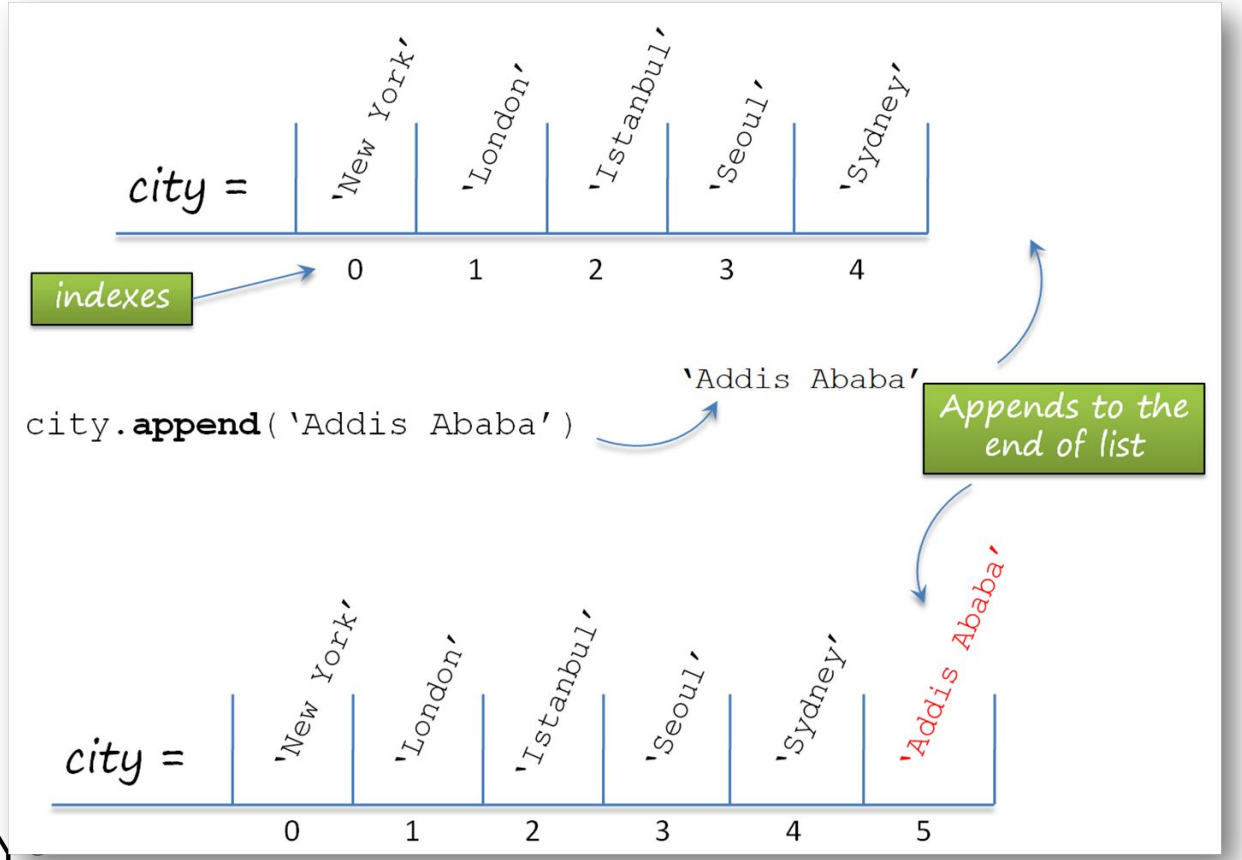
```
[6666, 'Multiverse', [0]]
```

# Basic Operations with `lists`

- Here's the pre-class example 👇

```
1  city = ['New York', 'London', 'Istanbul', 'Seoul', 'Sydney']
2  city.append('Addis Ababa')
3
4  print(city)
5
```

# Basic Operations with `lists`



# Basic Operations with `lists`

- Here's an another example 👇

```
1  city = ['New York', 'London', 'Istanbul', 'Seoul', 'Sydney']
2  city.append('Addis Ababa')
3
4  print(city)
5
```

```
1  ['New York', 'London', 'Istanbul', 'Seoul', 'Sydney', 'Addis Ababa']
2
```

# Basic Operations with `lists`

- **Task** :
  - Create an empty `list` and then collect the `int` numbers (1 - 4) one by one into the `list` you created using `.append()` method.

# Basic Operations with `lists`

- The code can be like :

```
numbers = []
numbers.append(1)
numbers.append(2)
numbers.append(3)
numbers.append(4)

print(numbers)
```

```
[1, 2, 3, 4]
```

# Basic Operations with `lists`

- `.insert()` adds  a new object to the `list` at a specific (given) **index**.

- `.append()`
- `.insert()`

# Basic Operations with `lists`

- `.insert()` adds a new object to the `list` at a specific (given) index.

- `.append()`
- `.insert()`

```
numbers = [1, 4, 7]
numbers.insert(2, 9)
print(numbers)
numbers.insert(2, 6)
print(numbers)
```

What is the output? Try to figure out in your mind...

# Basic Operations with `lists`

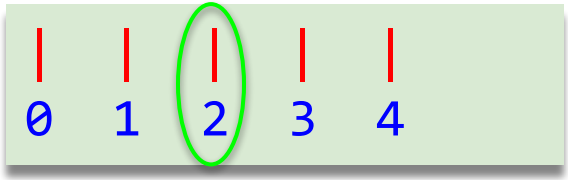
- `.insert()` adds a new object to the `list` at a specific (given) index.

- `.append()`
- `.insert()`

```
numbers = [1, 4, 7]
numbers.insert(2, 9)
print(numbers)
numbers.insert(2, 6)
print(numbers)
```

Adds into index '2'

```
[1, 4, 9, 7]
[1, 4, 6, 9, 7]
```



# Basic Operations with `lists`

- Consider this pre-class example 👇

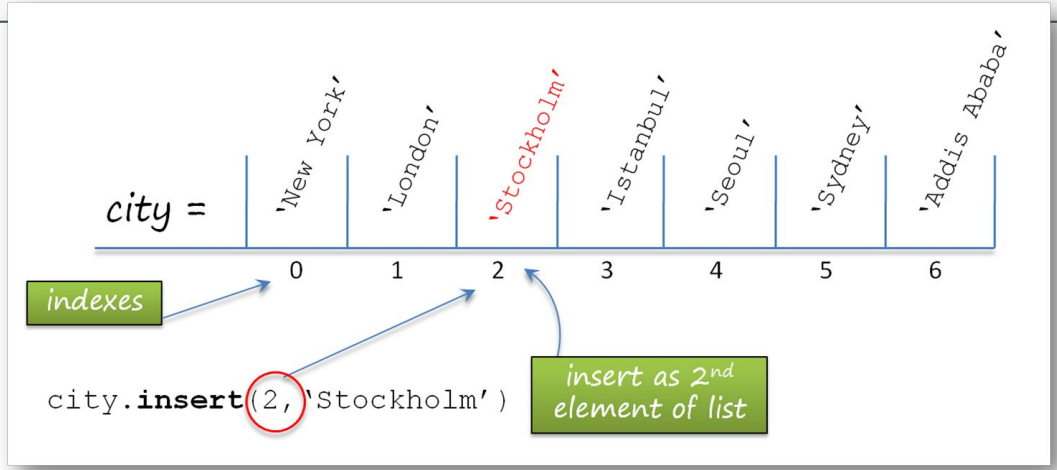
```
1  city = ['New York', 'London', 'Istanbul', 'Seoul', 'Sydney', 'Addis Ababa']
2  city.insert(2, 'Stockholm')
3
4  print(city)
5
```

# Basic Operations with `lists`

- Consider this pre-class example 👇

```
1  city = ['New York', 'London', 'Istanbul', 'Seoul', 'Sydney', 'Addis Ababa']
2  city.insert(2, 'Stockholm')
3
4  print(city)
5
```

```
1  ['New York', 'London', 'Stockholm', 'Istanbul', 'Seoul', 'Sydney', 'Addis Ababa']
2
```

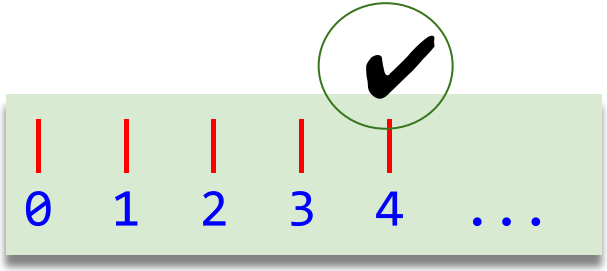


# Basic Operations with `lists`

- **Task** :
  - Create a `list` which consists of the `int` numbers (1 - 4) and then insert number **5** at the **end** of the `list` using `.insert()` method.

# Basic Operations with `lists`

- The code can be like :




```
numbers = [1, 2, 3, 4]
numbers.insert(4, 5)

print(numbers)
```

index

```
[1, 2, 3, 4, 5]
```

# Basic Operations with `lists`

- We can remove and sort the elements of the `list`.
- `.remove()` removes the elements from the `list`.

- `.remove()`
- `.sort()`

# Basic Operations with `lists`

- We can remove and sort the elements of the `list`.
- `.remove()` removes the elements from the `list`.

- `.remove()`
- `.sort()`

```
numbers = [1, 4, 7, 9]
numbers.remove(7)
print(numbers)
```

# Basic Operations with `lists`

- We can remove and sort the elements of the `list`.
- `.remove()` removes the elements from the `list`.

- `.remove()`
- `.sort()`

```
numbers = [1, 4, 7, 9]
numbers.remove(7)
print(numbers)
```

```
[1, 4, 9]
```

# Basic Operations with `lists`

- Consider this pre-class example 👇

```
1  city = ['New York', 'London', 'Stockholm', 'Istanbul', 'Seoul', 'Sydney', 'Addis Ababa']
2  city.remove('London')
3  print(city)  # we have deleted 'London'
4
```

# Basic Operations with `lists`

- Consider this pre-class example 👇

```
1  city = ['New York', 'London', 'Stockholm', 'Istanbul', 'Seoul', 'Sydney', 'Addis Ababa']
2  city.remove('London')
3  print(city)  # we have deleted 'London'
4
```

```
1  ['New York', 'Stockholm', 'Istanbul', 'Seoul', 'Sydney', 'Addis Ababa']
2
```

# Basic Operations with `lists`

- `.sort()` sorts the elements in the `list`.

- `.remove()`
- `.sort()`

# Basic Operations with `lists`

- `.sort()` sorts the elements in the `list`.

- `.remove()`
- `.sort()`

```
numbers = [4, 1, 9, 7]
numbers.sort()
print(numbers)
```

# Basic Operations with `lists`

- `.sort()` sorts the elements in the `list`.

- `.remove()`
- `.sort()`

```
numbers = [4, 1, 9, 7]
numbers.sort()
print(numbers)
```

```
[1, 4, 7, 9]
```

# Basic Operations with `lists`

- Here's the pre-class example 👇

```
1  city = ['New York', 'Stockholm', 'Istanbul', 'Seoul', 'Sydney', 'Addis Ababa']
2  city.sort()  # lists the items in alphabetical order
3  print(city)
4
```

# Basic Operations with `lists`

- Here's an example 👇

```
1  city = ['New York', 'Stockholm', 'Istanbul', 'Seoul', 'Sydney', 'Addis Ababa']
2  city.sort()  # lists the items in alphabetical order
3  print(city)
4
```

```
1  ['Addis Ababa', 'Istanbul', 'New York', 'Seoul', 'Stockholm', 'Sydney']
2
```

# Basic Operations with `lists`

- `.sort()` sorts the elements in the `list`.

```
mix_list = ['d', 1, 'a', 7]
mix_list.sort()
print(mix_list)
```

What is the output? Try to figure out in your mind...

# Basic Operations with `lists`

- `.sort()` sorts the elements in the `list`.

```
mix_list = ['d', 1, 'a', 7]
mix_list.sort()
print(mix_list)
```

```
TypeError                                 Traceback (most recent call
last)
<ipython-input-7-ad44188425eb> in <module>
      1 mix_list = ['d', 1, 'a', 7]
----> 2 mix_list.sort()
      3 print(mix_list)

TypeError: '<' not supported between instances of 'int' and 'str'
```

# Basic Operations with `lists`

- `.sort()` sorts the elements in the `list`.

```
mix_li
mix_li
print(
```

The items to be sorted in the list should be the same type.

```
TypeErro                                                    recent call
last)
<ipythor
      1 mix_list = [ d , 1,  a , 7]
----> 2 mix_list.sort()
      3 print(mix_list)

TypeError: '<' not supported between instances of 'int' and 'str'
```

# Basic Operations with `lists`

- We can also measure the length of the list by using `len()` function. Take a look at this pre-class example 👇

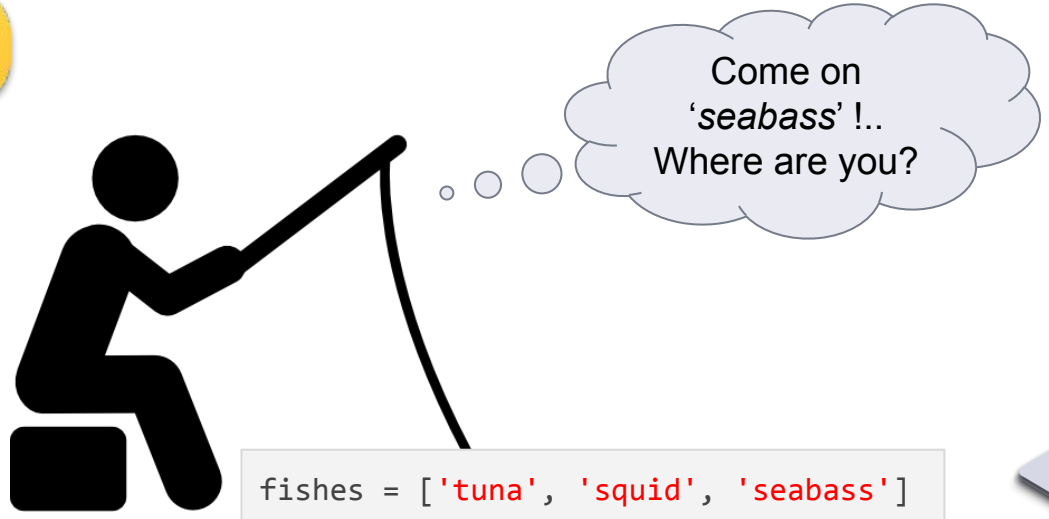
```
1  city = ['Addis Ababa', 'Istanbul', 'New York', 'Seoul', 'Stockholm', 'Sydney']
2  print(len(city))
3
```

# Basic Operations with `lists`

- We can also measure the length of the list by using `len()` function. Take a look at this pre-class example 👇

```
1  city = ['Addis Ababa', 'Istanbul', 'New York', 'Seoul', 'Stockholm', 'Sydney']
2  print(len(city))
3
```

```
1  6
2
```




```
fishes = ['tuna', 'squid', 'seabass']
```

# Accessing Lists

# Table of Contents

# Indexing a `list`

- The formula syntax 👇

```
list_name[index no]
```

# Indexing a `list`

- To access or use the elements of a **`list`**, we can use index numbers of the **`list`** enclosed by square brackets.

```
list_name[index no]
```

```
word = ['h', 'a', 'p', 'p', 'y']
print(word[1])
```

# Indexing a `list`

- To access or use the elements of a **`list`**, we can use index numbers of the **`list`** enclosed by square brackets.

```
list_name[index no]
```

```
word = ['h', 'a', 'p', 'p', 'y']
print(word[1])
```

```
a
```

# Indexing a `list` (review the pre-class)

- Here is the pre-class example of indexing a `list` :

```
1  colors = ['red', 'purple', 'blue', 'yellow', 'green']
2  print(colors[2])  # If we start at zero,
3  # the second element will be 'blue'.
4
```

# Indexing a `list` (review the pre-class)

- Here is an example of indexing a `list` :


```
1  colors = ['red', 'purple', 'blue', 'yellow', 'green']
2  print(colors[2])  # If we start at zero,
3  # the second element will be 'blue'.
4
```

```
1  blue
2
```

# Indexing a `list` (review the pre-class)

- Here is another pre-class example of indexing a nested `list`.

```
1  city = ['New York', 'London', 'Istanbul', 'Seoul', 'Sydney']
2
3  city_list = []
4  city_list.append(city) # we have created a nested list
5
6  print(city_list)
7
```

# Indexing a `list` (review the pre-class)

- Here is another pre-class example of indexing a nested `list`.

```
1  city = ['New York', 'London', 'Istanbul', 'Seoul', 'Sydney']
2
3  city_list = []
4  city_list.append(city) # we have created a nested list
5
6  print(city_list)
7
```

```
1  [['New York', 'London', 'Istanbul', 'Seoul', 'Sydney']]
2
```

How many items do the `city` list have?

# Indexing a `list` (review the pre-class)

- Here is another example of indexing a nested `list`.

```
1  city = ['New York', 'London', 'Istanbul', 'Seoul', 'Sydney']
2
3  city_list = []
4  city_list.append(city) # we have created a nested list
5
6  print(city_list)
7
```

```
1  [['New York', 'London', 'Istanbul', 'Seoul', 'Sydney']]
2
```

⚠ `city_list` has only one item.

**Tips :**

- If you notice that `city_list` has double square brackets.

# Indexing a `list`

- Let's access `city_list`'s first and the only element.

```
1  city_list = [['New York', 'London', 'Istanbul', 'Seoul', 'Sydney']]
2  print(city_list[0]) # access to first and only element
3
```

# Indexing a `list`

- Let's access `city_list`'s first and the only element.

```
1  city_list = [['New York', 'London', 'Istanbul', 'Seoul', 'Sydney']]
2  print(city_list[0]) # access to first and only element
3
```

```
1  ['New York', 'London', 'Istanbul', 'Seoul', 'Sydney']
2
```

# Indexing a `list`

- Let's access `city_list`'s first and the only element.

```
1  city_list = [['New York', 'London', 'Istanbul', 'Seoul', 'Sydney']]
2  print(city_list[0]) # access to first and only element
3
```

```
1  ['New York', 'London', 'Istanbul', 'Seoul', 'Sydney']
2
```

- The output of the syntax `city_list[0]` is a `list` type. So we can still access its elements.

```
1  city_list = [['New York', 'London', 'Istanbul', 'Seoul', 'Sydney']]
2  print(city_list[0][2])
3
```

⚠ it is also a `list`

What is the output? Try to figure out in your mind...

# Indexing a `list`

- Let's access `city_list`'s first and the only element.

```
city_list = [['New York', 'London', 'Istanbul', 'Seoul', 'Sydney']]
print(city_list[0]) # access to first and only element

```

```
['New York', 'London', 'Istanbul', 'Seoul', 'Sydney']

```

- The output of the syntax `city_list[0]` is a `list` type. So we can still access its elements.

```
city_list = [['New York', 'London', 'Istanbul', 'Seoul', 'Sydney']]
print(city_list[0][2])

```

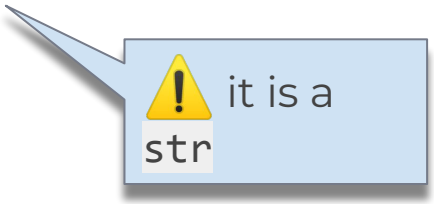
```
Istanbul

```

# Indexing a `list`

- The output of the syntax `city_list[0][2]` is a `str` type. So we can still access its elements.

```
1  city_list = [['New York', 'London', 'Istanbul', 'Seoul', 'Sydney']]
2  print(city_list[0][2][3])
3
```



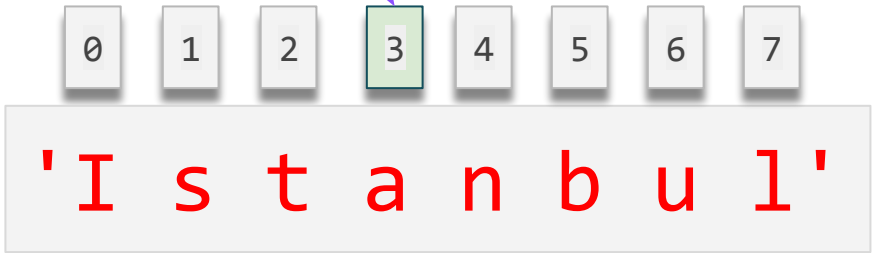

What is the output? Try to figure out in your mind...

# Indexing a `list`

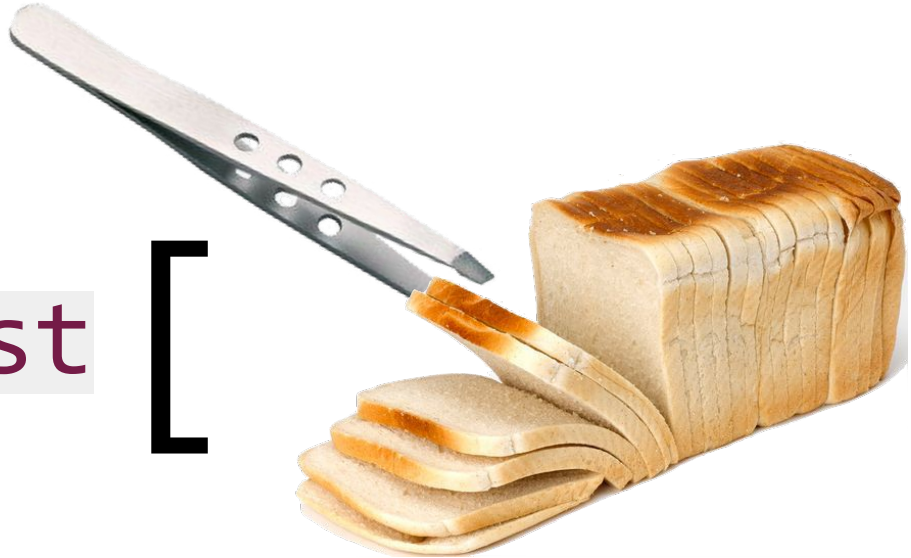
- The output of the syntax `city_list[0][2]` is a `str` type. So we can still access its elements.

```
1  city_list = [['New York', 'London', 'Istanbul', 'Seoul', 'Sydney']]
2  print(city_list[0][2][3])
3
```

```
1  a
2
```



# Slicing a `list`



# Slicing a `list`

- The formula syntax 👇

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

From '`start`' to '`stop-1`', by '`step`'.

# Slicing a `list`

- Consider this simple example :

```
1  even_numbers = [2, 4, 6, 8, 10, 12, 14]
2  print(even_numbers[2:5])
3
4
```

What is the output? Try to figure out in your mind...

# Slicing a `list`

- Consider this simple example :

```
1  even_numbers = [2, 4, 6, 8, 10, 12, 14]
2  print(even_numbers[2:5])
3
4
```

Output

```
[6, 8, 10]
```

⚠ The type of the output is also a `list`.

# range() function

- Returns a list of arithmetic progressions.
  - As we stated before, the formula syntax of the `range()` function is :

```
range(start, stop, step)
```

parameters

```
>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list(range(1, 11))
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> list(range(0, 30, 5))
[0, 5, 10, 15, 20, 25]
```

# Slicing a `list`

- Task :
  - Create a `list` of numbers from **1** to **10** using `range()` function and select **odd** ones by **slicing** method and then print the result.

Review the function

- `range(start,stop,step)` function returns an object that produces a sequence of integers from `start` (including) to `stop` (excluding) by `step`.

# Slicing a `list`

- The code can be like :

```
1  odd_numbers = list(range(11))
2
3  print(odd_numbers)
4  print(odd_numbers[1:11:2])
5
6
7
```

Output

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[1, 3, 5, 7, 9]
```

# Slicing a `list`

- Usage options of slicing are as follows :

- `my_list[:]`: returns the full copy of the sequence
- `my_list[start:]` : returns elements from `start` to the end element
- `my_list[:stop]` : returns element from the 1st element to `stop-1`
- `my_list[::step]` : returns each element with a given `step`

# Slicing a `list` (review the pre-class)

- Consider this pre-class example :

```
1  animals = ['elephant', 'bear', 'fox', 'wolf', 'rabbit', 'deer', 'giraffe']
2
3  print(animals[:])  # all elements of the list
4
```

# Slicing a `list` (review the pre-class)

- Consider this simple example :

```
1  animals = ['elephant', 'bear', 'fox', 'wolf', 'rabbit', 'deer', 'giraffe']
2
3  print(animals[:])  # all elements of the list
4
```

```
1  ['elephant', 'bear', 'fox', 'wolf', 'rabbit', 'deer', 'giraffe']
2
```

`print(animals) = print(animals[:])`

These syntaxes give the same output.

# Slicing a `list` (review the pre-class)

- Slicing options :

```
1  animals = ['elephant', 'bear', 'fox', 'wolf', 'rabbit', 'deer', 'giraffe']
2  print(animals[3:])
3
```

# Slicing a `list` (review the pre-class)

- The following example slices the `animals` starts at index=3 to the end.

```
1  animals = ['elephant', 'bear', 'fox', 'wolf', 'rabbit', 'deer', 'giraffe']
2  print(animals[3:])
3
```

```
1  ['wolf', 'rabbit', 'deer', 'giraffe']
2
```

# Slicing a `list` (review the pre-class)

- Slicing options :

```
1  animals = ['elephant', 'bear', 'fox', 'wolf', 'rabbit', 'deer', 'giraffe']
2  print(animals[:5])
3
```

# Slicing a `list` (review the pre-class)

- The following example slices the `animals` starts at index=0 to the index=4.

```
1  animals = ['elephant', 'bear', 'fox', 'wolf', 'rabbit', 'deer', 'giraffe']
2  print(animals[:5])
3
```

```
1  ['elephant', 'bear', 'fox', 'wolf', 'rabbit']
2
```

# Slicing a `list` (review the pre-class)

- Slicing options :

```
1  animals = ['elephant', 'bear', 'fox', 'wolf', 'rabbit', 'deer', 'giraffe']
2  print(animals[::2])
3
```

# Slicing a `list` (review the pre-class)

- This example slices animals starts at index=0 to the end with 2 step.

```
1  animals = ['elephant', 'bear', 'fox', 'wolf', 'rabbit', 'deer', 'giraffe']
2  print(animals[::2])
3
```

```
1  ['elephant', 'fox', 'rabbit', 'giraffe']
2
```

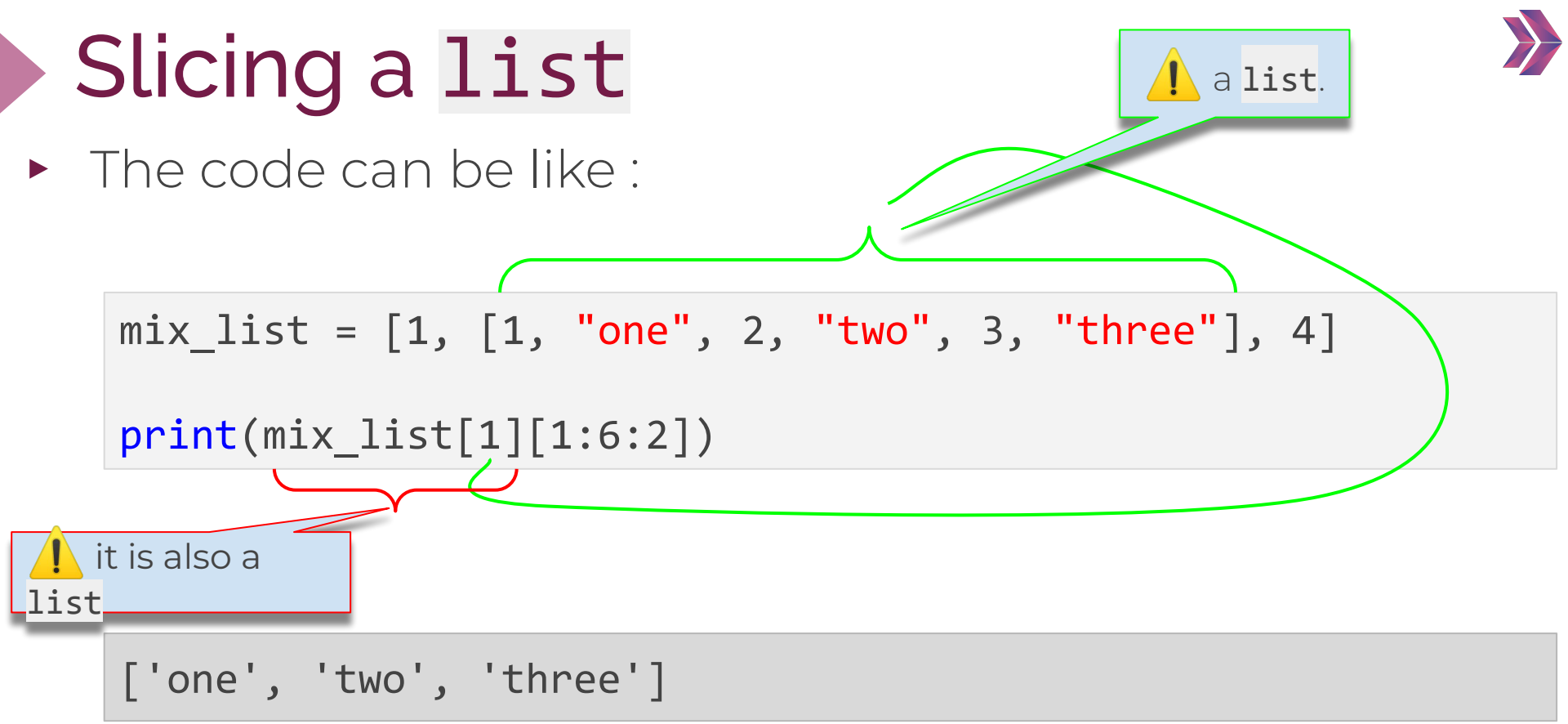
# Slicing a `list`

- Task :
  - Select and print the **string typed numbers** from the following `list` using *indexing* and *slicing* methods.
  - Your code must consist of a **single line**.

```
mix_list = [1, [1, "one", 2, "two", 3, "three"], 4]
```

# Slicing a `list`

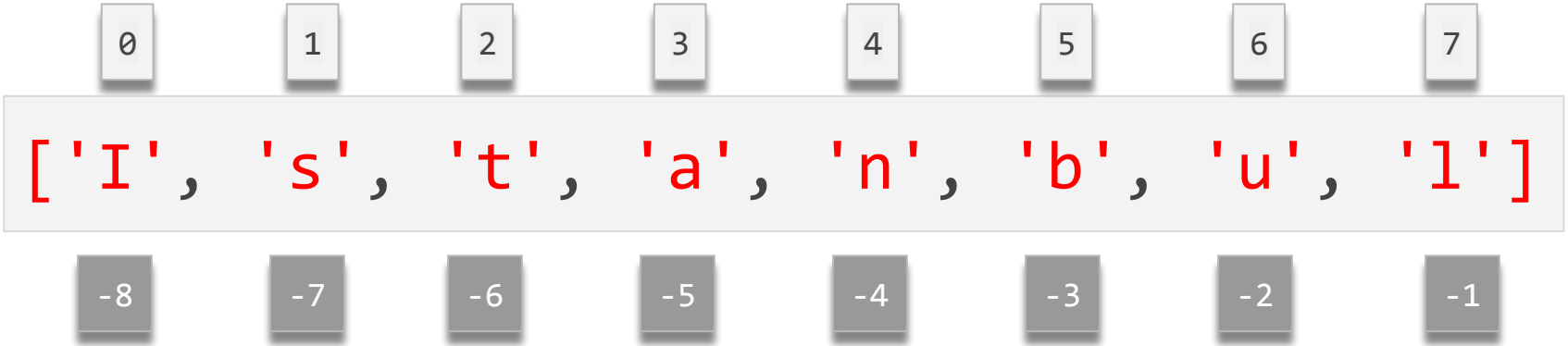
- The code can be like :



# Negative Indexing & Slicing

# Negative Indexing & Slicing

- Sample of the negative indices sequence



# Negative Indexing & Slicing(review)

- Take a look at this pre-class example of negative indexing.

```
1  city = ['New York', 'London', 'Istanbul', 'Seoul', 'Sydney']
2  print(city[-4])
3
```

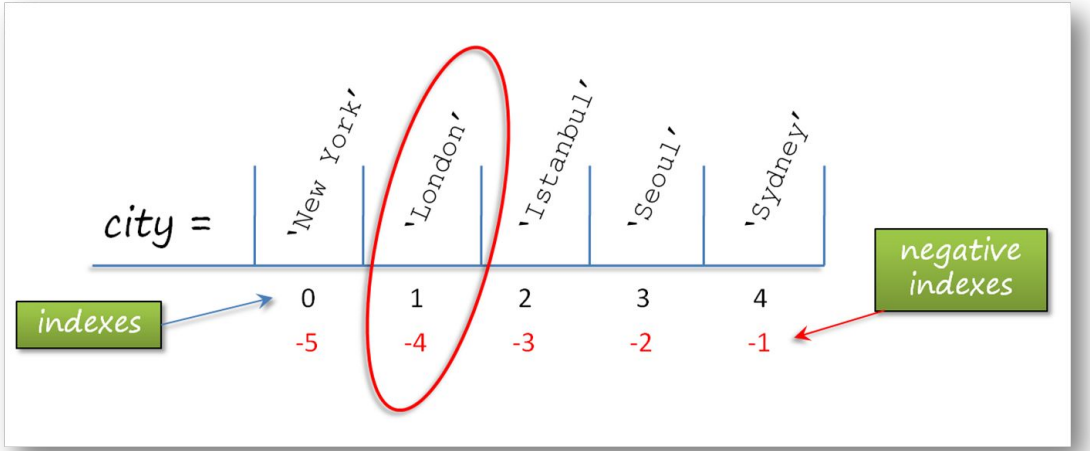
What is the output? Try to figure out in your mind...

# Negative Indexing & Slicing(review)

- The output is.

```
1  city = ['New York', 'London', 'Istanbul', 'Seoul', 'Sydney']
2  print(city[-4])
3
```

```
1  London
2
```



# Negative Indexing & Slicing(review)

- Now, let's consider this pre-class example of negative slicing.

```
1  reef = ['swordfish', 'shark', 'whale', 'jellyfish', 'lobster', 'squid', 'octopus']
2  print(reef[-3:])
3
```

What is the output? Try to figure out in your mind...

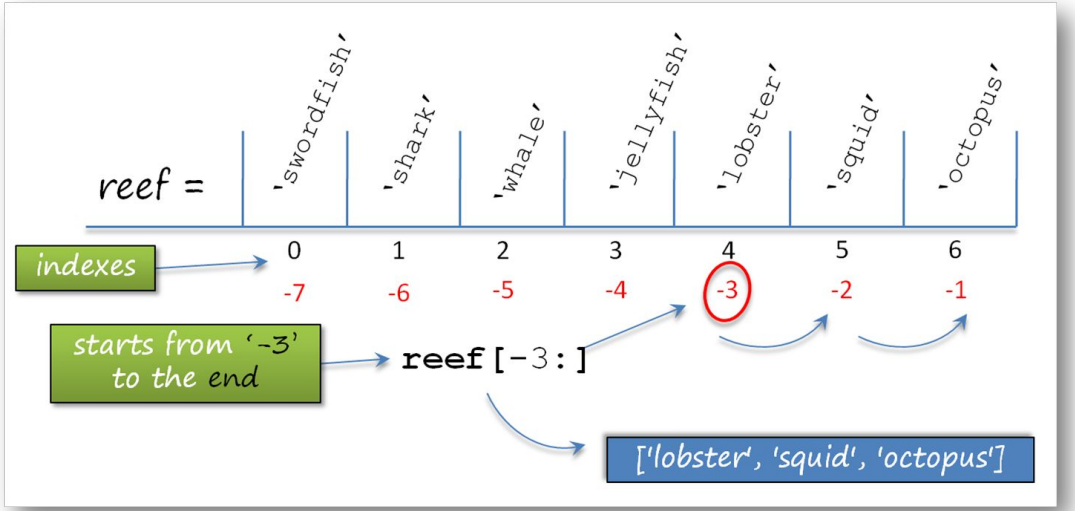
Students, write your response!

# Negative Indexing & Slicing(review)

- The output is :

```
1  reef = ['swordfish', 'shark', 'whale', 'jellyfish', 'lobster', 'squid', 'octopus']
2  print(reef[-3:])
3
```

```
1  ['lobster', 'squid', 'octopus']
2
```



# Negative Indexing & Slicing(review)

- Here's another pre-class example of negative slicing.

```
1  reef = ['swordfish', 'shark', 'whale', 'jellyfish', 'lobster', 'squid', 'octopus']
2  print(reef[:-3])
3
```

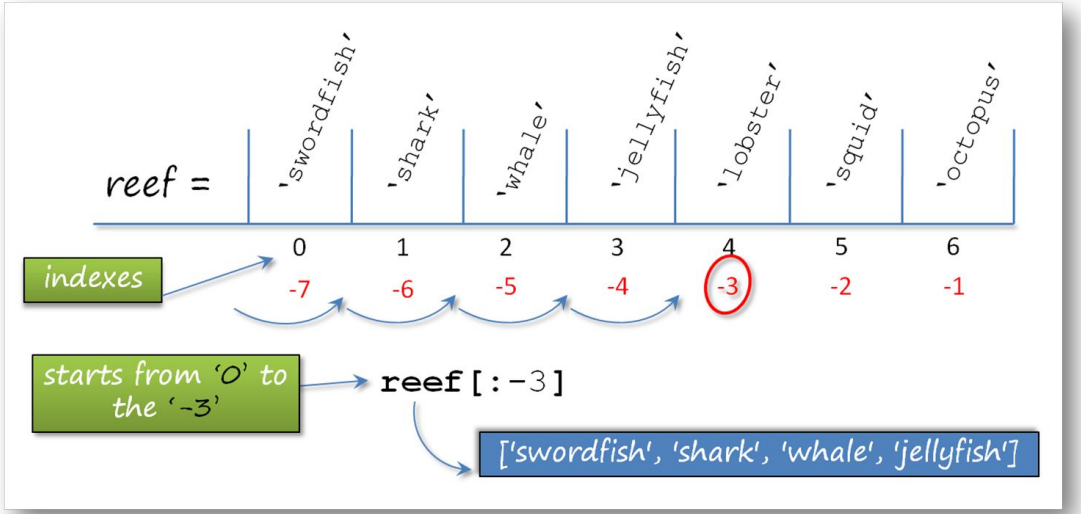
What is the output? Try to figure out in your mind...

# Negative Indexing & Slicing(review)

- The output is :

```
1  reef = ['swordfish', 'shark', 'whale', 'jellyfish', 'lobster', 'squid', 'octopus']
2  print(reef[:-3])
3
```

```
1  ['swordfish', 'shark', 'whale', 'jellyfish']
2
```



# Negative Indexing & Slicing(review)

- Let's see negative **step**ping in the pre-class content :

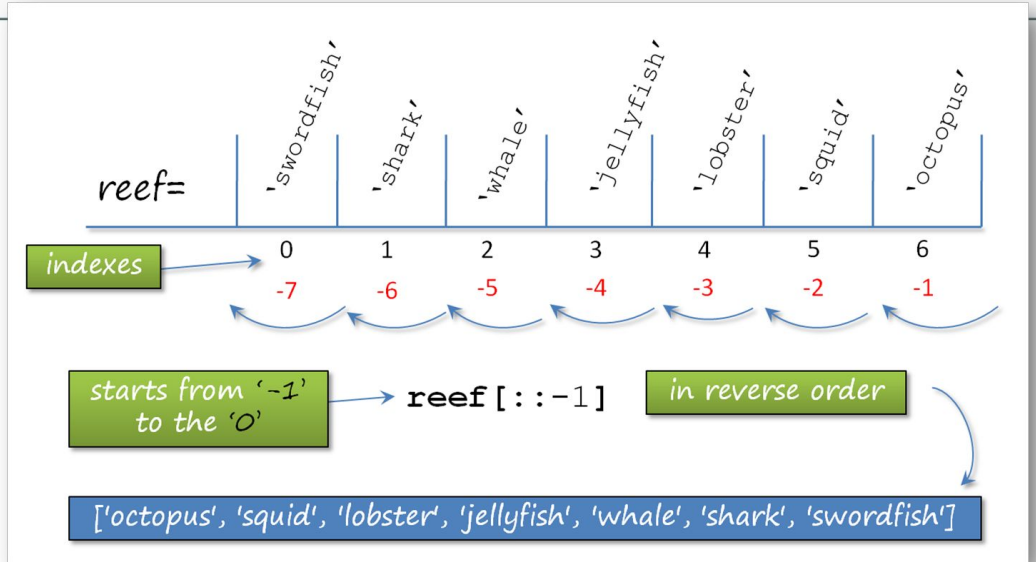
```
1  reef = ['swordfish', 'shark', 'whale', 'jellyfish', 'lobster', 'squid', 'octopus']
2  print(reef[::-1])  # we have produced the reverse of the list
3
```

# Negative Indexing & Slicing(review)

- The output is :

```
1  reef = ['swordfish', 'shark', 'whale', 'jellyfish', 'lobster', 'squid', 'octopus']
2  print(reef[::-1])  # we have produced the reverse of the list
3
```

```
1  ['octopus', 'squid', 'lobster', 'jellyfish', 'whale', 'shark', 'swordfish']
2
```



# Negative Indexing & Slicing(review)

- Here's another example of negative `step`ping.

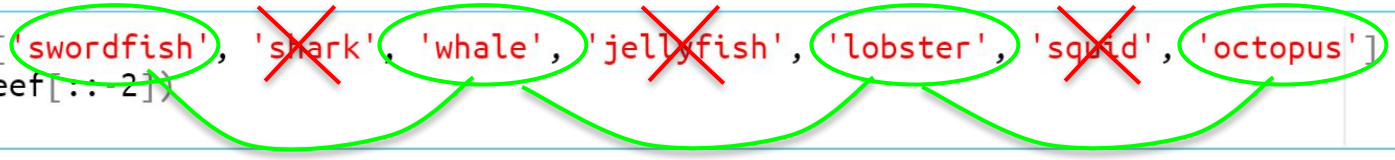
```
1  reef = ['swordfish', 'shark', 'whale', 'jellyfish', 'lobster', 'squid', 'octopus']
2  print(reef[::-2])
3
```

What is the output? Try to figure out in your mind...

# Negative Indexing & Slicing(review)

- Here's another example of negative **step**ping.

```
reef = ['swordfish', 'shark', 'whale', 'jellyfish', 'lobster', 'squid', 'octopus']
print(reef[::-2])
```



```
['octopus', 'lobster', 'whale', 'swordfish']
```

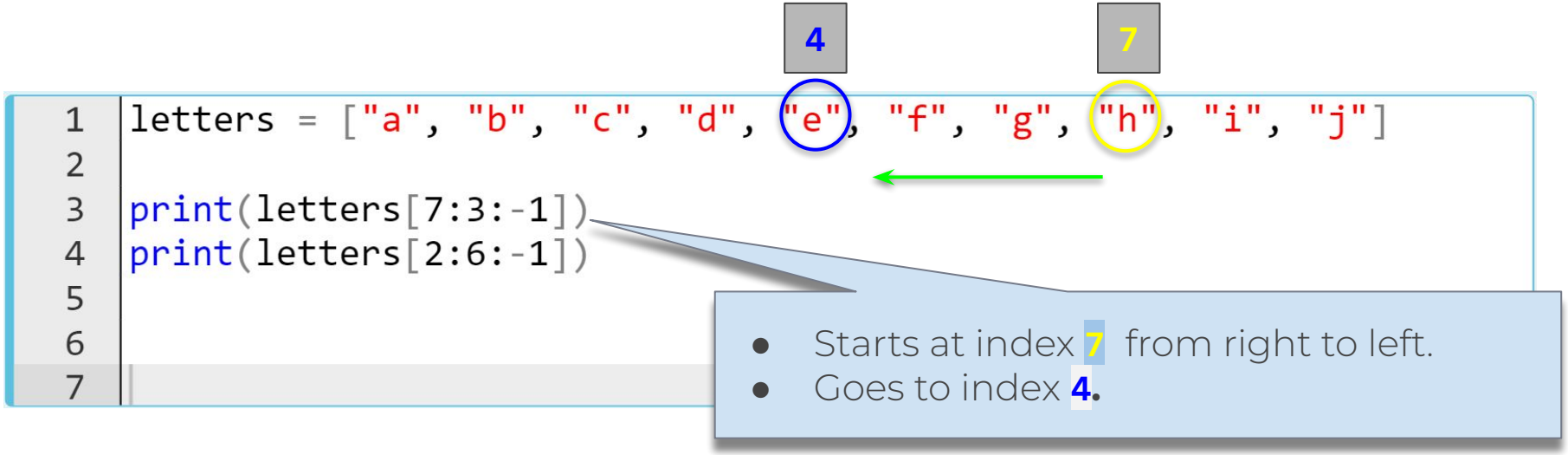
# Negative Indexing & Slicing

**Tips :**

- If you choose negative step with the start and end indexes together, those should be used accordingly, that is, the `end` index should be less than the `start` index.

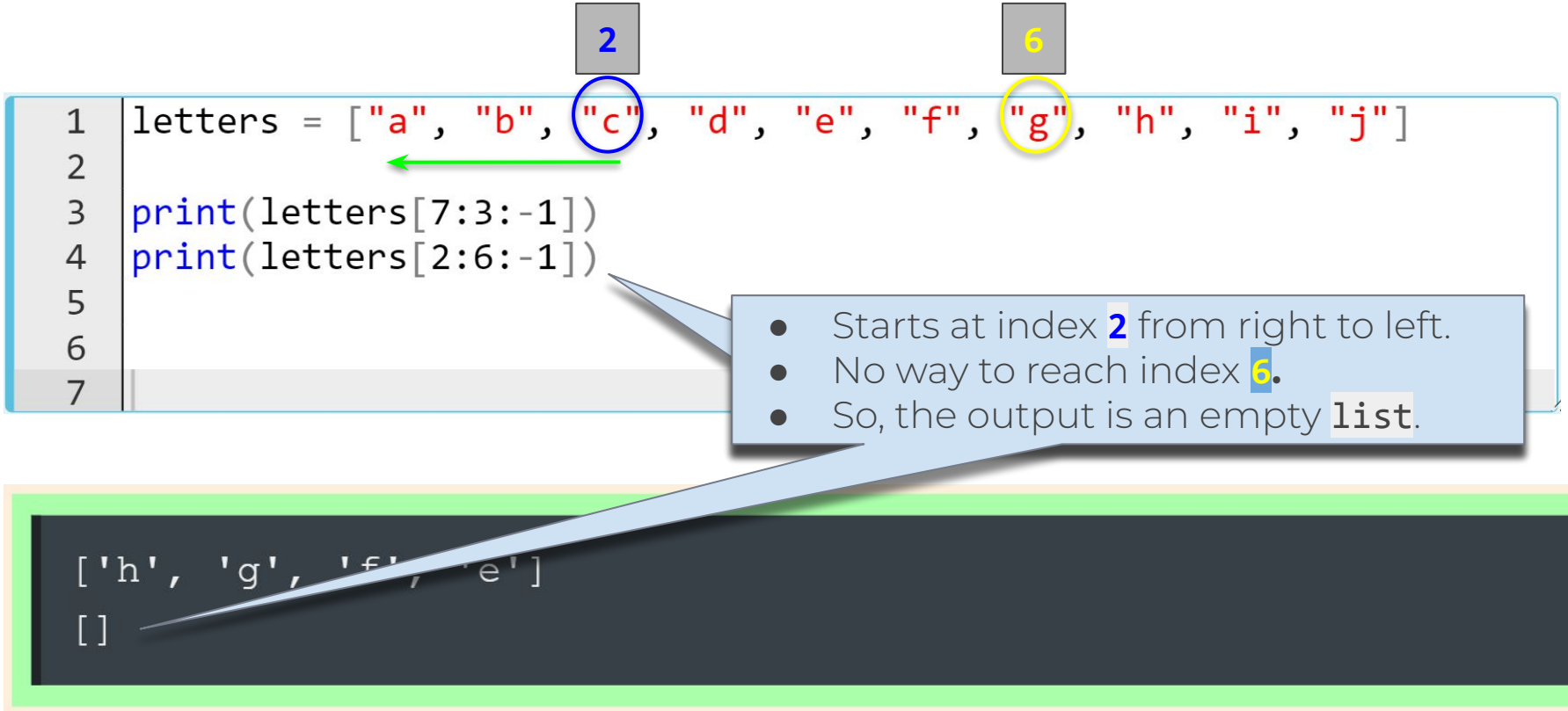
```
1  letters = ["a", "b", "c", "d", "e", "f", "g", "h", "i", "j"]
2
3  print(letters[7:3:-1])
4  print(letters[2:6:-1])
5
6
7
```

# Negative Indexing & Slicing




```
['h', 'g', 'f', 'e']
[]
```

# Negative Indexing & Slicing



# THANKS!

## End of the Lesson

(Lists)