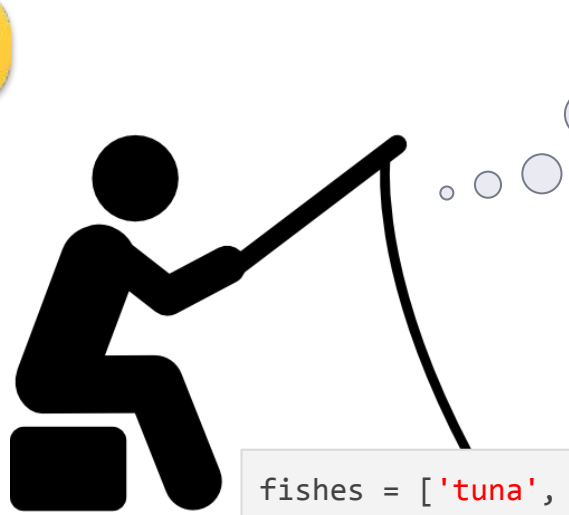



```
fishes = ['tuna', 'squid', 'seabass']
```

Accessing Lists

Table of Contents

```
fruit[1]
```

```
fruit = ['Apple', 'Orange', 'Banana']
```

Indexing a `list`

```
list()
```

How was your Python pre-class studyings?

Too hard

Just right

Too easy

Students, drag the icon!

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` :

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

```

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?

Students, write your response!

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
```

⚠ `city_list` has only one item.

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

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...

Students choose an option

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
```

```
1  Istanbul
2
```

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
```

⚠ it is a `str`

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

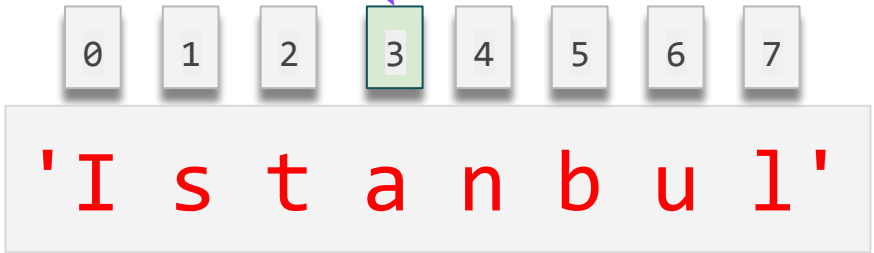
Students, write your response!

Indexing a `list`

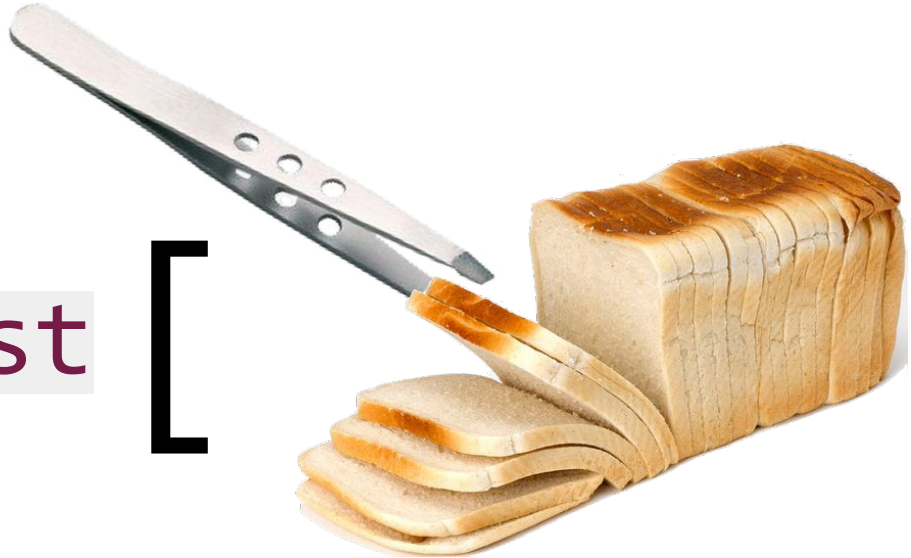
- The output of the syntax `city_list[0][2]` is a `str` type. So we can still access its elements.

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

```
1  a
2
```



Slicing a `list`



What do you understand from slicing a `list`?

Type your understanding from pre-class content.

EXCAVATE AND EXPAND

Students, write your response!

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...

Students, write your response!

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
```

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

Output

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

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
```

Tips :

- Slicing is just similar to indexing. The difference is adding **colon** or **colons** in square brackets.

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

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`

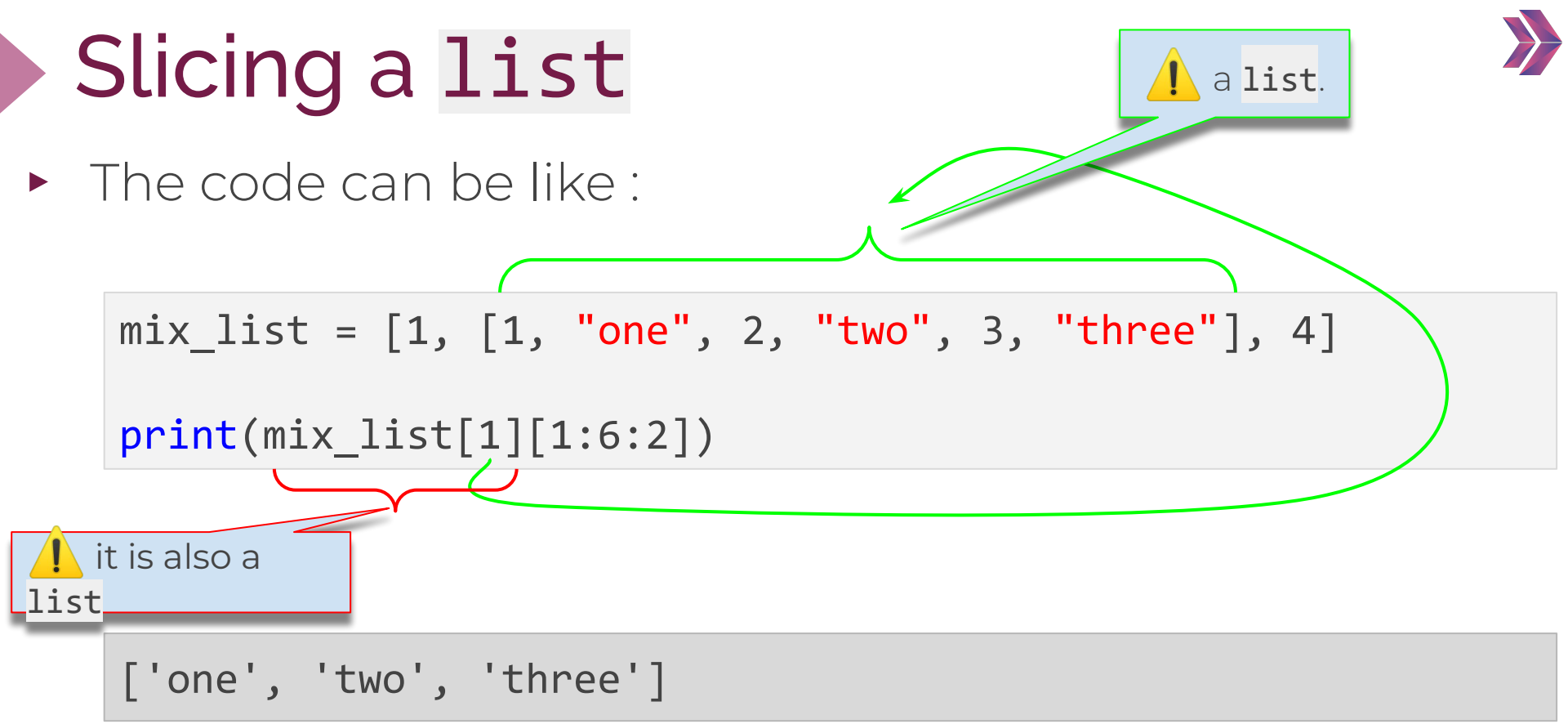
- Task :
  - Select and print the **string typed numbers** from the following `list` using *indexing* and *slicing* methods.
  - Your code must consist of a **single line**.

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

Students, write your response!

Slicing a `list`

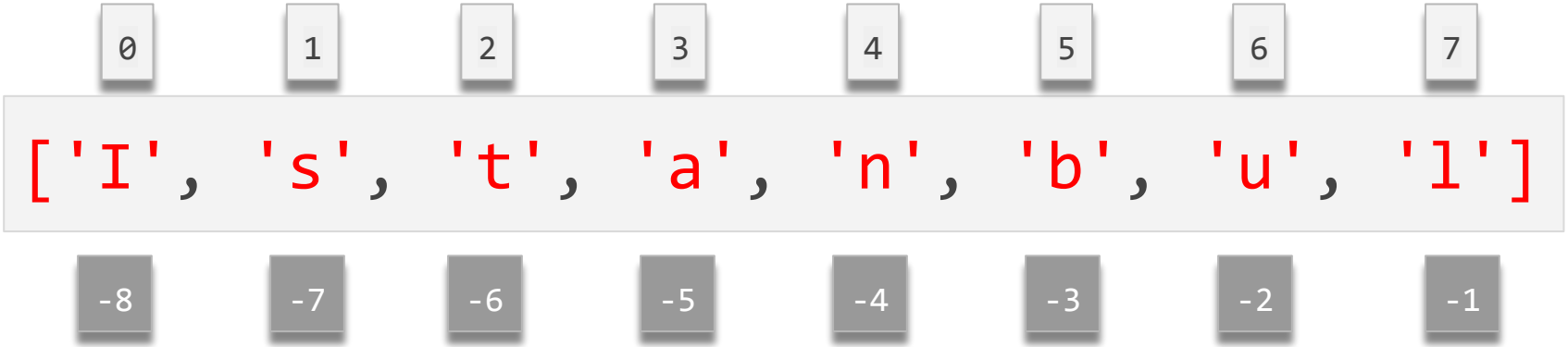
- The code can be like :



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
```

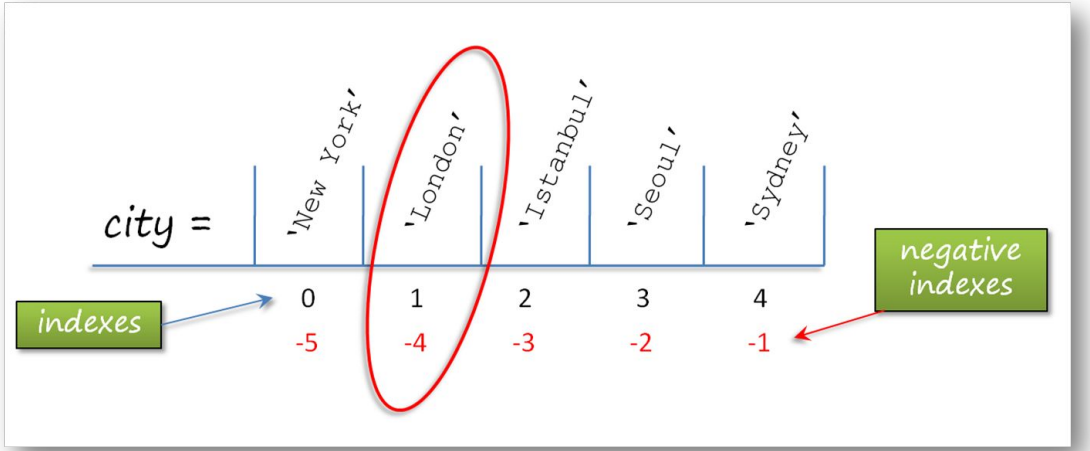
What is the output? Try to figure out in your mind...

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...

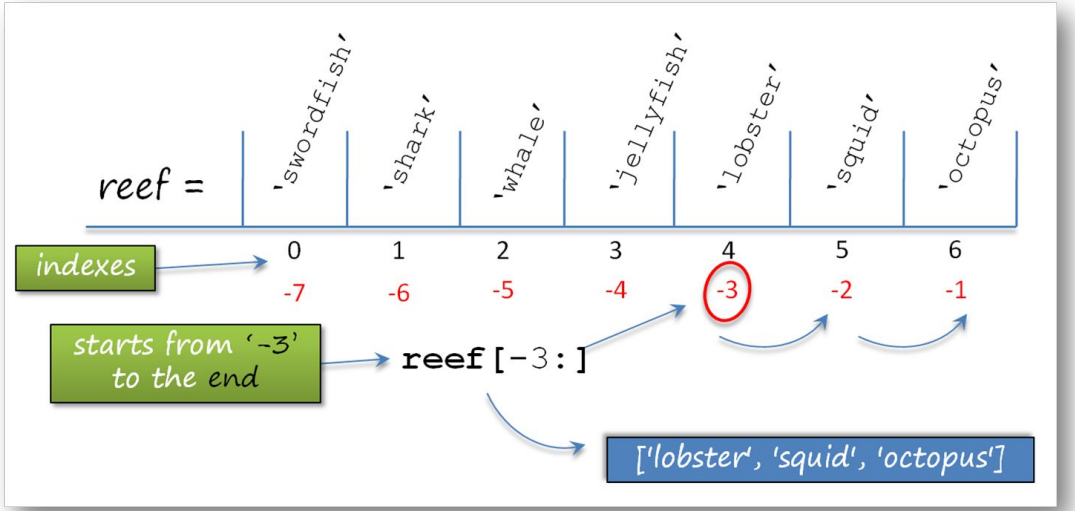
Students, write your response!

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...

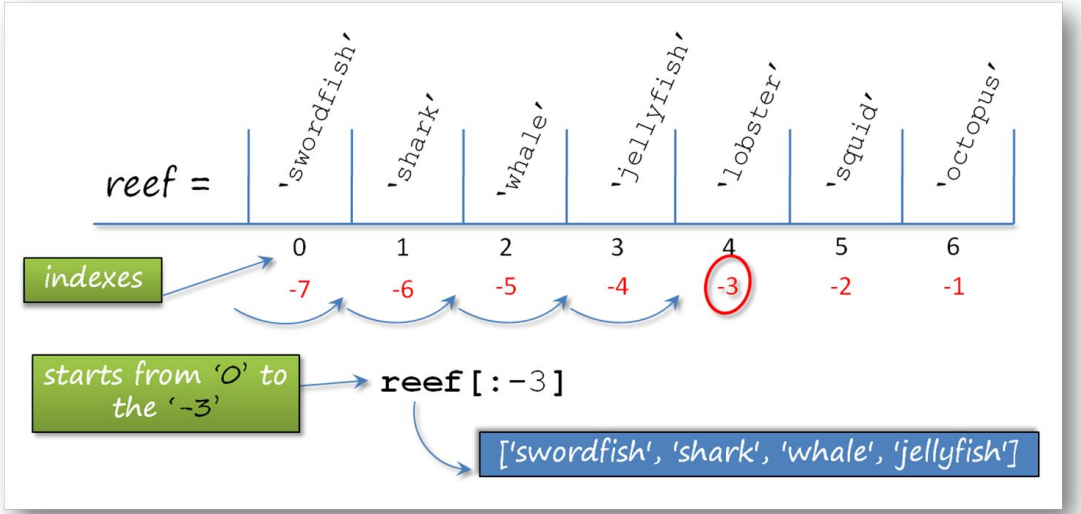
Students, write your response!

Negative Indexing & Slicing(review)

- The output is :

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

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



Negative Indexing & Slicing(review)

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

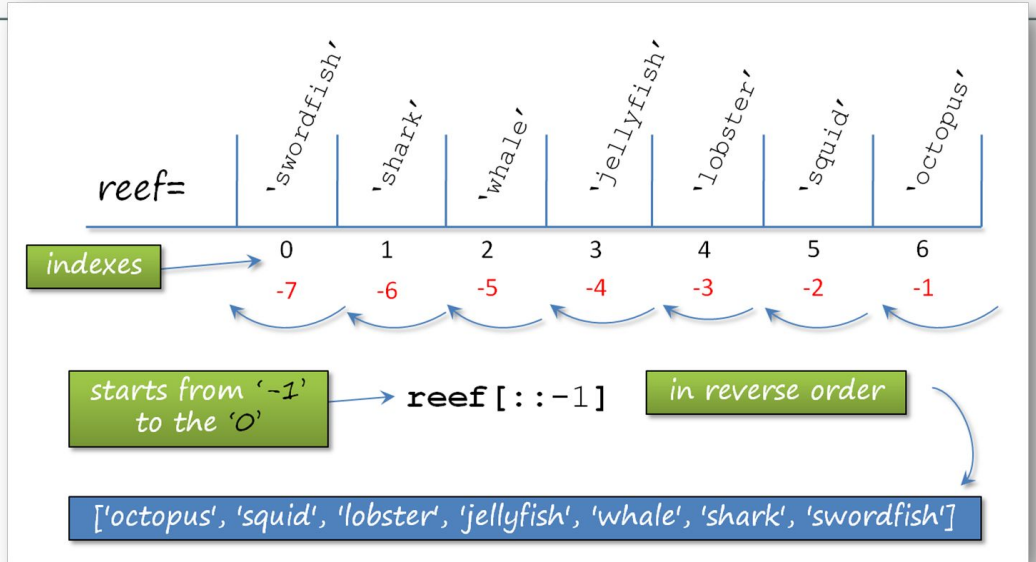
```
1  reef = ['swordfish', 'shark', 'whale', 'jellyfish', 'lobster', 'squid', 'octopus']
2  print(reef[::-1])  # we have produced the reverse of the list
3
```

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
```

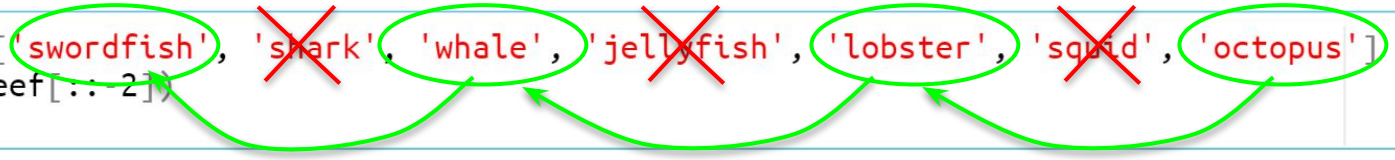
What is the output? Try to figure out in your mind...

Students, write your response!

Negative Indexing & Slicing(review)

- Here's another example of negative **stepping**.

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



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

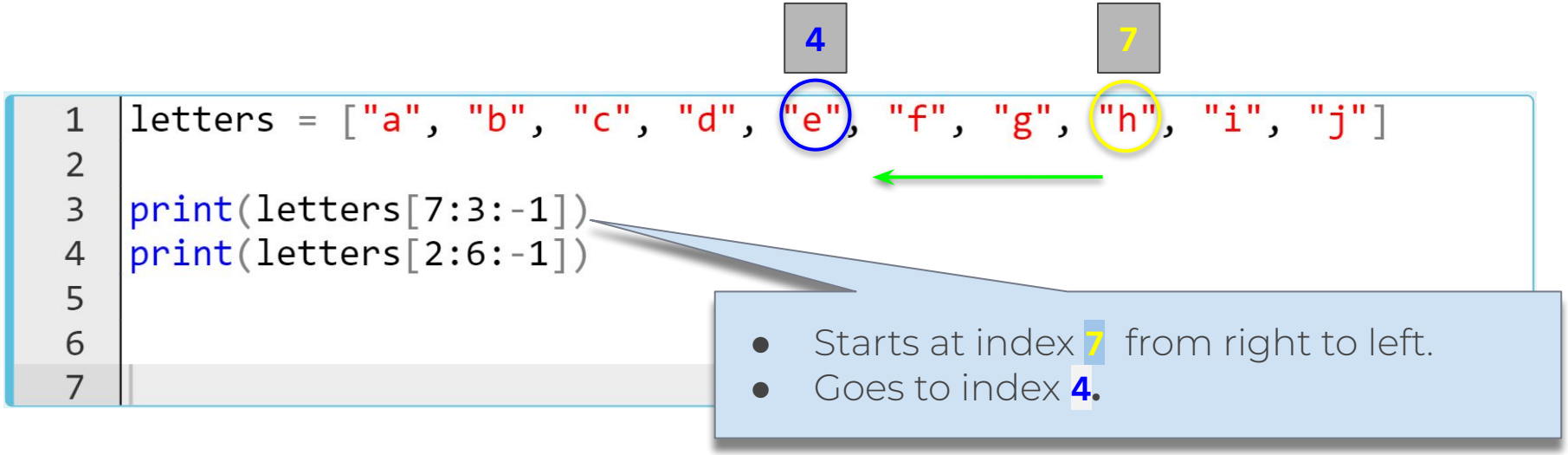
Negative Indexing & Slicing

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


```
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
```

- Starts at index **2** from right to left.
- No way to reach index **6**.
- So, the output is an empty `list`.

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