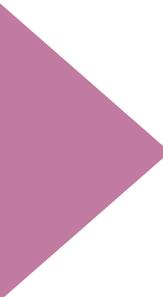




Basic Operations with Lists



Basic Operations with lists



- ▶ Creating an empty list is a phenomenon in Python.

How to create an empty list?



Students choose an option

REINVENT YOURSELF

Pear Deck Interactive Slide
Do not remove this bar

Basic Operations with lists



- ▶ Two methods for creating an empty list.

```
empty_list_1 = []
```

```
empty_list_2 = list()
```

Basic Operations with lists



- ▶ `.append()` appends an object to the end of a **list**.

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

Basic Operations with lists



- ▶ `.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('9')
print(numbers)
```

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

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...



DUWAY[®]
REINVENT YOURSELF

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]]
```

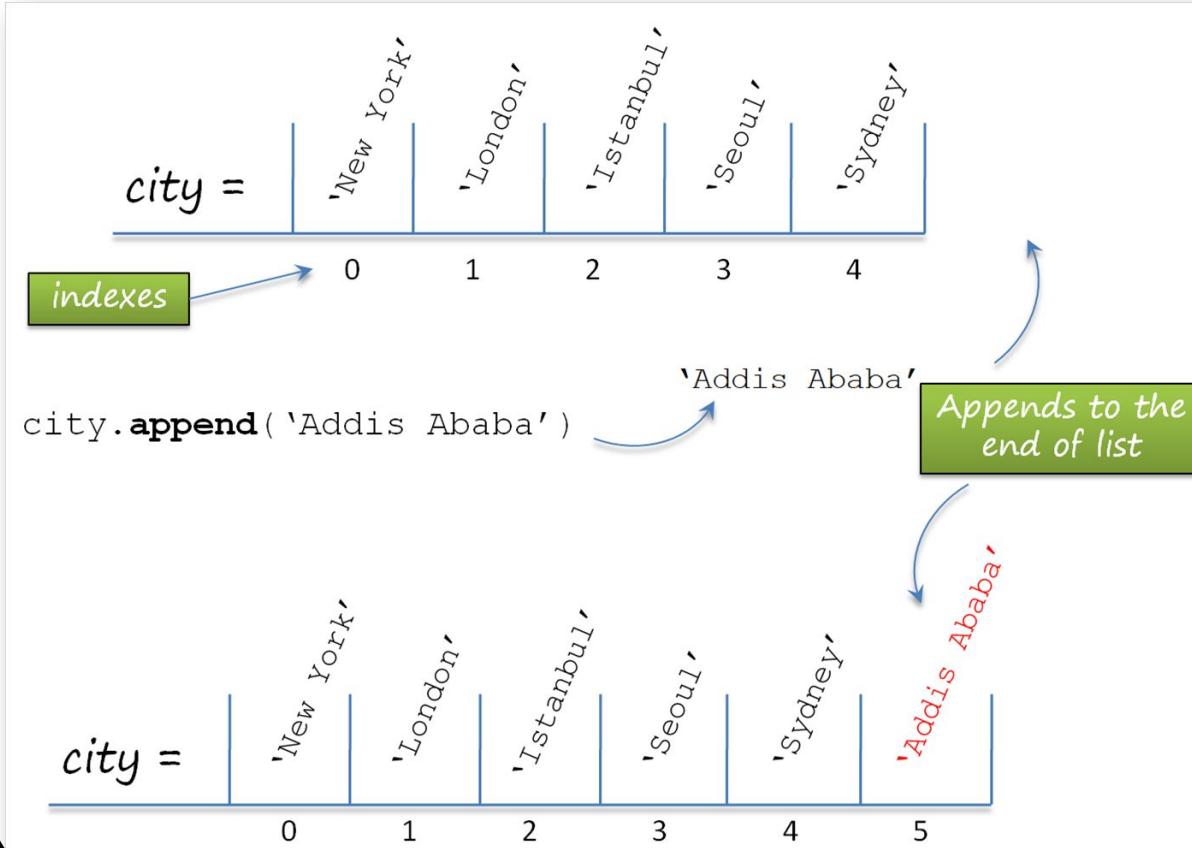
Basic Operations with lists

- ▶ 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...



USWY

REINVENT YOURSELF

Students, write your response!

Basic Operations with lists



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

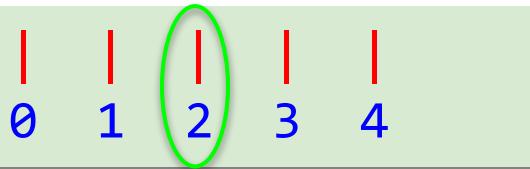
- `.append()`
- `.insert()`



```
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
```

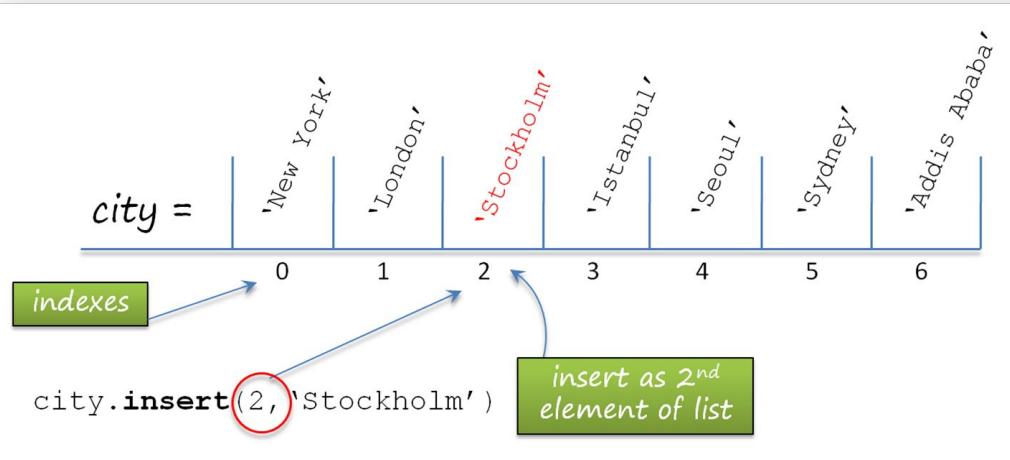
Basic Operations with lists

- ▶ 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

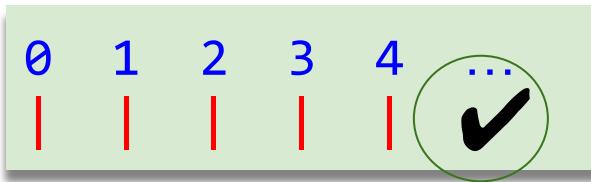


- ▶ Task :
 - ▷ Create a **list** which consists of the **int** numbers (1, 2, 3, 4) and then insert number **5** at the **end** of the **list** using **.insert()** method.

Basic Operations with lists



- The code can be like :



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

print(numbers)
```

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

Basic Operations with lists



- ▶ `.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)
```



USWY[®]
Students, write your response!

REINVENT YOURSELF

Pear Deck Interactive Slide
Do not remove this bar



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...



Students, write your response!

REINVENT YOURSELF

Basic Operations with lists



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

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

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

Traceback (most recent call

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()  
  
TypeError:  
last)  
<ipython  
1 mix_list = [ 0 , 1 ,  a , 7 ]  
----> 2 mix_list.sort()  
3 print(mix_list)
```

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

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

Basic Operations with lists



- ▶ Sort these elements in the `lists`. Do not play with Playground/IDEs, push your brains!..

```
list_1 = ['one', 'four', 'nine']
list_2 = ['@', '*-', 'False']
list_3 = [True, False]
list_4 = [[3], [44], [-12]]
list_5 = [[1, 3], [44, -40], [-12, 1]]
```

Basic Operations with lists



- Now! You can Play with Playgrounds. 😊

```
list_1 = ['one', 'four', 'nine']
list_2 = ['@', '*-', 'False']
list_3 = [True, False]
list_4 = [[3], [44], [-12]]
list_5 = [[1, 3], [44, -40], [-12, 1]]
```

ord()

gives the ASCII codes for any char

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
```

Basic Operations with lists



- ▶ Calculate the **length** of these **lists**. Do not play with Playground/IDEs, push your brains!..

```
list_1 = ['one', 'four', 'nine']
list_2 = ['@', '*-', 'False']
list_3 = [True, False]
list_4 = [[3], [44], [-12]]
list_5 = [[1, 3], [44, -40], [-12, 1]]
```

Basic Operations with lists



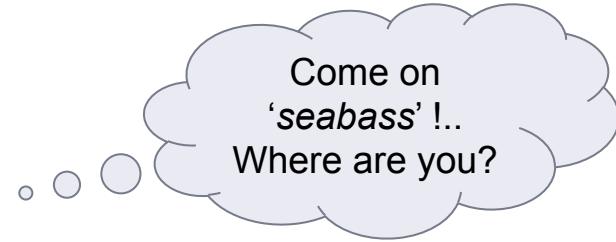
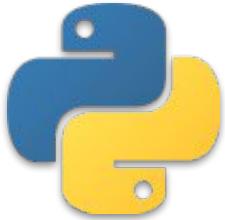
- Now! You can Play with Playgrounds. 😊

```
list_1 = ['one', 'four', 'nine']
list_2 = ['@', '*-', 'False']
list_3 = [True, False]
list_4 = [[3], [44], [-12]]
list_5 = [[1, 3], [44, -40], [-12, 1]]
```

Circle how you are feeling:



Students, draw anywhere on this slide!



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



Accessing Lists



Table of Contents

- ▶ Indexing a List
- ▶ Slicing a List
- ▶ Negative Indexing & Slicing



fruit[1]

Indexing a list

list()

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

Did you find this lesson (from pre-class) interesting or challenging?



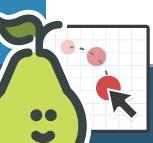
Too hard



Just right



Too easy



Students, drag the icon!



Pear Deck Interactive Slide
Do not remove this bar

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?



USWY[®]
Students, write your response!
REINVENT YOURSELF

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...



CSWAY®
Students choose an option

REINVENT YOURSELF



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...



USWY®
REINVENT YOURSELF



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
```

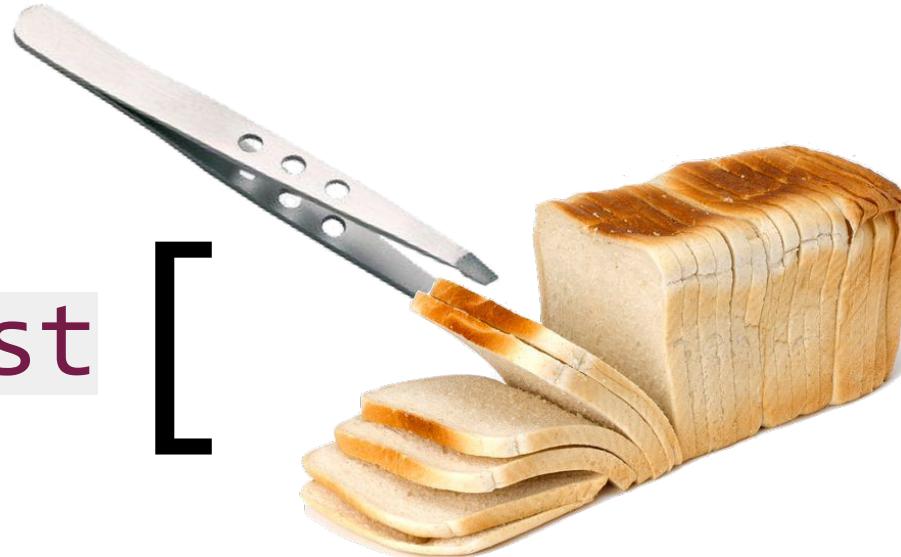


'I s t a n b u l'



Slicing a list

[



]



What do you understand from slicing a list?

Type your understanding from pre-class content.



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...



USWY[®]
Students, write your response!
REINVENT YOURSELF

Pear Deck Interactive Slide
Do not remove this bar



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**.

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]
```

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.



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 :

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

```
print(mix_list[1][1:6:2])
```

! it is also a
list

```
['one', 'two', 'three']
```



a list.



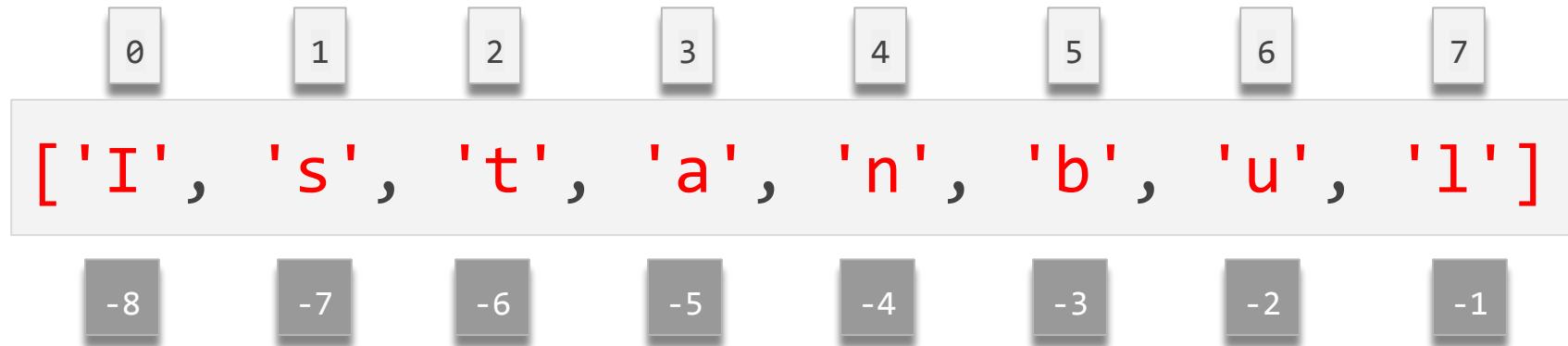
Negative Indexing & Slicing





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...

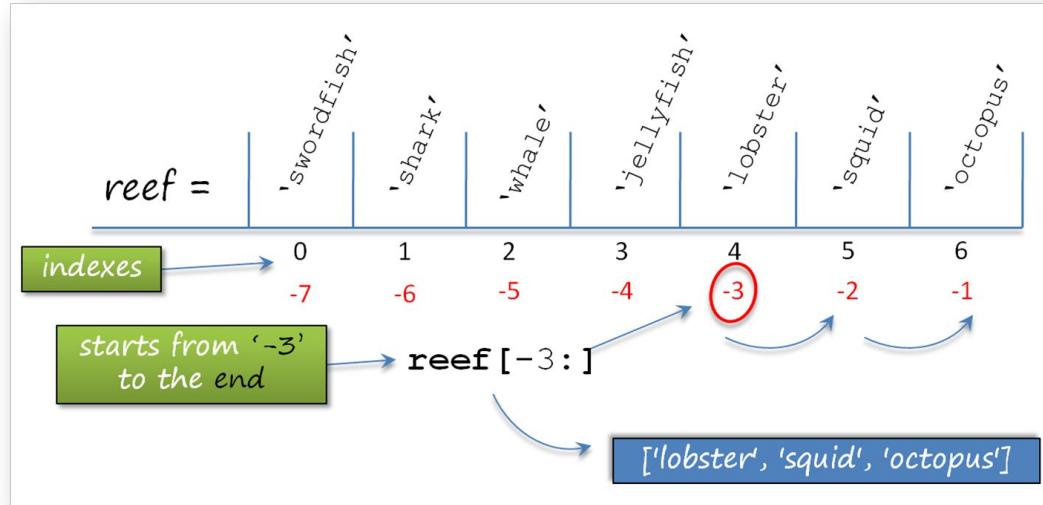


Negative Indexing & Slicing(review)

- 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...

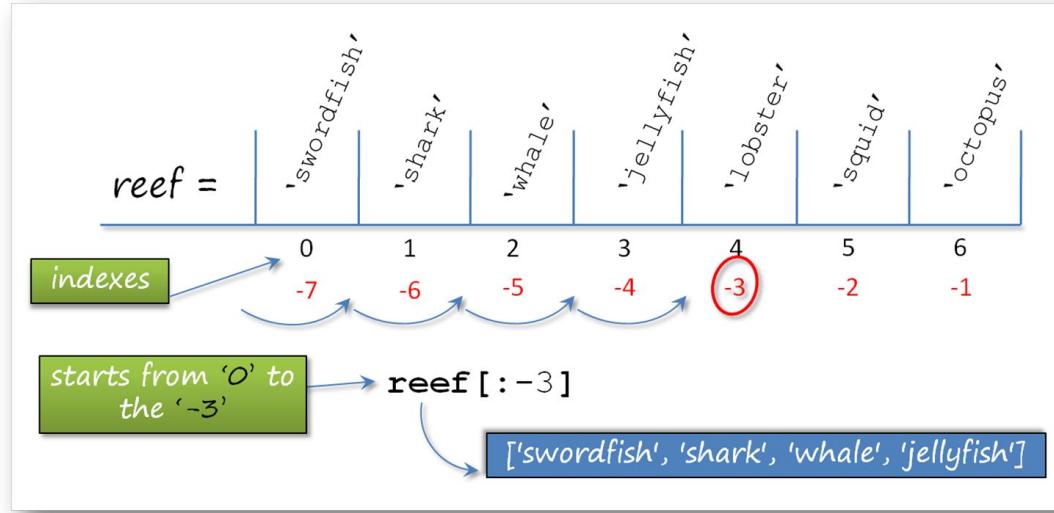


Negative Indexing & Slicing(review)

- 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 stepping in the pre-class content :

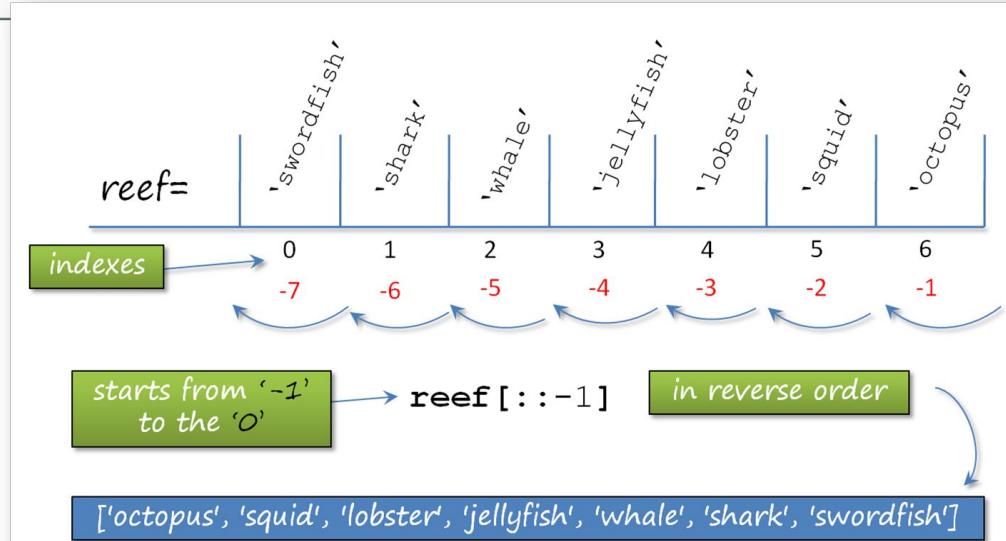
```
1 reef = ['swordfish', 'shark', 'whale', 'jellyfish', 'lobster', 'squid', 'octopus']  
2 print(reef[::-1]) # we have produced the reverse of the list  
3
```

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 **stepping**.

```
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!

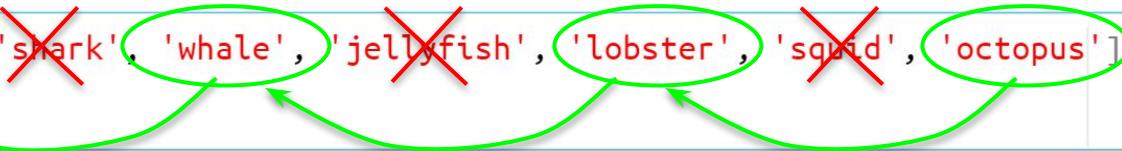
REINVENT YOURSELF

Pear Deck Interactive Slide
Do not remove this bar

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
```



```
1 ['octopus', 'lobster', 'whale', 'swordfish']
2
```

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

```
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])
```



- Starts at index 7 from right to left.
- Goes to index 4.

```
['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])
```

2

6



- 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']
[]
```