Main String Operations

Table of Contents

Searching a String

Searching a String

- `string.startswith()`

  Starts searching from the **beginning** to the end.

- `string.endswith()`

  Starts searching from the **very end** to the beginning.

Searching a String(review pre-class)

- Let's take a look these pre-class examples :

```
1  text = 'www.clarusway.com'
2  print(text.endswith('.com'))
3  print(text.startswith('http:'))
4
```

Searching a String(review pre-class)

- Let's take a look these pre-class examples :

```
1  text = 'www.clarusway.com'
2  print(text.endswith('.com'))
3  print(text.startswith('http:'))
4
```

```
1  True
2  False
3
```

Changing a String

Changing a String

- The formula syntax 👇

```
string.method(arguments)
```

Changing a String

- The summary of some common and the most important methods are as follows 👇

- `str.replace(old, new[, count])` replaces all occurrences of old with the new.

The count argument is optional, and if the optional argument count is given, only the first count occurrences are replaced. `count`: Maximum number of occurrences to replace. **-1** (the default value) means replace all occurrences.

- `str.swapcase()` converts upper case to lower case and vice versa.
- `str.capitalize()` changes the first character of the string to the upper case and the rest to the lower case.
- `str.upper()` converts all characters of the string to the upper case.
- `str.lower()` converts all characters of the string to the lower case.
- `str.title()` converts the first character of each word to upper case.

Changing a String

- Let's grasp these methods through the examples 👇

```
1  sentence = "I live and work in Virginia"
2
3  print(sentence.upper())
4
5  print(sentence.lower())
6
7  print(sentence.swapcase())
8
9  print(sentence)  # note that, source text is unchanged
10
```

Changing a String

- Let's grasp these methods through the examples 👇

```
1  sentence = "I live and work in Virginia"
2
3  print(sentence.upper())
4
5  print(sentence.lower())
6
7  print(sentence.swapcase())
8
9  print(sentence)  # note that, source text is unchanged
10
```

```
1  I LIVE AND WORK IN VIRGINIA
2  i live and work in virginia
3  i LIVE AND WORK IN vIRGINIA
4  I live and work in Virginia
5
```

Changing a String

💡 Note that,

All these methods return `str` type.

So we can use the following syntax.

```
string.method1().method2().method3()...
```

Changing a String

- Let's grasp these methods through the examples 👇

```
1  sentence = "I live and work in Virginia"
2  title_sentence = sentence.title()
3  print(title_sentence)
4
5  changed_sentence = sentence.replace("i", "+")
6  print(changed_sentence)
7
8  print(sentence)  # note that, again source text is unchanged
9
```

Changing a String

- Let's grasp these methods through the examples 👇

```
1  sentence = "I live and work in Virginia"
2  title_sentence = sentence.title()
3  print(title_sentence)
4
5  changed_sentence = sentence.replace("i", "+")
6  print(changed_sentence)
7
8  print(sentence)  # note that, again source text is unchanged
9
```

```
1  I Live And Work In Virginia
2  I l+ve and work +n V+rg+n+a
3  I live and work in Virginia
4
```

Changing a String

- **Task** 👇
  - Let's change the **first letters** of **each words** to **uppercase** of the following text 👇

```
text = 'the better the family, the better the society'
```

Changing a String

- **The code may look like 👇**

```
text = text.title()
print(text)
```

```
The Better The Family, The Better The Society
```

Changing a String

- **Task** 👇
  - In the `text` below, accidentally the number 0(zero) is used instead of the letter ‘o’ (oh). Fix them using `.replace()` method and **change** the **value of the variable** considering the new text and print the result.

```
text = 'S0d0me and G0m0re'
```

Changing a String

- **The code may look like 👇**

```
text = 'S0d0me and G0m0re'
text = text.replace('0', 'o')
print(text)
```

```
Sodome and Gomore
```

Editing a String

Editing a String

- The formula syntax 👇

```
string.method(arguments)
```

Editing a String

- The summary of some common and the most important methods are as follows 👇

- `string.strip()` : removes all spaces (or specified characters) from both sides.
- `string.rstrip()` : removes spaces (or specified characters) from the right side.
- `string.lstrip()` : removes spaces (or specified characters) from the left side.

Editing a String

- Let's review the pre-class examples 👇

```
string.strip(arg)
string.lstrip(arg)
string.rstrip(arg)
```

```
space_string = "     listen first      "
print(space_string.strip())  # removes all spaces from both sides

source_string = "interoperability"
# removes trailing "y" or "i" or "yi" or "iy" from both sides
print(source_string.strip("yi"))

space_string = "     listen first"
print(space_string.lstrip())  # removes all spaces from left sides

source_string = "interoperability"
# removes "i" or "n" or "in" or "ni" from the left side
print(source_string.lstrip("in"))

space_string = "     listen first      "
print(space_string.rstrip())  # removes all spaces from right side

source_string = "interoperability"
# removes "t" or "y" or "ty" or "yt" from the right side
print(source_string.rstrip("ty"))
```

Editing a String

- **Task** 👇
  - In the `text` below, accidentally there are additional letters. Remove the additional letters and make the all words uppercase.
  - Except for the `print()` line, your code should consist of a single line.

```
text = 'tyou can learn almost everything in pre-classz'
```

desired output

```
YOU CAN LEARN ALMOST EVERYTHING IN PRE-CLASS
```

Editing a String

- **The code may look like 👇**

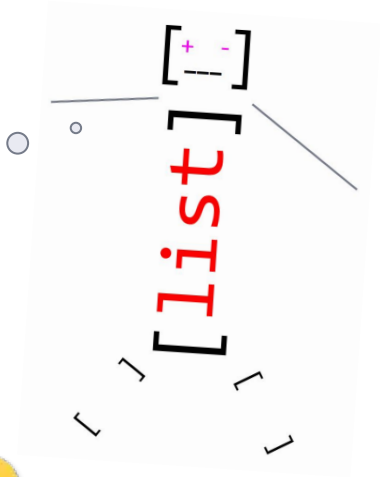
```
text = text.rstrip('z').lstrip('t').upper()
print(text)
```

```
YOU CAN LEARN ALMOST EVERYTHING IN PRE-CLASS
```

Collection Types

- List
- Tuple
- Dictionary
- Set

I'm the best...



Lists

Table of Contents

Creating a `list`

- We have two basic ways to create a list.

- `[]`
- `list()`

Creating a `list`

- The output :

- `[]`
- `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 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
```

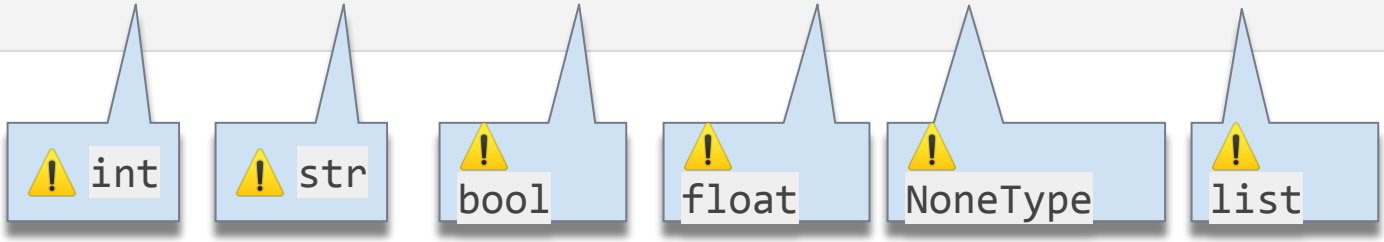
Tips:

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

Creating a `list`

- The components of a `list` are not limited to a single data type.
- Here is an example :

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



Creating a `list`

- Task:

```
my_list = ['Aslan', 'Clarusway', 2022]
new_list1 = list(my_list)
new_list2 = [my_list]

print(new_list1)
print(len(new_list1))
print(new_list2)
print(len(new_list2))
```

Creating a `list`

- The output:.

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

Creating a `list`

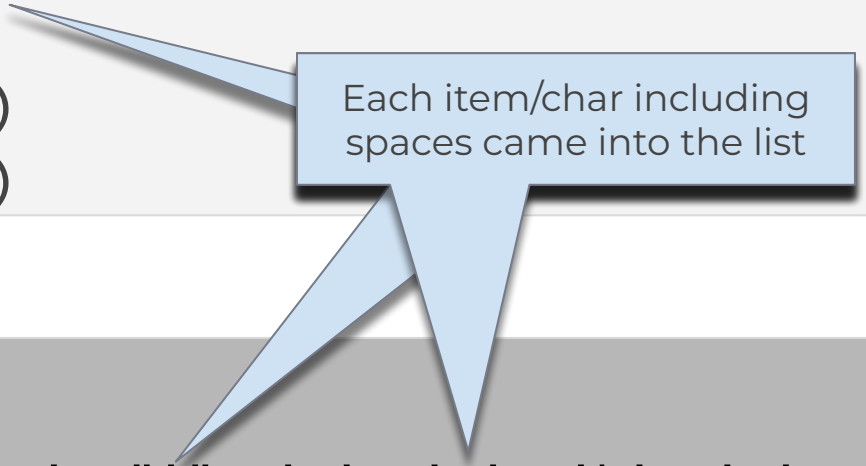
- Task:
  - Create a list from **string** value of `"2022's hard"`.
  - Use both `list()` function and square brackets `[]`.

Creating a `list`

- The code :

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

print(my_list1)
print(my_list2)
```




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

Basic Operations with Lists

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)
```

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
```

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

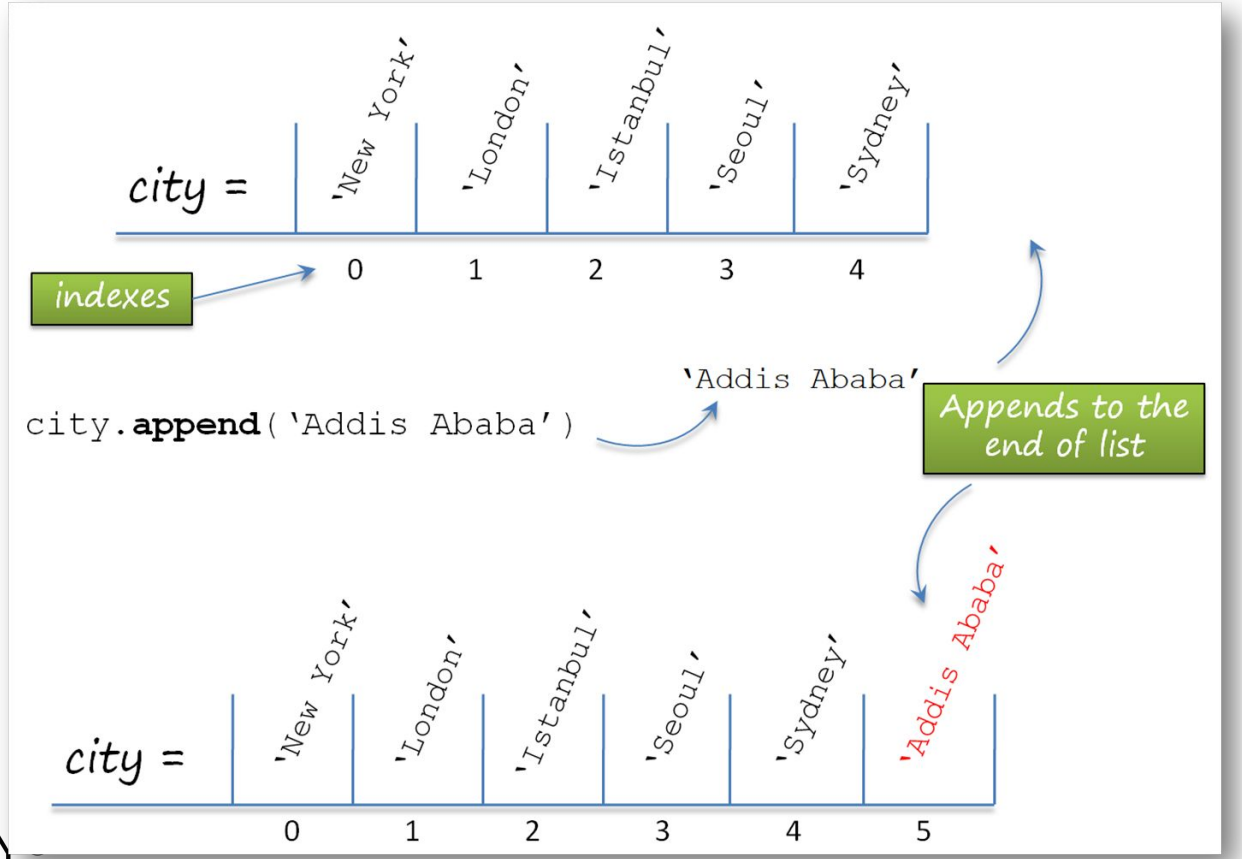
```
[6666, 'Multiverse', [0]]
```

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`

```
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`

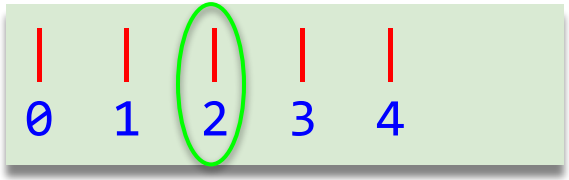
- `.insert()` adds a new object to the `list` at a specific (given) index.

- .append()
- .insert()

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

Adds into index '2'

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



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)
```

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

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)
```

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

Basic Operations with `lists`

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

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

Basic Operations with `lists`

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

```
mix_list =
mix_list.so
print(mix_l
```

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

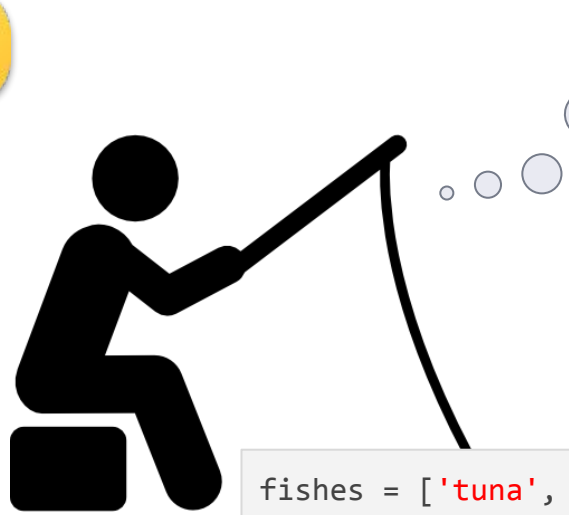
```
TypeError                                                  ent call
last)
<ipython-input-7-ad44188425eb> in <module>
      1 mix_list = ['d', 1, 'a', 7]
----> 2 mix_list.sort()
      3 print(mix_list)

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

Basic Operations with `lists`

- 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]]
```




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

Accessing Lists

Table of Contents

```
fruit[1]
```

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

Indexing a `list`

```
list()
```

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`

```
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`

- 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']

```

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

```

```
Istanbul

```

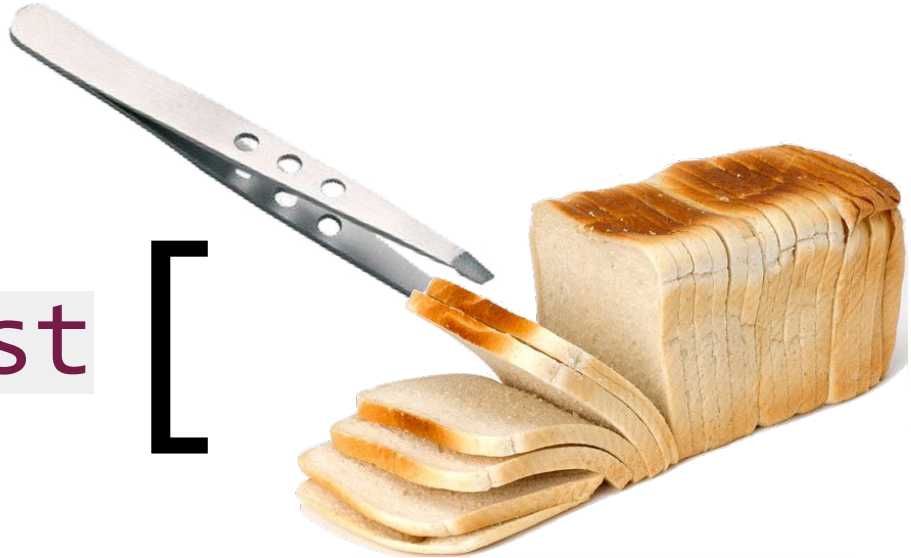
```
city_list = [['New York', 'London', 'Istanbul', 'Seoul', 'Sydney']]
print(city_list[0][2][3])

```

```
a

```

Slicing a `list`



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)

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

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

```
animals = ['elephant', 'bear', 'fox', 'wolf', 'rabbit', 'deer', 'giraffe']
print(animals[3:])  # lists elements from 3rd index to end
```

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

```
animals = ['elephant', 'bear', 'fox', 'wolf', 'rabbit', 'deer', 'giraffe']
print(animals[:5])  # lists elements until 5th index.
```

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

```
animals = ['elephant', 'bear', 'fox', 'wolf', 'rabbit', 'deer', 'giraffe']
print(animals[::2])  # lists elements with 2 step
```

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

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]
```