Programming Skills - Python

Learn the basics of Python Programming

Day - 4

Lists and Tuples

Agenda

4.1. Brief Recap

A brief recap of what we discussed yesterday.

4.2. Introduction to Lists

What they are, why they are used, and how to use them.

4.3. Creating and Modifying

Syntax to create lists and how they can be modified.

4.4. List Operations

Concatenation, repetition, and slicing.

Agenda

4.5. Iterating through Lists

For loops with and without index.

4.6. Mutable v/s Immutable

Defining and understanding.

4.7. Introduction to Tuples

What they are and why they are used.

4.8. Creating and Accessing

Basic operations on Tuples.

Agenda

4.9. Tuple Operations

Concatenation and repetetion.

4.10. Packing and Unpacking

Understanding and using.

4.11. Immutability

Limitations and advantages of tuples.

4.12. When to use what

Understanding when to prefer tuples over lists and vice-versa.

4.1. Brief Recap

4.2. Introduction to Lists

Lists

What are they?

A list is a collection of items in a particular order.

Why are they used?

To store data that can be grouped in a single variable.

How to use it?

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

Replit →

4.3. Creating and Modifying

Creating a List

Empty list:

```
my_list = []
```

List with elements:

```
fruits = [“apple”, “banana”]
```

List can contain multiple datatypes:

```
mixed = [1, "str", 3.14, True]
```

Modifying a List

Add element at the end:

```
fruits.append(“mango”)
```

Add element at index:

```
fruits.insert(1, “mango”)
```

Remove a specific element:

```
fruits.remove(“mango”)
```

Remove by index:

```
fruits.pop(2)
```

4.4. List Operations

Concatenation

Combining of two or more lists.

Example:

```
list1 = [1, 2]
list2 = [3, 4]
combined = list1 + list2
```

Replit →

Repetition

Repeating one element multiple times.

Example:

```
repeated = [0] * 3  # [0, 0, 0]
```

Replit →

Slicing

Creating and accessing sub lists.

Example:

```
my_list = [1, 2, 3, 4, 5]
slice = my_list[1:3]  # [2, 3]
```

Replit →

4.5. Iterating through Lists

Interating using Index

Example:

```
for i in range(len(fruits)):
    print(fruits[i])
```

Replit →

Interating without Index

Example:

```
for fruit in fruits:
    print(fruit)
```

Replit →

4.6. Mutable v/s Immutable

Mutable v/s Immutable

Mutability means ability to change or modify. Lists can be modified after being defined, so lists are mutable.

```
my_list = [1, 2, 3]
my_list[0] = 10

my_str = "hello"
my_str[0] = "H"  # error.
```

4.7. Introduction to Tuples

Introduction to Tuples

A tuple is a collection of ordered items like lists.

Tuples are immutable.

Tuples are defined by ().

```
my_tuple = (1, 2, 3)
```

4.8. Creating and Accessing

Creating Tuples

Creating an empty tuple:

```
empty_tuple = ()
```

Creating tuple with elements:

```
fruits = (“apple”, “orange”)
```

Creating a single element tuple:

```
single_element = (1,)
```

Replit →

Accessing Tuples

Indexing:

```
fruits = (“apple”, “orange”,
“banana”)
print(fruits[1]) # orange
```

Slicing:

```
print(fruits[0:2])
```

Replit →

4.9. Tuple Operations

Concatanation

Example:

```
tuple1 = (1, 2)
tuple2 = (3, 4)
result = tuple1 + tuple2
# (1, 2, 3, 4)
```

Replit →

Repetition

Example:

```
repeated = (0,) * 2  # (0, 0)
```

Replit →

4.10. Packing and Unpacking

Packing

Example:

```
packed = 1, 2, "apple"
print(packed)
# Output: (1, 2, "apple")
```

Replit →

Unpacking

Example:

```
a, b, c = packed
print(a, b, c)
# Output: 1 2 apple
```

Replit →

4.11. Immutability of Tuples

Immutability of Tuples

Tuples cannot be modified after creation. If you want to modify a tuple, you have to create a new tuple by concatenation, and the original tuple remains unchanged.

4.12. When to use What

Lists v/s Tuples

Tuples are to be used when immutability is desired. Since they are immutable, they are also slightly faster.

Prefer lists when the data needs to be modified.

Q&A

Thank You!