



# Python Track

Basic data structures  
(Lists and Tuples)

# Lecture Flow

- Lists
- Tuples



# Lists



# What are lists?

- Lists are a fundamental data structures in Python used to store collections of data.
- They can hold items of any data type, including numbers, strings, and even other lists.
- Lists are ordered, changeable, and allow duplicate values.



# Creating lists

- Lists can be created using square **brackets []** and separating items with commas.
- The **list()** constructor can also be used to create lists.

```
# Creating a list using square brackets  
fruits = ["apple", "banana", "cherry"]
```

```
# Convert other data structures to list using the list()  
constructor  
numbers = list((1, 2, 3, 4, 5))
```

# List data types

can be of any data type

```
list1 = ["apple", "banana", "cherry"]
```

```
list2 = [1, 5, 7, 9, 3]
```

```
list3 = [True, False, False]
```

```
list4 = ["abc", 34, True, 40, "male"]
```

# Accessing List Items

- List items are accessed using their index number, starting from 0.
- Negative indexing can be used to access items from the end of the list.

```
nums = [12, 34, 42, 63, 47, 58, 63, 37, 98, 90]
```

```
# Accessing the first item  
nums[0]      # 12
```

```
# Accessing the last item  
nums[-1]     # 90
```

# Slicing Lists

- Slicing allows extracting a sublist from a list.
- Slicing uses the **colon (:)** to separate start and end indices (inclusive).
- Slicing follows the format: [Start : End : Step].

```
nums = [0, 41, 23, 36, 74, 59, 76, 78, 28, 9]  
# Extracting a sublist from index 2 to index 4
```

```
nums[2 : 5]      # [23, 36, 74]
```

```
nums[-4 : -1] ??
```

# Modifying Lists

- Lists are mutable, allowing you to change their contents.
- You can modify items using their index or extend the list using **append()** and **insert()**.
- You can also remove items using **remove()** and **pop()**.

# Examples

```
fruits = ["apple", "banana", "cherry"]
```

```
# Changing the first item
```

```
fruits[0] = "orange" # fruits = ["orange", "banana", "cherry"]
```

```
# Adding an item to the end
```

```
fruits.append("mango") # fruits = ["orange", "banana", "cherry", "mango"]
```

```
# Removing an item by value
```

```
fruits.remove("cherry") # fruits = ["orange", "banana", "mango"]
```

```
# Removing the last item
```

```
removed_item = fruits.pop() # removed_item = "mango", fruits =  
["orange", "banana"]
```

# Common List Operations

- Checking if an item exists: `in` keyword
- Sorting a list: `sort()` method
- `sorted( nums , key = myFunction() , reverse = True/False)`
- Reversing a list: `reverse()` method

# Examples

```
# Checking if "apple" exists in the list  
  
if "apple" in fruits:  
  
    print("Yes, apple is in the list")  
  
# Sorting the list in ascending order  
  
fruits.sort() # fruits = ["banana", "orange"]  
  
# Reversing the sorted list  
  
fruits.reverse() # fruits = ["orange", "banana"]
```

# Examples

```
# Sorting a list of words based on their lengths
words = ["apple", "banana", "cherry", "date"]

# Using the key parameter to sort by word length
sorted_words = sorted(words, key=len)

print(sorted_words) # Output:
['date', 'apple', 'banana', 'cherry']
```

# Examples

```
import copy

original_list = [[1, 2, 3], [4, 5, 6]]

shallow_copied_list = copy.copy(original_list)

deep_copied_list = copy.deepcopy(original_list)

# Modifying the nested list in the original

original_list[0][0] = 99

print(shallow_copied_list) # Output: [[99, 2, 3], [4, 5, 6]]  
  
(Affected)

print(deep_copied_list) # Output: [[1, 2, 3], [4, 5, 6]] (Unaffected)
```

# Combining Lists

- Concatenating lists using the + operator or extend() method
- Adding items from one list to another individually

# Examples

```
numbers = [1, 2, 3]
```

```
fruits = ["orange", "banana"]
```

```
# Concatenating lists using '+' operator
```

```
new_list = fruits + numbers # new_list = ["orange", "banana", 1, 2, 3]
```

```
# Extending a list using extend() method
```

```
fruits.extend(numbers) # fruits = ["orange", "banana", 1, 2, 3]
```

# Traversing Lists

- Iterating through lists using for loops
- Accessing both index and value using enumerate() function

- `for index in range(len(nums)):  
 print(nums[index])`
- `for num in nums:  
 print(num)`
- `for index, num in enumerate(nums):  
 print(index, num)`

# List Comprehension

- Creating new lists based on existing lists
- Using expressions and conditions to filter and transform list elements

```
# Creating a list of even numbers from a list of numbers
```

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

```
even_numbers = [num for num in numbers if num % 2 == 0]
```

```
# even_numbers = [2, 4]
```

# Other List Methods

Method	Description
<u>append()</u>	Adds an element at the end of the list
<u>clear()</u>	Removes all the elements from the list
<u>copy()</u>	Returns a copy of the list
<u>count()</u>	Returns the number of elements with the specified value
<u>extend()</u>	Add the elements of a list (or any iterable), to the end of the current list
<u>index()</u>	Returns the index of the first element with the specified value
<u>insert()</u>	Adds an element at the specified position
<u>pop()</u>	Removes the element at the specified position
<u>remove()</u>	Removes the item with the specified value
<u>reverse()</u>	Reverses the order of the list
<u>sort()</u>	Sorts the list

# Tuples



# What are Tuples?

- A tuple is a collection which is **ordered**, allows **duplicates** and is **unchangeable**. Tuples are also known as **Immutable Lists**.
- Tuples are written with parenthesis.
  - `fruits = ("apple", "banana", "cherry")`
  - `fruit = ("apple",)`



# Creating Tuples

- Tuples are written with **round** brackets **()**.

```
fruits = ("apple", "banana", "cherry")
```

```
fruit = ("apple",) # or just () to create an empty one
```

- The **tuple()** constructor:

```
fruits = tuple(["apple", "banana", "cherry"])
```

```
numbers = tuple()
```

# Tuples

- Is it possible to
  - add an element to a Tuple? How?
  - delete an element?
  - join two tuples?



# Tuple Similarities with List

- Similar data types
- Slicing and Indexing
- Similar Iteration

Q: Is it possible to have “Tuple Comprehension” ?

# Tuple Methods

<b>Method</b>	<b>Description</b>
<u>count()</u>	Returns the number of times a specified value occurs in a tuple
<u>index()</u>	Searches the tuple for a specified value and returns the position of where it was found

# Problems

[Lists](#)

[Build Array from Permutation](#)

[Presents](#)

[Maximum Product of Three Numbers](#)

## Quote of the Day

**“A boat doesn’t go forward if each one is rowing their own way.” – Swahili Proverb**