Newton School of Technology

16

Tuples in Python

by Kunal Goyal

C14: Fundamentals of Programming

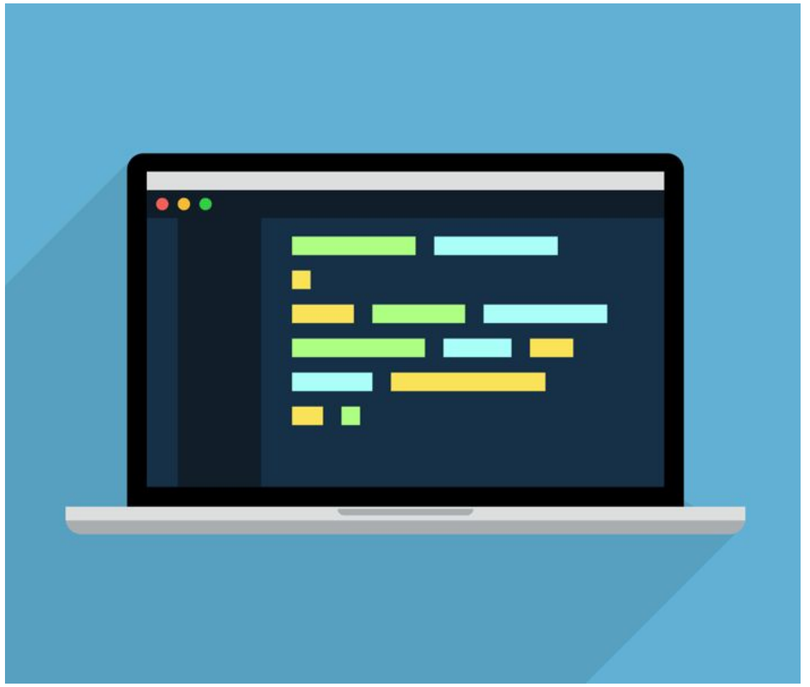
Quick Recap :

- **Iterating Lists**
- **Built-in List Functions:**

  min(), max(), any(), all(), sum()

- **List Methods:**

  sort(), sorted(), reverse(), count(), index()



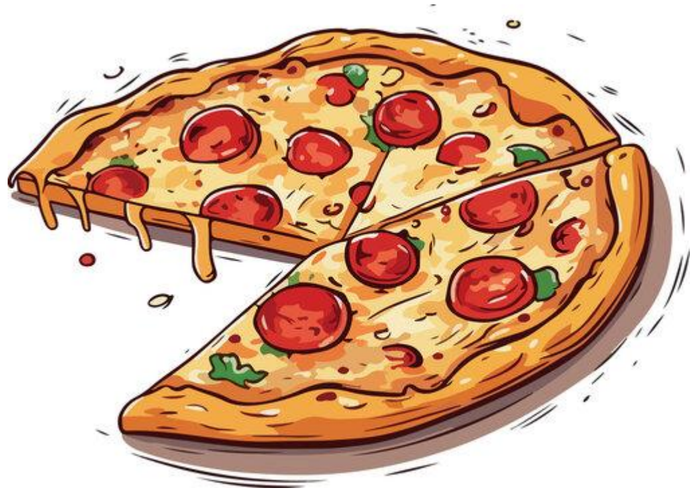
Introduction to Tuples

Aman is one of the best cooks in Newton Restaurant.

Introduction to Tuples

Newton School of Technology

- He has invented a recipe for making a very tasty pizza.
- To implement this he has listed down some ingredients.
- He decides to store these items in a list for his teammates to access and use.



Introduction to Tuples

But there is a mischievous chef who wants to change the items on this list and spoil his recipe.

Introduction to Tuples

Newton School of Technology

Being aware of his plans Aman decides to store his recipe in a data structure which is immutable!

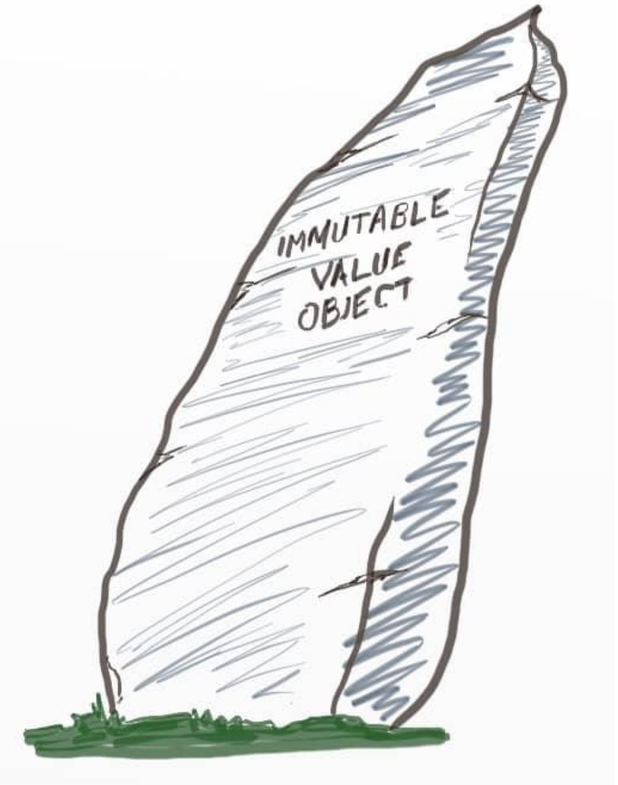
Newton School of Technology

Yes its tuples!

Definition :

Newton School of Technology

- In Python, a tuple is an immutable sequence of elements.
- This means that once a tuple is created, its contents cannot be modified—no elements can be added, removed, or changed.



Syntax:

Newton School of Technology

Tuples are defined using **parentheses ()** to enclose their elements, whereas lists use s**quare brackets []**. Here's a simple comparison:

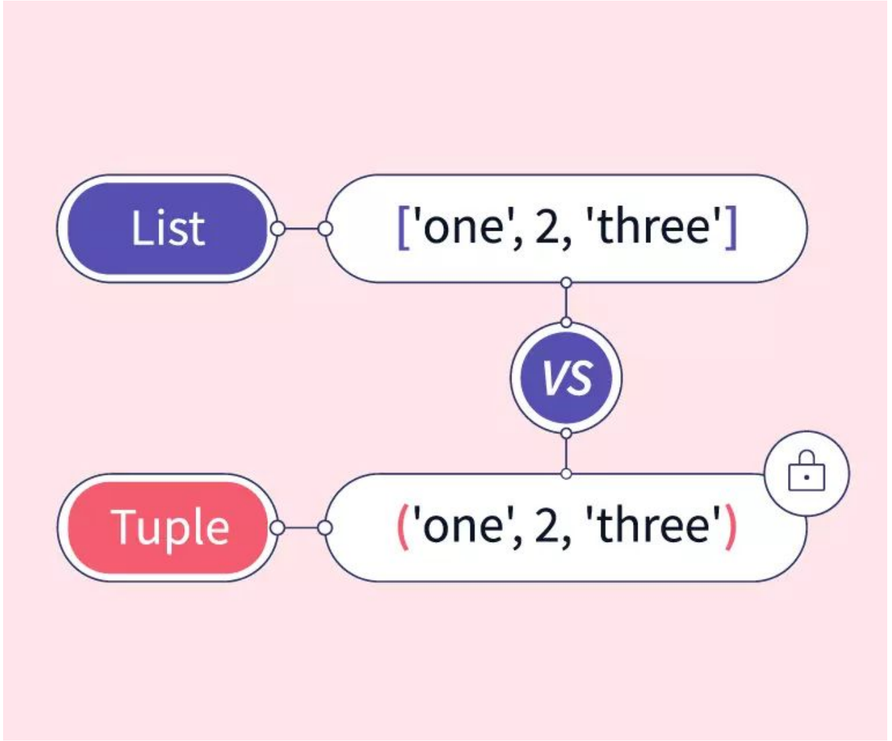
- **Tuple Syntax**: `(element1, element2, element3)`
- **List Syntax**: `[element1, element2, element3]`

python Copy code

```python
my_tuple = (1, 2, 3)
my_list = [1, 2, 3]
```

Mutability :

- **List:** Mutable, meaning you can modify it (e.g., add, remove, or change elements).
- **Tuple:** Immutable, meaning once created, it cannot be altered.



Tuples usability

Newton School of Technology

Using a Tuple for Fixed Data

Tuples are ideal when dealing with data that should remain constant throughout your program, like coordinates, dates, or configuration settings.

Scenario: Storing Coordinates

Coordinates are often fixed and don’t need to be modified, making tuples a good choice.

```
# Tuple example: coordinates
coordinates = (40.7128, -74.0060)  # New York City latitude and longitude
print("Coordinates of NYC:", coordinates)

# Attempting to modify the coordinates would raise an error
# coordinates[0] = 41.0  # This would result in a TypeError
```

Tuples usability

Newton School of Technology

- **Safety:** When you want to ensure that the data cannot be altered by any part of the program, tuples provide a way to safeguard this data, avoiding unintended side effects.

- **Hashability** : Because tuples are immutable, they can be used as keys in dictionaries or elements in sets, unlike lists. This is a result of their hashable nature, which relies on their fixed structure.

Tuples usability

Newton School of Technology

- **Multiple Return Values:** Tuples are often used to return multiple values from a function.

Example :

```python
def get_coordinates():
    return (40.7128, 74.0060)

lat, lon = get_coordinates()
```

Newton School of Technology

Creating Tuples

Empty Tuple

Newton School of Technology

An empty tuple is a tuple that contains no elements. It is created by simply using a pair of parentheses with no content inside.

```python
# Creating an empty tuple
empty_tuple = ()
print(empty_tuple)  # Output: ()
```

Multiple Elements Tuple

Newton School of Technology

- A tuple with multiple elements is created by placing the elements within **parentheses and separating them with commas**.
- Tuples can contain any number of elements and can include different **data types**.

python

```python
# Creating a tuple with multiple elements
multi_element_tuple = (1, "hello", 3.14)
print(multi_element_tuple)  # Output: (1, 'hello', 3.14)
```

Single Element Tuple

Newton School of Technology

- To create a tuple with a single element, you must include a trailing comma after the element.
- **Without the comma, Python will interpret the parentheses as just enclosing the element, not creating a tuple.**

```python
# Creating a single-element tuple
single_element_tuple = (5,)
print(single_element_tuple)  # Output: (5,)
```

Single Element Tuple

Newton School of Technology

Let’s see what will happen if we omit the **comma**.

```python
not_a_tuple = (5)
print(type(not_a_tuple))  # Output: <class 'int'>
```

Newton School of Technology

Accessing Elements in Tuple

Indexing

Newton School of Technology

Indexing allows you to retrieve elements from a tuple by specifying their **position**. Python uses **zero-based** indexing, meaning the first element has an index of `0`, the second has an index of `1`, and so on.

python

```python
# Creating a tuple
my_tuple = (10, 20, 30, 40, 50)

# Accessing elements using indexing
first_element = my_tuple[0]  # 10
third_element = my_tuple[2]  # 30

print(first_element)  # Output: 10
print(third_element)  # Output: 30
```

Negative Indexing

Newton School of Technology

Negative indexing allows you to access elements **from the end** of the tuple. The last element has an index of `-1`, the second-to-last element has an index of `-2`, and so on.

python

```python
# Creating a tuple
my_tuple = ('a', 'b', 'c', 'd', 'e')

# Accessing elements using negative indexing
last_element = my_tuple[-1]   # 'e'
second_last_element = my_tuple[-2]  # 'd'

print(last_element)  # Output: 'e'
print(second_last_element)  # Output: 'd'
```

Slicing Tuples

Newton School of Technology

Slicing allows you to access a range of elements from a tuple. You can specify a starting index, a stopping index, and a **step value**. The syntax for slicing is `tuple[start:stop:step]`, where:

- **start:** The index where the slice begins (inclusive).
- **stop:** The index where the slice ends (exclusive).
- **step:** The interval between elements in the slice (optional).

Slicing Tuple

Newton School of Technology

```python
# Creating a tuple
my_tuple = (1, 2, 3, 4, 5, 6, 7, 8, 9)

# Slicing the tuple
slice1 = my_tuple[2:5]   # (3, 4, 5)
slice2 = my_tuple[:4]    # (1, 2, 3, 4)
slice3 = my_tuple[5:]    # (6, 7, 8, 9)
slice4 = my_tuple[::2]   # (1, 3, 5, 7, 9)

print(slice1)  # Output: (3, 4, 5)
print(slice2)  # Output: (1, 2, 3, 4)
print(slice3)  # Output: (6, 7, 8, 9)
print(slice4)  # Output: (1, 3, 5, 7, 9)
```

Newton School of Technology

Tuple Operations

Concatenation

Newton School of Technology

- Concatenation allows you to join two tuples together into a single, larger tuple.
- This is done using the + operator.
- This operation does not modify the original tuples but produces a new one that combines their contents.

```python
tuple1 = (1, 2, 3)
tuple2 = (4, 5, 6)
concatenated_tuple = tuple1 + tuple2
print(concatenated_tuple)  # Output: (1, 2, 3, 4, 5, 6)
```

Repetition

Newton School of Technology

- Repetition allows you to create a new tuple by repeating the elements of an existing tuple a specified number of times.
- This is done using the * operator.

```python
tuple1 = (1, 2, 3)
repeated_tuple = tuple1 * 3
print(repeated_tuple)  # Output: (1, 2, 3, 1, 2, 3, 1, 2, 3)
```

Membership Test (in keyword)

Newton School of Technology

- The membership test allows you to check whether a specific element exists within a tuple.
- This is achieved using the **in** keyword, which returns **True** if the element is found in the tuple and **False** otherwise.

```
# Tuple of weekend days
weekends = ("Saturday", "Sunday")

# Day to check
day = "Saturday"

# Membership test
if day in weekends:
    print(f"{day} is a weekend!")
else:
    print(f"{day} is a weekday.")
```

python

```
tuple1 = (1, 2, 3, 4)
print(2 in tuple1)  # Output: True
print(5 in tuple1)  # Output: False
```

len() Function

Newton School of Technology

- The **length operation** allows you to determine the number of elements present in a tuple.
- This is done using the **len()** function, which returns an integer representing the **number of items** in the tuple.

```python
tuple1 = (1, 2, 3, 4)
length = len(tuple1)
print(length)  # Output: 4
```

Newton School of Technology

Tuple Methods

Count method

Newton School of Technology

- This method takes a single argument, which is the value whose occurrences you want to count.
- It returns an integer representing how many times that value appears in the tuple.

```python
tuple1 = (1, 2, 3, 2, 4, 2)
count_of_2 = tuple1.count(2)
print(count_of_2)  # Output: 3
```

Index method

Newton School of Technology

- The index method helps you find the **first occurrence** of a specific value within a tuple.
- This method takes the value you want to find and returns its **index** (position) in the tuple.
- If the value is not present, it raises a **ValueError**.

```python
tuple1 = (10, 20, 30, 40, 30)
index_of_30 = tuple1.index(30)
print(index_of_30)  # Output: 2
```

Newton School of Technology

Iteration in Tuples

Using ‘for’ loop

Newton School of Technology

A **for loop** in Python provides a simple and intuitive way to iterate through **each item** in a tuple. This method allows you to access **each element** one by one and perform operations on it.

Syntax:

```
for item in tuple:
    #perform operations with item
```

Example using ‘for’ loop

Newton School of Technology

python

```python
fruits = ('apple', 'banana', 'cherry', 'date')

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

bash

```
apple
banana
cherry
date
```

Using enumerate() for iteration

Newton School of Technology

- In Python, enumerate() is a built-in function that allows you to loop over a list (or any iterable) while keeping track of both the index and the value of each item.
- This is especially useful when you need to work with both the position and content of each element in a list.

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

for index, fruit in enumerate(fruits, start=1):
    print(f"{index}: {fruit}")
```

Newton School of Technology

Tuple Packing and Unpacking

Packing and Unpacking

Packing : One person gets all the melody’s.

Unpacking : Each person gets one melody.

Packing

Newton School of Technology

- **Packing** refers to the process of assigning multiple values to a single variable, which creates a tuple. This is a straightforward and efficient way to group related values together.
- Tuples are automatically created when multiple values are **separated by commas**, even if parentheses are omitted.

```python
packed_tuple = 1, 2, 'a'
print(packed_tuple)  # Output: (1, 2, 'a')
```

Unpacking

Newton School of Technology

- **Unpacking** refers to the process of **extracting** the values from a tuple and assigning them to **individual variables**.
- This is useful for when you need to work with each value separately after it has been grouped into a tuple.
- Tuples are automatically created when multiple values are **separated by commas**, even if parentheses are omitted.

```
packed_tuple = 1, 2, 'a'
a, b, c = packed_tuple
print(a)  # Output: 1
print(b)  # Output: 2
print(c)  # Output: 'a'
```

Newton School of Technology

Nested Tuples

Nested Tuples

Newton School of Technology

- A nested tuple is essentially a tuple that contains one or more tuples as its elements.
- This allows you to create multi-dimensional data structures and manage related groups of data efficiently.

Example:

```python
nested_tuple = (1, (2, 3), 4)
```

Nested Tuples Examples

Newton School of Technology

```python
nested_tuple = (1, (2, 3), 4)
nested_element = nested_tuple[1][0]
print(nested_element)  # Output: 2
```

- In this example: **nested_tuple[1]** accesses the second element of nested_tuple, which is the tuple **(2, 3)**.
- **nested_tuple[1][0]** accesses the first element of the nested tuple (2, 3), which is **2**.

Thank You!