Python Programming

Prepared by

Dr. Ramesh Saha

Assistant professor

Computer Science and Engineering

Indian Institute of Information Technology (IIIT), Sonepat

UNIT III Python

Lists: Values and Accessing Elements, Lists are mutable, traversing a List, Deleting elements from List, Built-in List Operators, Concatenation, Repetition, In Operator, Built-in List functions and methods Tuples and Dictionaries: **Tuples**, Accessing values in Tuples, Tuple Assignment, Tuples as return values, Variable-length argument tuples, Basic tuples operations, Concatenation, Repetition, in Operator, Iteration, Built-in Tuple Functions Creating a Dictionary, Accessing Values in a dictionary, Updating Dictionary, Deleting Elements from Dictionary, Properties of Dictionary keys, Operations in Dictionary, Built-In Dictionary Functions, Built-in Dictionary Method

Tuples

- A tuple is the same as a list, except that the set of elements is **enclosed in parentheses** instead of square brackets.
- **Tuple is an immutable list**. i.e. once a tuple has been created, you can't add elements to a tuple or remove elements from the tuple.

Definition

A tuple is an ordered sequence of elements of different data types, such as integer, float, string, list or even a tuple. Elements of a tuple are enclosed in parenthesis (round brackets) and are separated by commas. Like list and string, elements of a tuple can be accessed using index values, starting from 0.

❖ Fundamental Data Types vs Immutability :

- All Fundamental Data types are immutable.
  i.e., once we create an object, we cannot perform any changes in that object.
- If we are trying to change then with those changes a new object will be created.

This non-changeable behaviour is called immutability.

```
In [67]:

x = 10
print(id(x))
x = x+1
print(id(x))

140714560754784
140714560754816
```

```
x = 10
y = x
print(id(x))
print(id(y))
y = y + 1
print(x)
print(y)
print(id(x))
print(id(y))

140714560754784
140714560754784
10
11
140714560754784
140714560754816
```

Tuples

❖ Fundamental Data Types vs Immutability: Need of Immutability

➢ Why Immutability?

Who is responsible for creating an Object in Python?

✓ Python Virtual Machine (PVM) is responsible for creating an object in Python.

In Python, if a new object is required, then PVM won't create an object immediately.

First, it will check if any object is available with the required content or not.
If available, then the existing object will be reused. If it is not available then only a new object will be created.

```
In [75]:

a = 10
b = 10
c = 10
print(id(a))
print(id(b))
print(id(c))

140714560754784
140714560754784
140714560754784
```

The advantage of this approach is memory utilization and performance will be improved.

Tuples

❖ Immutability vs Mutability

➢ The reusability concept is not applicable to complex types.

```
In [3]:

a=10+20j
b=10+20j
print(a is b)

False
```

#editor specific result

We can execute Python programs or scripts from Python IDLE console or Python console also. These things are called as **REPL (Read Evolve Print under Loop)** tools. These tools are not standard editors for executing Python programs. These are used to test small code only.

```
In [9]:

a = 100
b = 100
print(a is b)

True

In [14]:

a = 256
b = 256
print(a is b)

True
```

```
In [19]:

a = 257
b = 257
print(a is b)

False
```

#maybe this editor range is restricted to 0 ==> 256 only

Tuples

- A tuple is same as list, except that the set of elements is **enclosed in parentheses** instead of square brackets.
- **Tuple is an immutable list**. i.e. once a tuple has been created, you can't add elements to a tuple or remove elements from the tuple.

Benefits of Tuple:

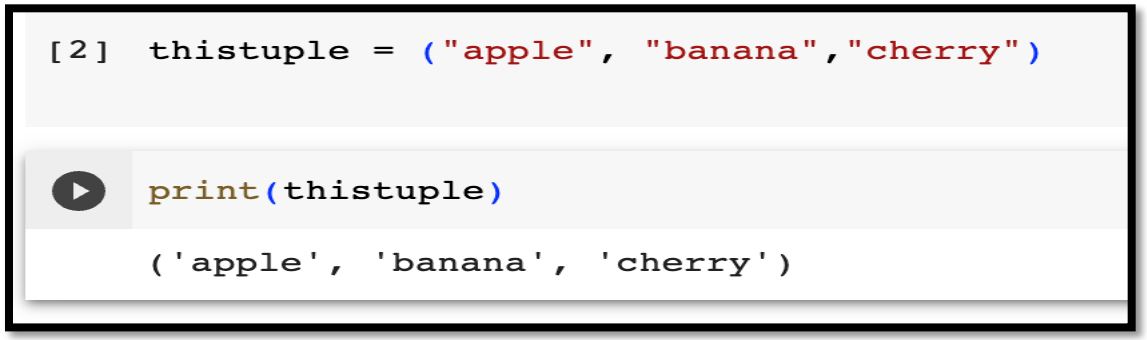
- Tuples are faster than lists.
- If the user wants to protect the data from accidental changes, a tuple can be used.
- Tuples can be used as keys in dictionaries, while lists can’t.

What is the difference between a list and a tuple?

- The list is mutable and the tuple is non-mutable.
- List elements are represented by using square brackets. Tuple elements are represented by using parenthesis.
- To store tuple elements, Python Virtual Memory requires less memory. To store list elements, Python Virtual Memory requires more memory.
- Tuple elements can be accessed within less time because they are fixed (Performance is more).
  Performance is less compared with tuples.

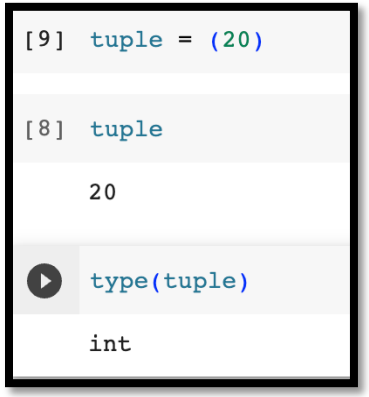
Tuples

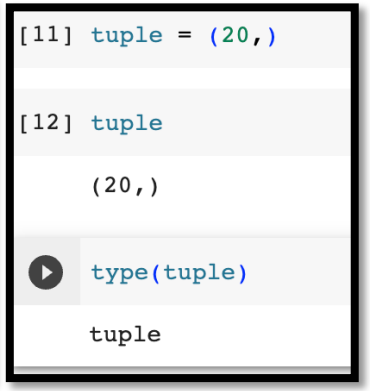
- Tuples are used to **store multiple items in a single variable.**
- Tuple is one of 4 built-in data types in Python used to store collections of data, the other 3 are **List, Set, and Dictionary,** all with different qualities and usage.
- A tuple is a collection which is ordered and unchangeable.
- Tuples are written with round brackets.

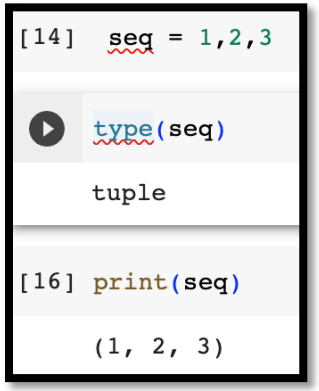


Key point:

- If there is only a single element in a tuple then the element should be followed by a comma. If we assign the value without a comma it is treated as an integer. It should be noted that a sequence without parenthesis is treated as a tuple by default.







Tuples

Tuple Items

Tuple items are ordered, unchangeable, and allow duplicate values.

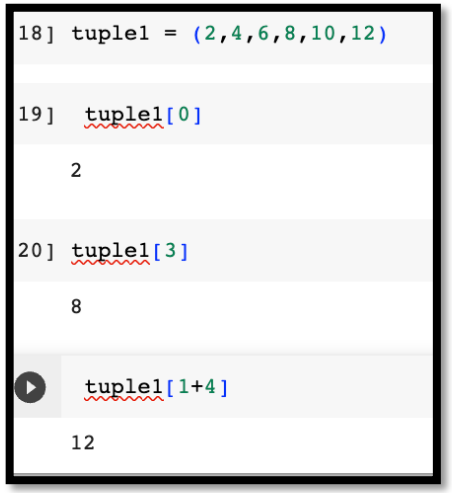
Tuple items are indexed, the first item has index [0], the second item has index [1] etc.

Ordered

When we say that tuples are ordered, it means that the items have a defined order, and that order will not change.

Unchangeable

Tuples are unchangeable, meaning that we cannot change, add or remove items after the tuple has been created.



Tuples

Allow Duplicates

Since tuples are indexed, they can have items with the same value

```
[23] thistuple= ("apple", " banana","cherry", "apple")

     print(thistuple)

     ('apple', ' banana', 'cherry', 'apple')
```

Tuple Length

To determine how many items a tuple has, use the len() function:

```
[23] thistuple= ("apple", " banana","cherry", "apple")

[24] print(thistuple)

     ('apple', ' banana', 'cherry', 'apple')

     print(len(thistuple))

     4
```

Tuples

Create Tuple With One Item

To create a tuple with only one item, you have to add a comma after the item, otherwise Python will not recognize it as a tuple.

Example

One item tuple, remember the comma:

```
thistuple = ("apple",)
print(type(thistuple))

#NOT a tuple
thistuple = ("apple")
print(type(thistuple))
```

Tuples

Tuple Items - Data Types

Tuple items can be of any data type:

Example

String, int and boolean data types:

```
tuple1 = ("apple", "banana", "cherry")
tuple2 = (1, 5, 7, 9, 3)
tuple3 = (True, False, False)
```

```
# ('apple', 'banana', 'cherry')
  (1, 5, 7, 9, 3)
  (True, False, False)
```

A tuple with strings, integers and boolean values:

```
tuple1 = ("abc", 34, True, 40, "male")
```

```
# ("abc", 34, True, 40, "male")
```

Tuples

❖ TUPLE OPERATIONS

➢ Concatenation

Python allows us to join tuples using the concatenation operator depicted by the symbol +. We can also create a new tuple that contains the result of this concatenation operation.

```
[29] tuple1 = (1,3,5,7,9)

[30] tuple2 = (2,4,6,8,10)

     tuple1 + tuple2

     (1, 3, 5, 7, 9, 2, 4, 6, 8, 10)
```

```
[32] tuple3 = ('Red','Green','Blue')

[33] tuple4 = ('Cyan', 'Magenta', 'Yellow' ,'Black')

[34] tuple5 = tuple3 + tuple4

[35] tuple5

     ('Red', 'Green', 'Blue', 'Cyan', 'Magenta', 'Yellow', 'Black')
```

A concatenation operator can also be used for extending an existing tuple. When we extend a tuple using concatenation a new tuple is created.

```
[3]  tuple8 = (1,2,3,4,5)

[4]  tuple8 = tuple8 + (6,)

[5]  print(tuple8)

     (1, 2, 3, 4, 5, 6)
```

Tuples

❖ TUPLE OPERATIONS

➢ Repetition

Repetition operation is depicted by the symbol *. It is used to repeat elements of a tuple. We can repeat the tuple elements. The repetition operator requires the first operand to be a tuple and the second operand to be an integer only.

```
tuple1 = ('Hello','World')
tuple1 * 2

('Hello', 'World', 'Hello', 'World')
```

```
[42] tuple2 = ('Red','Green','Blue')

'Green' in tuple2

True
```

➢ Membership

The in operator checks if the element is present in the tuple and returns True, else it returns False.

➢ Slicing

Like string and list, slicing can be applied to tuples also.

```
tuple3[::-1] #tuple is traversed in reverse order

(80, 70, 60, 50, 40, 30, 20, 10)
```

```
[51] tuple3 = (10,20,30,40,50,60,70,80)

[53] tuple3[2:7]

     (30, 40, 50, 60, 70)

[54] tuple3[0:len(tuple3)] #all elements of tuple are printed

     (10, 20, 30, 40, 50, 60, 70, 80)

[55] tuple3[:5]    #slice starts from zero index

     (10, 20, 30, 40, 50)

     tuple3[2:] #slice is till end of the tuple

     (30, 40, 50, 60, 70, 80)
```

Tuples

Tuple Methods

Python has two built-in methods that you can use on tuples.

Method	Description
count()	Returns the number of times a specified value occurs in a tuple
index()	Searches the tuple for a specified value and returns the position of where it was found
tuple()	Creates an empty tuple if no argument is passed. Creates a tuple if a sequence is passed as an argument.

Count ( )

```
[5] tuple1 = (10,20,30,10,40,10,50)

[6] tuple1.count(10)

    3

    tuple1.count(90)

    0
```

Index ( )

```
]  tuple1 = (10,20,30,40,50)


   tuple1.index(30)

   2
```

tuple ( )

```
[13] tuple5 = tuple ()
     tuple5

     ()

     tuple6= tuple('ramesh') #string
     tuple6

     ('r', 'a', 'm', 'e', 's', 'h')
```

Tuples

Tuple Methods

Python has two built-in methods that you can use on tuples.

Method	Description
sorted()	Takes elements in the tuple and returns a new sorted list. It should be noted that, sorted() does not make any change to the original tuple
Min() & Max()	Returns minimum or smallest element of the tuple, Returns maximum or largest element of the tuple
Sum ()	Returns sum of the elements of the tuple

sorted ( )

```
tuple7 = ("Rama","Heena","Raj",
"Mohsin","Aditya")
sorted (tuple7)
```

```
['Aditya', 'Heena', 'Mohsin', 'Raj', 'Rama']
```

Min( ) & Max ( ) & sum ()

```
[23] tuple8 = (19,12,56,18,9,87,34)
     min(tuple8)

     9

[24] max (tuple8)

     87

     sum (tuple8)

     235
```

Tuples

Tuple Items - Data Types

Tuple items can be of any data type:

Example

String, int and boolean data types:

```
tuple1 = ("apple", "banana", "cherry")
tuple2 = (1, 5, 7, 9, 3)
tuple3 = (True, False, False)
```

```
# ('apple', 'banana', 'cherry')
  (1, 5, 7, 9, 3)
  (True, False, False)
```

A tuple with strings, integers and boolean values:

```
tuple1 = ("abc", 34, True, 40, "male")
```

```
# ("abc", 34, True, 40, "male")
```

Tuples

Access Tuple Items

You can access tuple items by referring to the index number, inside square brackets:

```
thistuple = ("apple", "banana", "cherry")
print(thistuple[1])                    # banana
```

Negative Indexing

Negative indexing means start from the end.

-1 refers to the last item, -2 refers to the second last item etc.

```
thistuple = ("apple", "banana", "cherry")
print(thistuple[-1])                   #cherry
```

Tuples

Range of Indexes

You can specify a range of indexes by specifying where to start and where to end the range.
When specifying a range, the return value will be a new tuple with the specified items.

Return the third, fourth, and fifth item:

```
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(thistuple[2:5])                    #('cherry', 'orange', 'kiwi')
```

```
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(thistuple[:4])                     #('apple', 'banana', 'cherry', 'orange')
```

Tuples

Check if Item Exists

To determine if a specified item is present in a tuple use the in keyword:

```
thistuple = ("apple", "banana", "cherry")
if "apple" in thistuple:
  print("Yes, 'apple' is in the fruits tuple")          #Yes, 'apple' is in the fruits tuple
```

Tuples

Change Tuple Values

Once a tuple is created, you cannot change its values.

Tuples are unchangeable, or immutable as it also is called.

But there is a workaround. You can convert the tuple into a list, change the list, and convert the list back into a tuple.

Convert the tuple into a list to be able to change it:

```
x = ("apple", "banana", "cherry")
y = list(x)
y[1] = "kiwi"
x = tuple(y)

print(x)                    #("apple", "kiwi", "cherry")
```

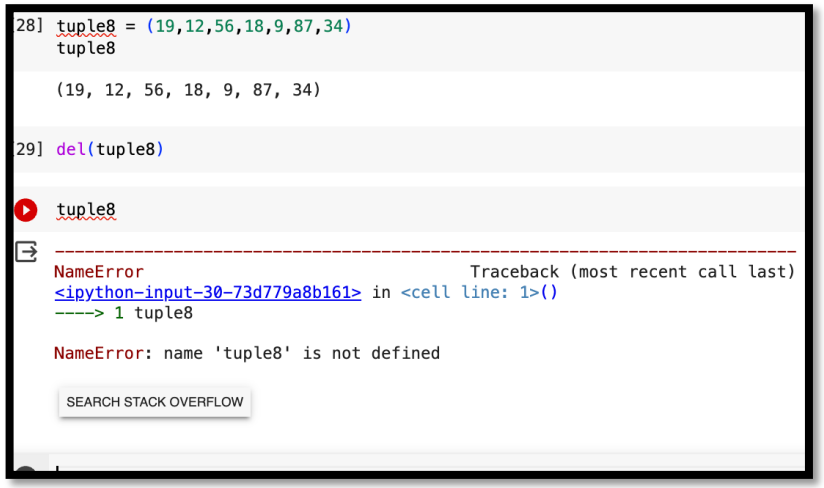
Tuples

Remove Items

Note: You cannot remove items in a tuple.

Tuples are **unchangeable**, so you cannot remove items from it, but you can use the same workaround as we used for changing and adding tuple items:

```
thistuple = ("apple", "banana", "cherry")
y = list(thistuple)
y.remove("apple")
thistuple = tuple(y)
print(thistuple)                    #("banana", "cherry")
```



Or you can delete the tuple completely:

The del keyword can delete the tuple completely:

```
thistuple = ("apple", "banana", "cherry")
del thistuple
print(thistuple) #this will raise an error because the tuple no longer exists
```

Tuples

TUPLE ASSIGNMENT

Assignment of tuple is a useful feature in Python. It allows a tuple of variables on the left side of the assignment operator to be assigned respective values from a tuple on the right side. The number of variables on the left should be the same as the number of elements in the tuple.

```
[11] (num3,num4) = (10+5,20+5)

[12] print(num3)

     15

[13] print(num4)

     25
```

The first element 10 is assigned to num1 and *the second element 20 is assigned to num2*

```
[1] (num1,num2) = (10,20)

[2] print(num1)

    10

[3] print(num2)

    20

    record = ( "Pooja",40,"CS")

[5] (name,rollNo,subject) = record

[7] name

    'Pooja'

[8] subject

    'CS'

[9] (a,b,c,d) = (5,6,8)

    ---------------------------------------------------------------------------
    ValueError                                Traceback (most recent call last)
    <ipython-input-9-1e8a30bde87f> in <cell line: 1>()
    ----> 1 (a,b,c,d) = (5,6,8)

    ValueError: not enough values to unpack (expected 4, got 3)
```

If there is an expression on the right side then first that expression is evaluated and finally, the result is assigned to the tuple.

Tuples

NESTED TUPLE

A tuple inside another tuple is called a nested tuple. In the program 10-1, roll number, name and marks (in percentage) of students are saved in a tuple. To store details of many such students we can create a nested tuple.

Q1. This is a program to create a nested tuple to store the roll number, name and marks of students

```
st=((101,"Aman",98),(102,"Geet",95),(103,"Sahil",87),(104,"Pawan",79))
print("S_No"," Roll_No"," Name"," Marks")
for i in range(0,len(st)):
 print((i+1),'\t',st[i][0],'\t',st[i][1],'\t',st[i][2])
```

```
S_No  Roll_No  Name  Marks
1        101     Aman    98
2        102     Geet    95
3        103     Sahil   87
4        104     Pawan   79
```

Tuples

Problems

1. *Write a program to swap two numbers without using a temporary variable.*
2. *Write a program to compute the area and circumference of a circle using a function.*
3. *Write a program to input n numbers from the user. Store these numbers in a tuple. Print the maximum and minimum number from this tuple.*

Mapping and Dictionaries

- **Unordered and Mutable data type**
- **Dictionaries**
- Lists are ordered sets of objects, whereas **dictionaries are unordered sets**.
- Dictionary is created by using **curly brackets**. i,e. {}
- Dictionaries **are accessed via keys** and not via their position.
- A dictionary is an associative array (also known as **hashes)**. Any key of the dictionary is
- associated (or mapped) to a value.
- The values of a dictionary can be any Python data type. So dictionaries are **unordered key-value-pairs (**The association of a key and a value is called a key-value pair)
- Dictionaries don't support the sequence operation of the sequence data types like strings, tuples and lists.