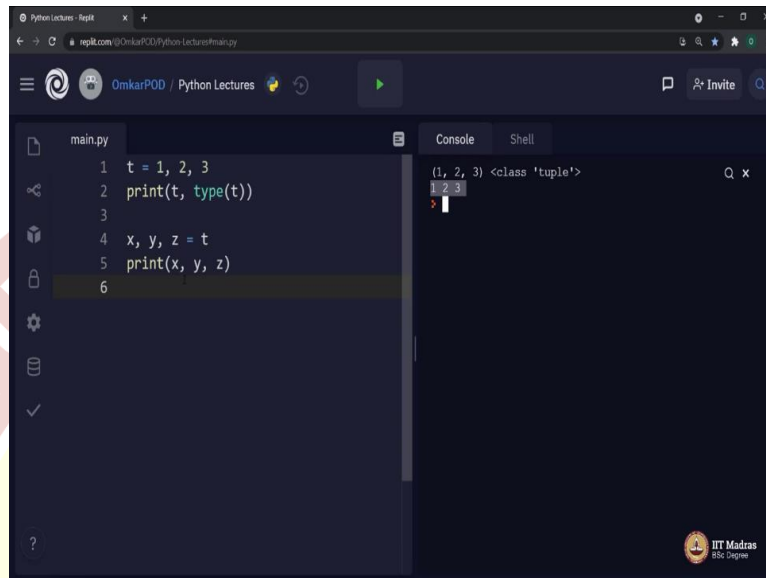




IIT Madras
ONLINE DEGREE

Programing in Python
Professor. Sudarshan Iyengar
Department of Computer Science and Engineering
Indian Institute of Technology Ropar
More on Tuples

(Refer Slide Time: 00:16)



```
Python Lecture - Repl  
replit.com/@OmkarPOD/Python-Lecture/main.py  
OmkarPOD / Python Lectures  
main.py  
1 t = 1, 2, 3  
2 print(t, type(t))  
3  
4 x, y, z = t  
5 print(x, y, z)  
6  
Console  
(1, 2, 3) <class 'tuple'>  
1 2 3  
>
```

Hello Python students. In previous lecture, we discussed about some advanced concepts related to list. We will continue on those same lines and in this lecture, we will discuss some concepts related to tuples.

Although, lists and tuples are very similar, but they have one major difference related to mutability. I hope, you remember list is a mutable, whereas, tuples are immutable, as tuples are immutable we cannot modify the values inside tuple once the tuple is created. Tuple may not allow us to modify its values but still we can perform various operations on tuples. We can access elements in the tuple using their index.

We can perform slicing operation on tuples similar like lists or strings. We can iterate over tuples. Tuple also provides methods like list except methods like append, remove, insert, pop for obvious reason which is correct, tuple is immutable. As we have seen earlier most of the times, we use list in our python programs and that is what most of the programmers do, they hardly use tuples anywhere while writing a Python code.

So, now you must be wondering, if they are not going to use tuples while writing Python programs, then why we are even studying it and the reason is, we may not use tuples directly in our python programs, but still python programming language use tuples extensively for

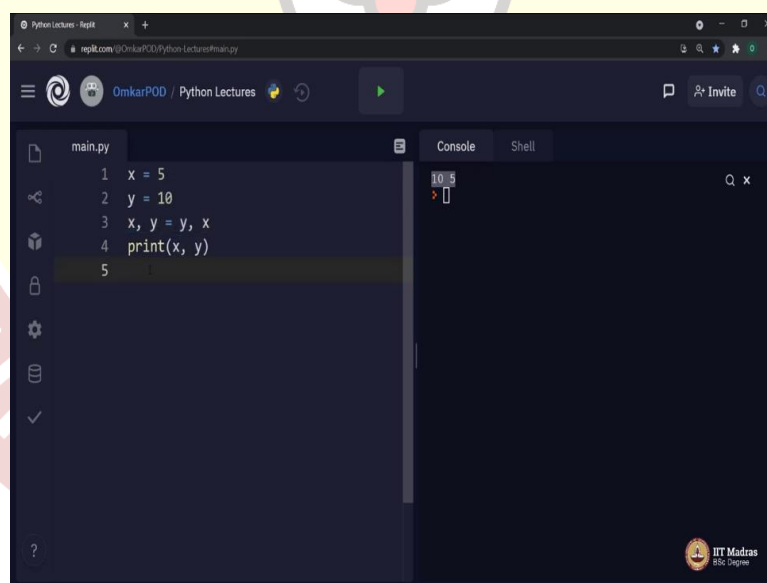
various different operations and the most common application of tuple is packing and unpacking.

Let us see some examples of python programs where python language use tuple for packing and unpacking. Let us look at this python code t is equal to 1, 2 3, print t and then type of t. On contrary, we are doing x, y, z is equal to t and then printing x, y, and z. Let us first execute this code and then we will see how tuples are involved in this particular code.

As you can see, while writing this particular code we simply wrote these three numbers separated by comma. We did not create a tuple of these values. We all know, tuples are created using these round brackets, which are not there in this python code, but still computer converted these three values into a tuple and initialize this tuple to variable t.

This is the place where we are packing these three values into a single entity called tuple. On the other hand, when we say x, y, z is equal to t we are unpacking this tuple into three independent integer values because of that this second print statement is printing three independent integers. This is the place where unpacking is happening.

(Refer Slide Time: 04:25)

A screenshot of a web-based Python REPL interface. The left pane shows a file named 'main.py' with the following code:

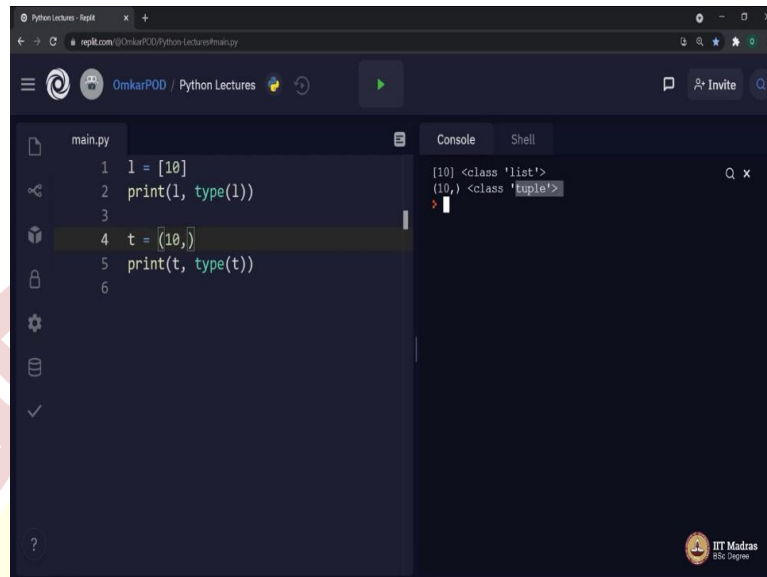
```
1 x = 5
2 y = 10
3 x, y = y, x
4 print(x, y)
5
```

The right pane is split into 'Console' and 'Shell' tabs. The 'Console' tab shows the output '10 5' on the first line and a prompt '>' on the second line. The 'Shell' tab is empty. The interface includes a top bar with 'OmkarPOD / Python Lectures' and a bottom right corner with the 'IIT Madras BSc Degree' logo.

Let us look at one more similar example, x is equal to 5, y is equal to 10 and then x, y is equal to y, x. We have seen this kind of expression in earlier weeks and if you remember, it simply swaps the values of x, y with each other. The output is 10, 5, instead of 5, 10. This is not a new concept, but one part which we did not noticed earlier is when we say x, y is equal to y, x python is actually packing these two variables, y and x into a tuple and then unpacking it on

the left-hand side. Before closing this lecture, let us look at few very interesting examples using tuple.

(Refer Slide Time: 05:21)

A screenshot of a Python REPL interface. The left pane shows a file named 'main.py' with the following code:

```
1 l = [10]
2 print(l, type(l))
3
4 t = (10,)
5 print(t, type(t))
6
```

The right pane shows the console output:

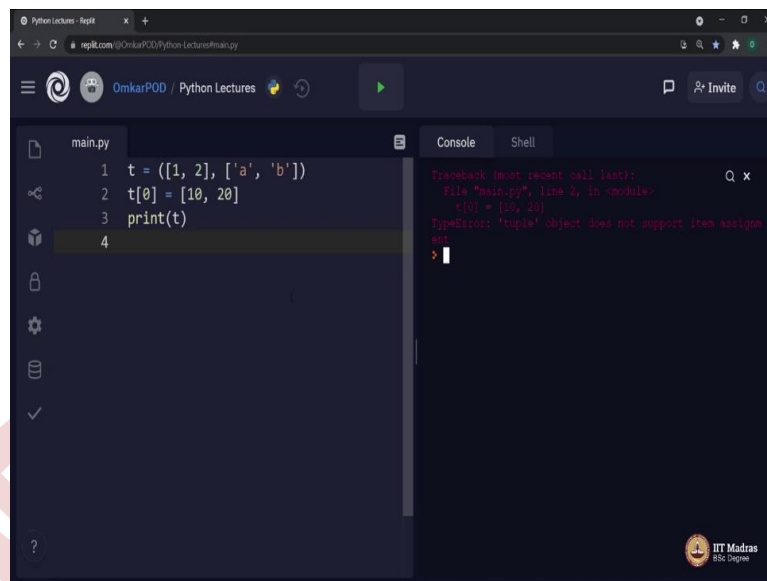
```
[10] <class 'list'>
(10,) <class 'tuple'>
```

The interface includes a top bar with 'OmkarPOD / Python Lectures' and a bottom right corner with the 'IIT Madras' logo.

Let us look at this code `l` is equal to a list with one element inside `print l` type of `l`, `t` is equal to round bracket one element inside it `print t`, type `t`. Can you predict, what will be the output of this code? Let us execute and see. Is that output, which you predicted or is it different. With first print statement, we are getting a list with one element and type is list which is alright, but in second line, it says 10 as a single value instead of a tuple and type is integer and that is the interesting part of tuple.

Whenever, you use these brackets to create a tuple and you add only one element inside it, python consider it as a single value instead of a tuple. Therefore, in order to create a tuple with a single element inside we have to add one, after this particular value. Let us try the same code again. Now we got the value inside round brackets and the type is tuple. Let us look at one more interesting example.

(Refer Slide Time: 06:57)



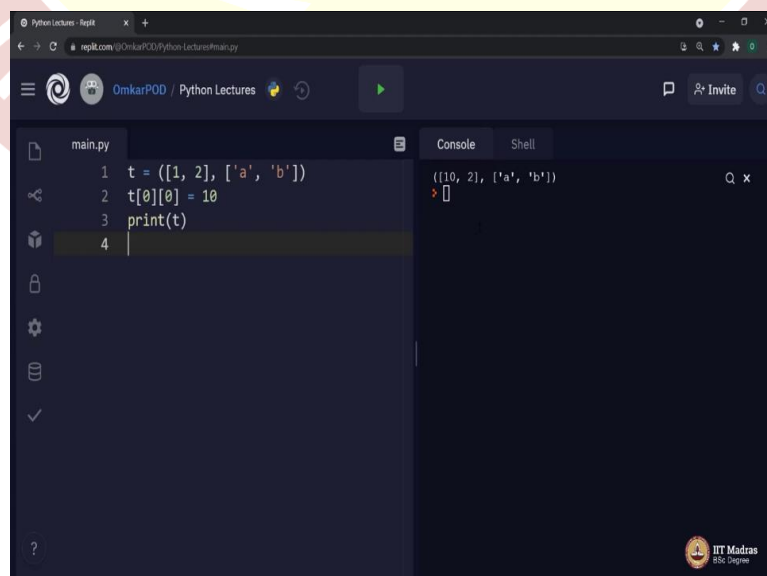
```
main.py
1 t = ([1, 2], ['a', 'b'])
2 t[0] = [10, 20]
3 print(t)
4
```

```
Traceback (most recent call last):
  File "main.py", line 2, in <module>
    t[0] = [10, 20]
TypeError: 'tuple' object does not support item assignment
```

Let us look at this particular tuple. It is a nested tuple with lists. T is a tuple variable, but the values inside tuple are two different lists. Before we execute this code, can you tell me whether python will allow us to do something like this? Let us try. As you can see yes, it is possible. We can create a tuple with list as a value inside it.

Now let us play with this tuple. T of 0 is equal to a new list, which is 10, 20, which means we are trying to replace this list 1, 2 with 10, 20. Let us execute. As expected we got an error. It says tuple object does not support item assignment. We are getting this error because tuples are immutable. That is fine, which means we cannot modify values of tuples, but can we modify a value of a list which is inside a tuple? Let us try that.

(Refer Slide Time: 08:15)

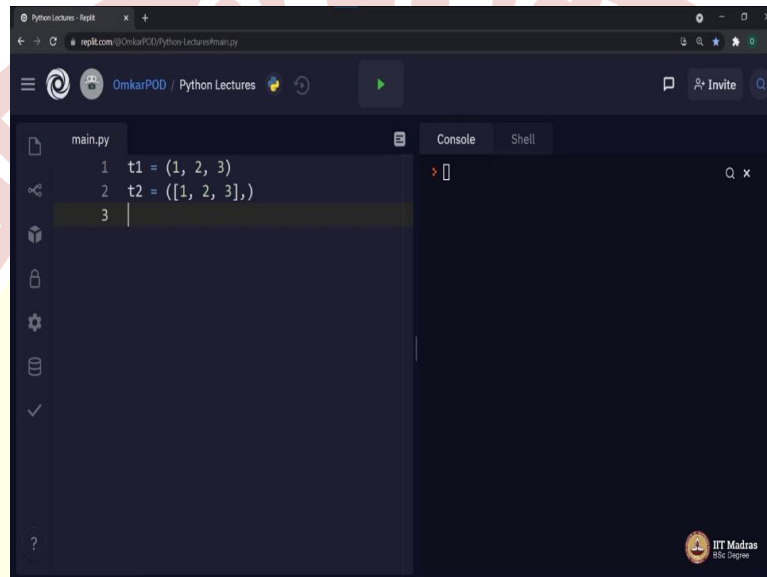


```
main.py
1 t = ([1, 2], ['a', 'b'])
2 t[0][0] = 10
3 print(t)
4
```

```
((10, 2), ['a', 'b'])
```

T 0, 0 is equal to 10. T 0 is this list and again 0 is this element 1, we are trying to replace this 1 with 10. Let us execute it. It worked, which means tuples immutability principle does not stop us from modifying the value of a list which is inside a tuple. Let me repeat, we cannot modify tuple values, but if a value inside a tuple is a list, then we can modify the values inside that list. In Python programming language, this particular principle is referred as hashable. Let me elaborate more on this particular new term hashable.

(Refer Slide Time: 09:13)

A screenshot of a web-based Python REPL interface. The left pane shows a file named 'main.py' with the following code:

```
1 t1 = (1, 2, 3)
2 t2 = ([1, 2, 3],)
3
```

The right pane is split into 'Console' and 'Shell' tabs, both of which are currently empty. The interface includes a search bar, a file explorer on the left, and a status bar at the bottom right indicating 'JIT Madras BSc Degree'.

Look at these two tuples, t1 and t2. t1 has three values directly inside a tuple whereas in t2 a value inside a tuple is a list and inside a list, we have three values. As we know, both are tuples therefore both are immutable, but the first tuple, which is t1 is hashable as well, whereas, the second tuple t2 is not hashable.

The term hashable is related to hashing. So, we will not go into those details instead, I will explain this term in simple words. If the values inside tuple are also immutable then the tuple is considered as hashable, whereas, if values inside tuple are mutable then tuple is referred as non-hashable. Remember this term hashable. We are going to refer this again in our next lecture, which is related to dictionaries. Thank you for watching this lecture. Happy learning.