Tuple in Python

What is Tuple in Python?

- In Python programming Language Tuple is an ordered sequence of items and it is similar to a list.
- The only difference is that tuples are immutable.
- Tuples are once created cannot be modified.
- They are used to write-protect data and are usually faster than list as it cannot change dynamically.
- Tuple is defined within parentheses () where items are separated by commas.

  Syntax:-

  ```
  tuple_1=(val1,val2,val3…….)
  ```

- Since, tuples are quite similar to lists, both of them are used in similar situations as well

Advantages of Tuple over List

- Since tuple are immutable, iterating through tuple is faster than with list. So there is a slight performance boost.
- Tuples that contain immutable elements can be used as key for a dictionary. With list, this is not possible.
- If you have data that doesn't change, implementing it as tuple will guarantee that it remains write-protected.

Creating a Tuple

- A tuple is created by placing all the items (elements) inside a parentheses (), separated by comma.
- The parentheses are optional but is a good practice to write it.
- A tuple can have any number of items and they may be of different types (integer, float, list, string etc.).
- Create Empty tuple →

```
my_tuple = () print(my_tuple)
```

- Tuple having integers →

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

- Tuple with mixed data types →

```
my_tuple = (1, "Hello", 3.4) print(my_tuple)
```

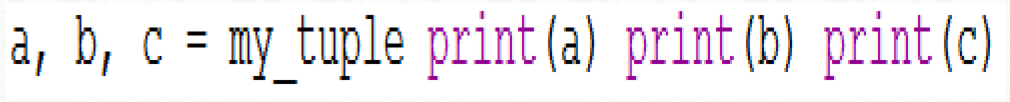
- Nested tuple →

```
my_tuple = ("mouse", [8, 4, 6], (1, 2, 3)) print(my_tuple)
```

- Tuple can be created without parentheses also called tuple packing

```
my_tuple = 3, 4.6, "dog" print(my_tuple)
```

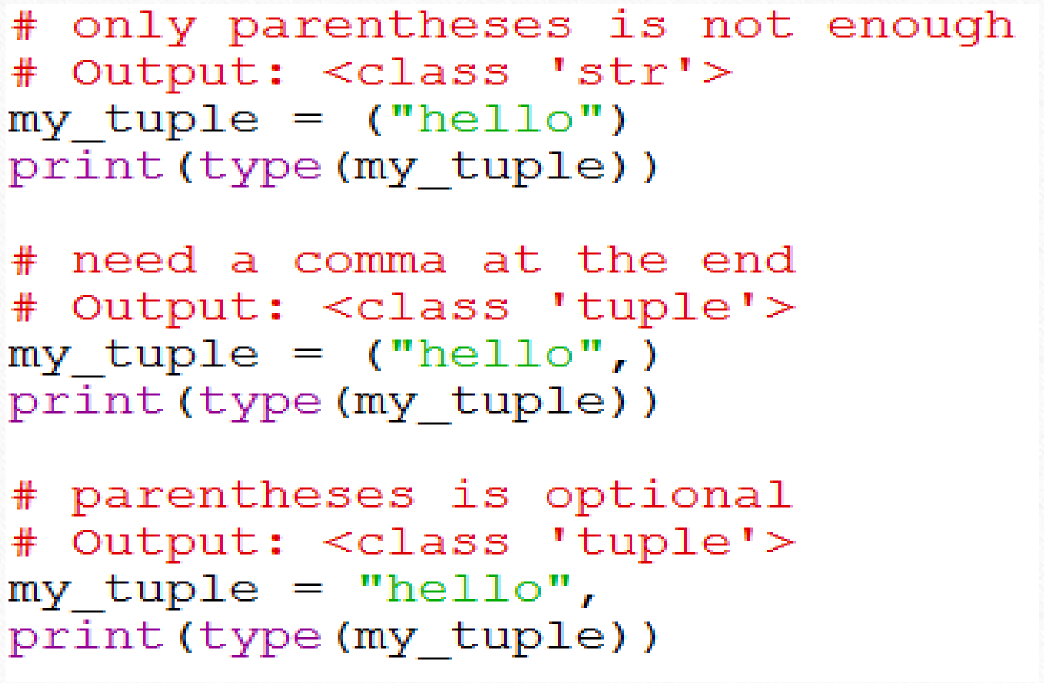
- Like Tuple packing ,tuple unpacking also possible.



```
a, b, c = my_tuple print(a) print(b) print(c)
```

- Creating a tuple with one element is Not to be trusted.

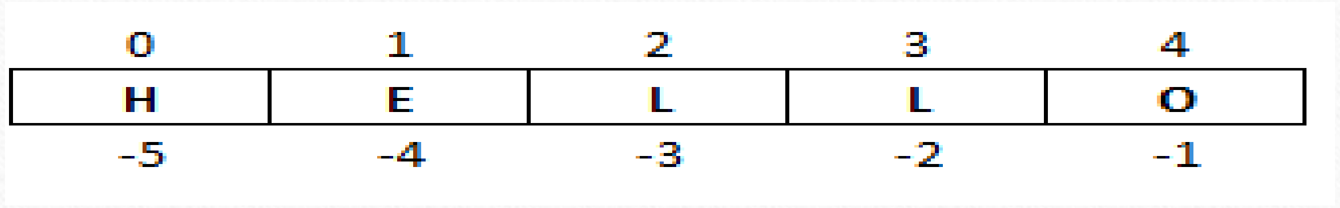
- Having one element within parentheses is not enough. We will need a trailing comma to indicate that it is in fact a tuple.



```
# only parentheses is not enough
# Output: <class 'str'>
my_tuple = ("hello")
print(type(my_tuple))

# need a comma at the end
# Output: <class 'tuple'>
my_tuple = ("hello",)
print(type(my_tuple))

# parentheses is optional
# Output: <class 'tuple'>
my_tuple = "hello",
print(type(my_tuple))
```

Accessing Elements in a Tuple

- Accessing elements in a tuple two ways in python programing language.
- We can access elements by using index operator **[]** and index value must be an **integer**
- We cannot use **float** or **other types** ,this will leads to rise an error call **typeError**
    - Positive Indexing
    - Negative Indexing
    - Slicing Operations



Positive Indexing:

- In positive indexing index value start with 0 and end with length-1
- If you try access elements more than **(>) length-1** it will rise an **indexError** or **out of the indexError**

Negative Indexing:

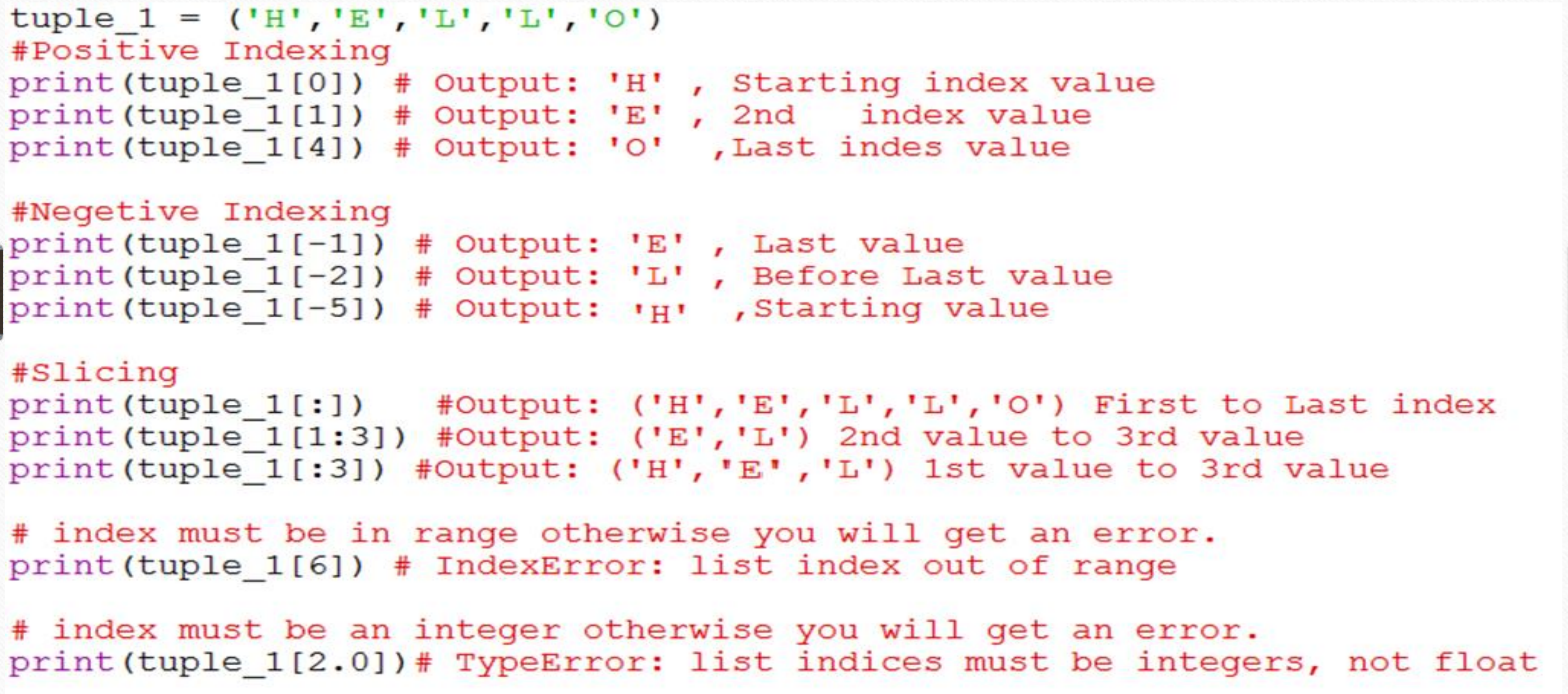
- Python allows negative indexing for its sequences.
- The index of -1 refers to the last item, -2 to the second last item and so on.

Slicing :

- We can access a range of items in a tuple by using the slicing operator - **colon ":"**.

Syntax : - **[start : end : step]**

Accessing Elements in tuple



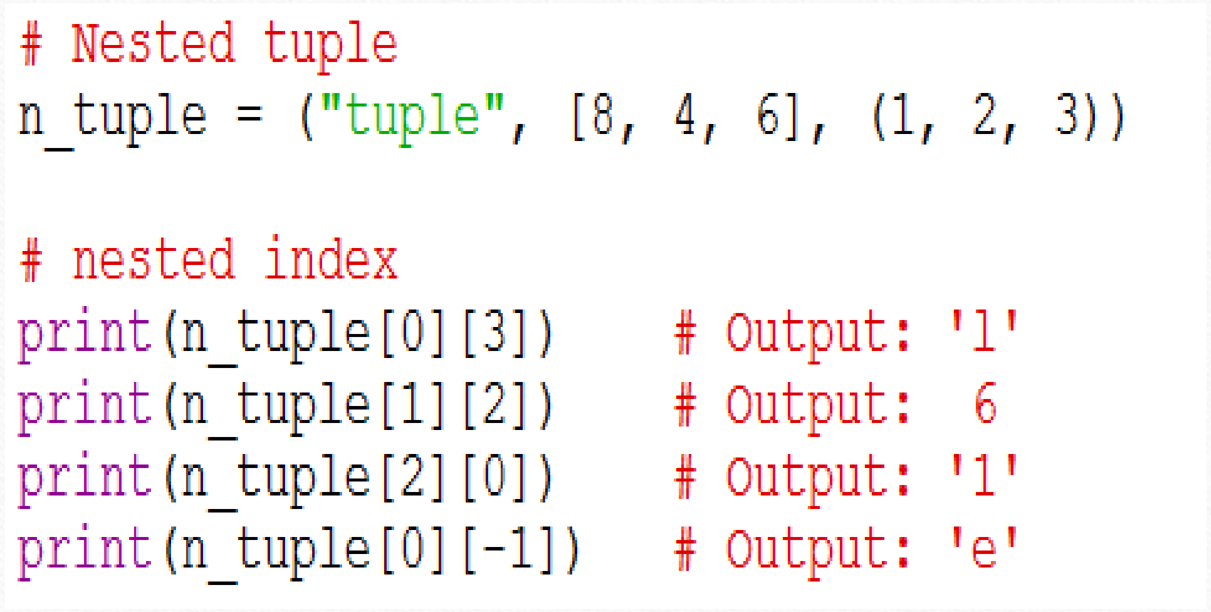
```
tuple_1 = ('H','E','L','L','O')
#Positive Indexing
print(tuple_1[0]) # Output: 'H' , Starting index value
print(tuple_1[1]) # Output: 'E' , 2nd   index value
print(tuple_1[4]) # Output: 'O'  ,Last indes value

#Negetive Indexing
print(tuple_1[-1]) # Output: 'E' , Last value
print(tuple_1[-2]) # Output: 'L' , Before Last value
print(tuple_1[-5]) # Output: 'H'  ,Starting value

#Slicing
print(tuple_1[:])   #Output: ('H','E','L','L','O') First to Last index
print(tuple_1[1:3]) #Output: ('E','L') 2nd value to 3rd value
print(tuple_1[:3]) #Output: ('H','E','L') 1st value to 3rd value

# index must be in range otherwise you will get an error.
print(tuple_1[6]) # IndexError: list index out of range

# index must be an integer otherwise you will get an error.
print(tuple_1[2.0])# TypeError: list indices must be integers, not float
```

Nested Indexing in tuple

- We can also access nested tuple using nested indexing .

Syntax:

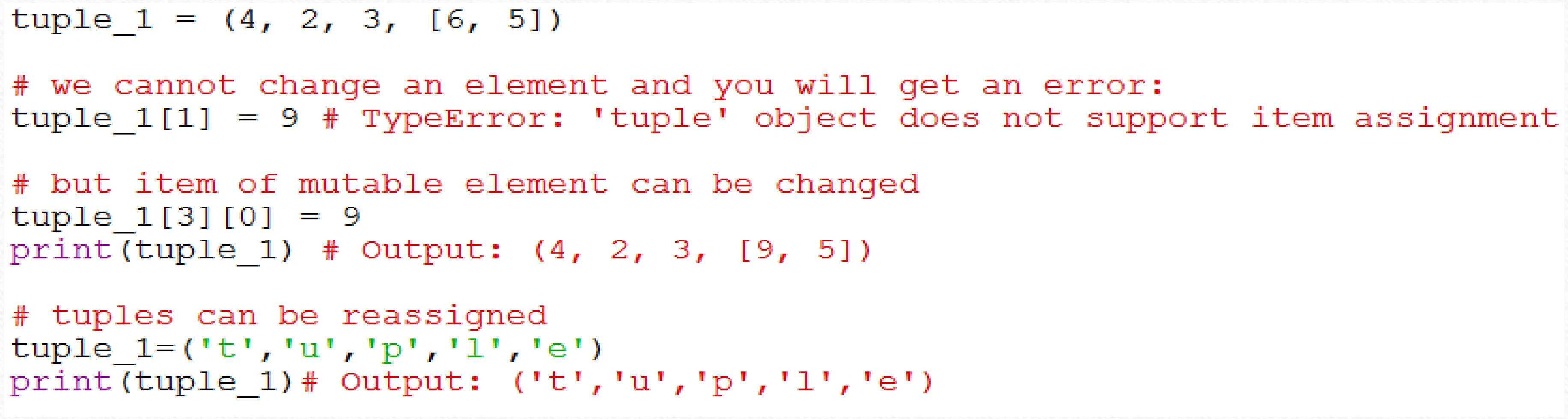
n_tuple[**tupleIndex**][**nestedTuple_index**]



```
# Nested tuple
n_tuple = ("tuple", [8, 4, 6], (1, 2, 3))

# nested index
print(n_tuple[0][3])    # Output: 'l'
print(n_tuple[1][2])    # Output:  6
print(n_tuple[2][0])    # Output: '1'
print(n_tuple[0][-1])   # Output: 'e'
```

How to Change a tuple ?

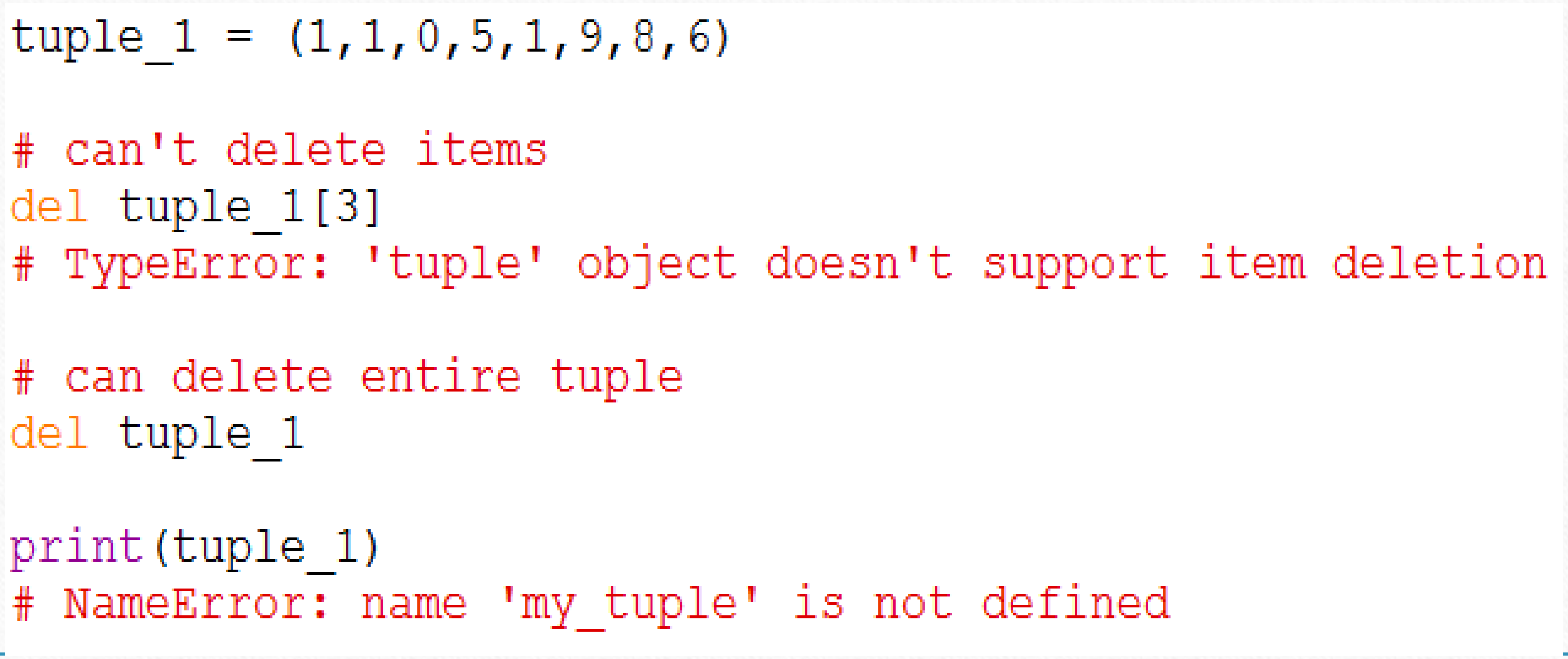
- Unlike lists tuples are immutable
- If you have data that doesn't change, implementing it as tuple will guarantee that it remains write-protected.
- This means that elements of a tuple cannot be changed once it has been assigned.
- But, if the element is itself a mutable datatype like list, its nested items can be changed.
- We can also assign a tuple to different values (reassignment).



```
tuple_1 = (4, 2, 3, [6, 5])

# we cannot change an element and you will get an error:
tuple_1[1] = 9 # TypeError: 'tuple' object does not support item assignment

# but item of mutable element can be changed
tuple_1[3][0] = 9
print(tuple_1) # Output: (4, 2, 3, [9, 5])

# tuples can be reassigned
tuple_1=('t','u','p','l','e')
print(tuple_1)# Output: ('t','u','p','l','e')
```

How to Delete a tuple ?

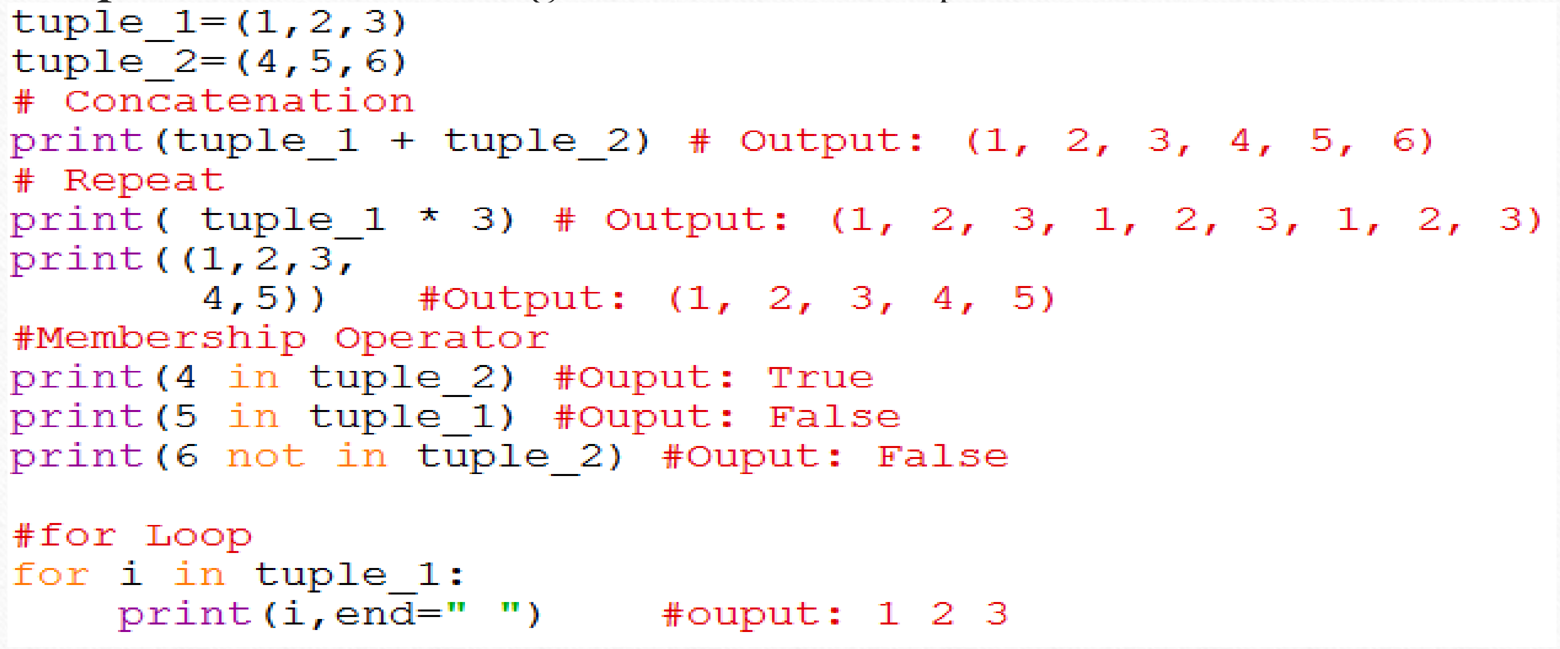
- As discussed in previous slide, we cannot change the elements in a tuple.
- That means we cannot delete or remove items from a tuple.
- But deleting entire tuple is possible by using the keyword del.



```python
tuple_1 = (1,1,0,5,1,9,8,6)

# can't delete items
del tuple_1[3]
# TypeError: 'tuple' object doesn't support item deletion

# can delete entire tuple
del tuple_1

print(tuple_1)
# NameError: name 'my_tuple' is not defined
```

Tuple Operations

- We can use **+** operator to combine two tuples. This is also called **concatenation**.
- We can also **repeat** the elements in a tuple for a given number of times using the ***** operator.
- We can test if an item exists in a tuple or not, using the keyword **in** or **not in (membership operator)**.
- Using a **for loop** we can iterate though each item in a tuple.



```
tuple_1=(1,2,3)
tuple_2=(4,5,6)
# Concatenation
print(tuple_1 + tuple_2) # Output: (1, 2, 3, 4, 5, 6)
# Repeat
print( tuple_1 * 3) # Output: (1, 2, 3, 1, 2, 3, 1, 2, 3)
print((1,2,3,
       4,5))    #Output: (1, 2, 3, 4, 5)
#Membership Operator
print(4 in tuple_2) #Ouput: True
print(5 in tuple_1) #Ouput: False
print(6 not in tuple_2) #Ouput: False

#for Loop
for i in tuple_1:
    print(i,end=" ")    #ouput: 1 2 3
```

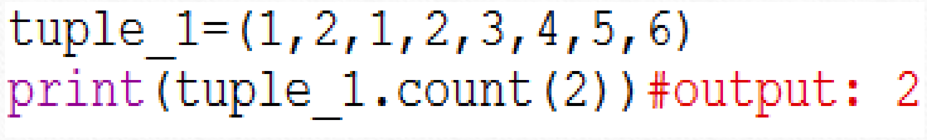
Built-in Functions & Methods in Python

- Methods that add items or remove items are not available with tuple. Only **count()** and **index()** methods are available.
- Built-in functions are commonly used with tuple to perform different tasks.

count(x)

- The count() method returns the number of occurrences of an element in a tuple

Syntax: - **tuple. count(element)**

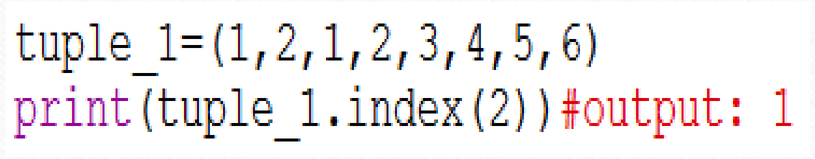


```
tuple_1=(1,2,1,2,3,4,5,6)
print(tuple_1.count(2))#output: 2
```

index(x)

- The index() method searches an element in a tuple and returns its index

Syntax:- **tuple.index(element)**

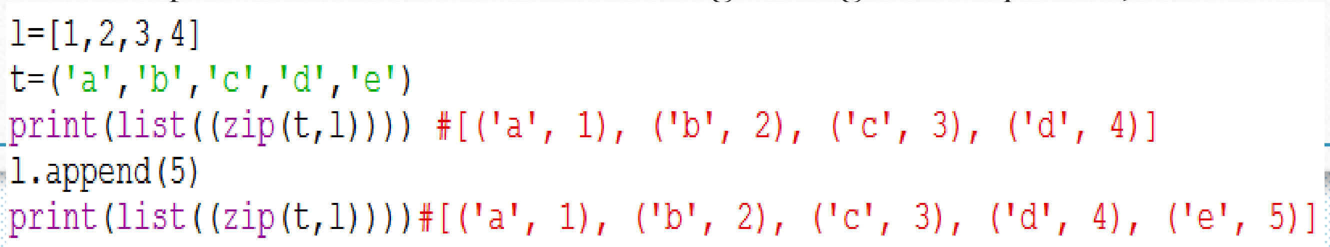


```
tuple_1=(1,2,1,2,3,4,5,6)
print(tuple_1.index(2))#output: 1
```

Zip()

- The zip() function takes two or more sequences and **zips** them into list of tuples

Syntax: - **list(zip(sequences))** {* list of tuples formed with smallest length of given sequence}

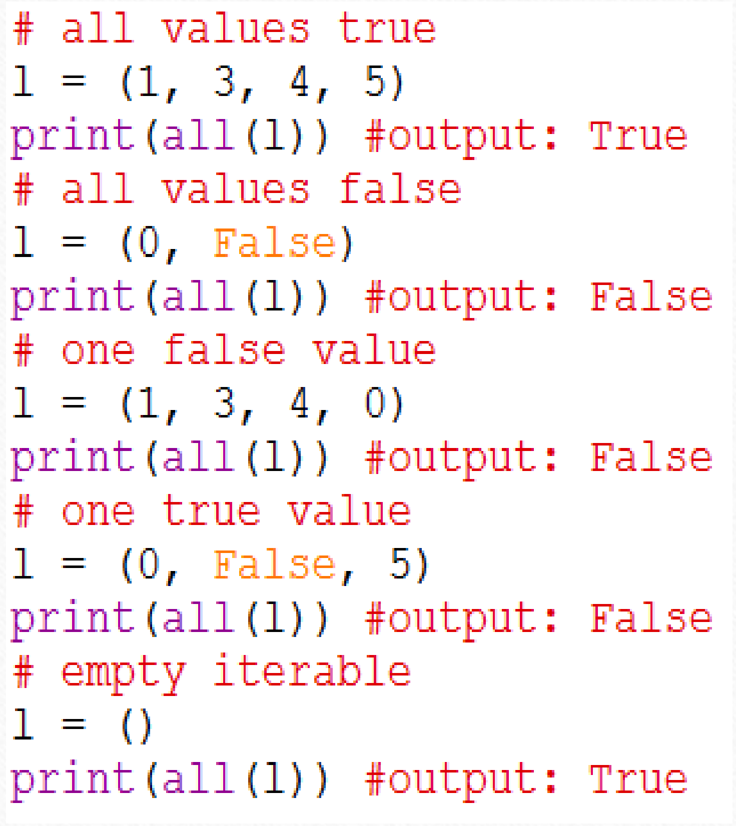


```
l=[1,2,3,4]
t=('a','b','c','d','e')
print(list((zip(t,l)))) #[('a', 1), ('b', 2), ('c', 3), ('d', 4)]
l.append(5)
print(list((zip(t,l))))#[('a', 1), ('b', 2), ('c', 3), ('d', 4), ('e', 5)]
```

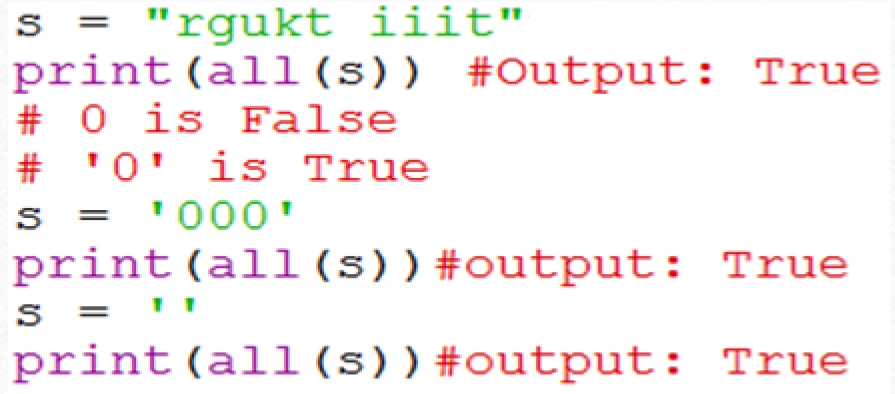
all()

- Return True if all elements of the tuple are true (or if the tuple is empty is true).

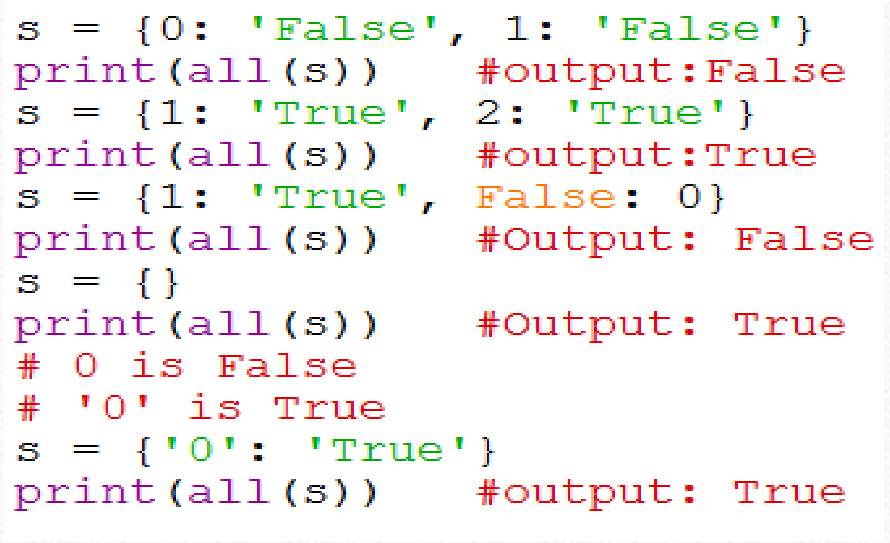
  Syntax:- **all(iterable)**



```
# all values true
l = (1, 3, 4, 5)
print(all(l)) #output: True
# all values false
l = (0, False)
print(all(l)) #output: False
# one false value
l = (1, 3, 4, 0)
print(all(l)) #output: False
# one true value
l = (0, False, 5)
print(all(l)) #output: False
# empty iterable
l = ()
print(all(l)) #output: True
```



```
s = "rgukt iiit"
print(all(s)) #Output: True
# 0 is False
# '0' is True
s = '000'
print(all(s))#output: True
s = ''
print(all(s))#output: True
```



```
s = {0: 'False', 1: 'False'}
print(all(s))    #output:False
s = {1: 'True', 2: 'True'}
print(all(s))    #output:True
s = {1: 'True', False: 0}
print(all(s))    #Output: False
s = {}
print(all(s))    #Output: True
# 0 is False
# '0' is True
s = {'0': 'True'}
print(all(s))    #output: True
```

any()

- Return True if any element of the tuple is true. If the tuple is empty, return False.

Syntax: - **any(iterable)**

enumerate()

- Return an enumerate object. It contains the index and value of all the items of tuple as pairs.

  Syntax:- **enumerate(iterable,start=0)**

len()

- Return the length (the number of items) in the tuple.

  Syntax:- **len(tuple)**

max() and **min()**

- Max Return the largest item in the tuple. Min Returns the smallest item in the tuple

  Syntax:- **max(iterable, *iterables[,key, default]) or max(arg1, arg2, *args[, key])**

sorted()

- Take elements in the tuple and return a new sorted list (does not sort the tuple itself).

  Syntax: - **sorted(iterable[, key][, reverse])**

sum()

- Retrun the sum of all elements in the tuple.

  Syntax: - **sum(iterable, start)**

tuple()

- Convert an iterable (list, string, set, dictionary) to a tuple.

  Syntax: - **tuple(iterable)**

Example Programs in Tuple

1. Write a program to swap two values using assignment?
2. Write a program using a function that returns the area and circumference of a circle whose radius is passed as an argument?
3. Write a program that has a nested tuple to store toppers details. Edit the details and reprint the details ?
4. Write a program that has a list of numbers (both positive as well as negative).Make a new tuple that has only positive values from this list?