SNS COLLEGE OF ENGINEERING
Kurumbapalayam (Po), Coimbatore – 641 107

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

19IT103 – COMPUTATIONAL THINKING AND PYTHON PROGRAMMING

- A readable, dynamic, pleasant, flexible, fast and powerful language

Session wise Agenda

- **session 1 - List (Operations, Slice, Methods)**
- **Session 2 - List (Loop, Mutability)**
- **Session 3 - List (Aliasing, Cloning, Parameters)**
- **Session 4 - Tuples (Assignment, as return value)**
- Session 5 - Dictionaries (operations and methods)
- Session 6 - Advance List processing, List Comprehension
- Session 7 - Simple Sort, Histogram
- Session 8 - Student Mark Statement
- Session 9 - Retail Bill preparation

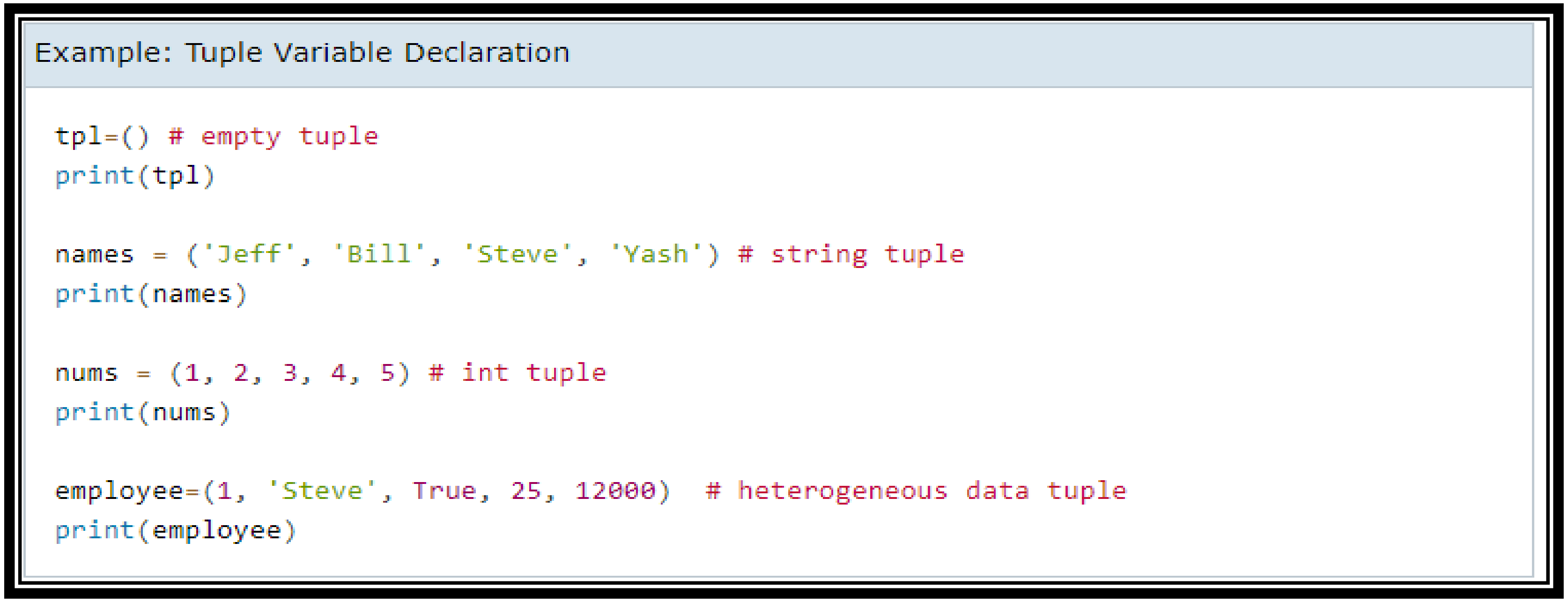
Recap

- Aliasing ☐ copying the List i.e. the memory will be same for both the List variables. If any changes made in one list will affect other.
- Cloning ☐ copying the List but the memory location is different. If any changes made in one list will not affect other.
- List as Parameter ☐ List is passed as parameter to a function i.e. as Call by Reference (Address). If any changes made in the list inside function the change will occur in the calling function also

Tuple

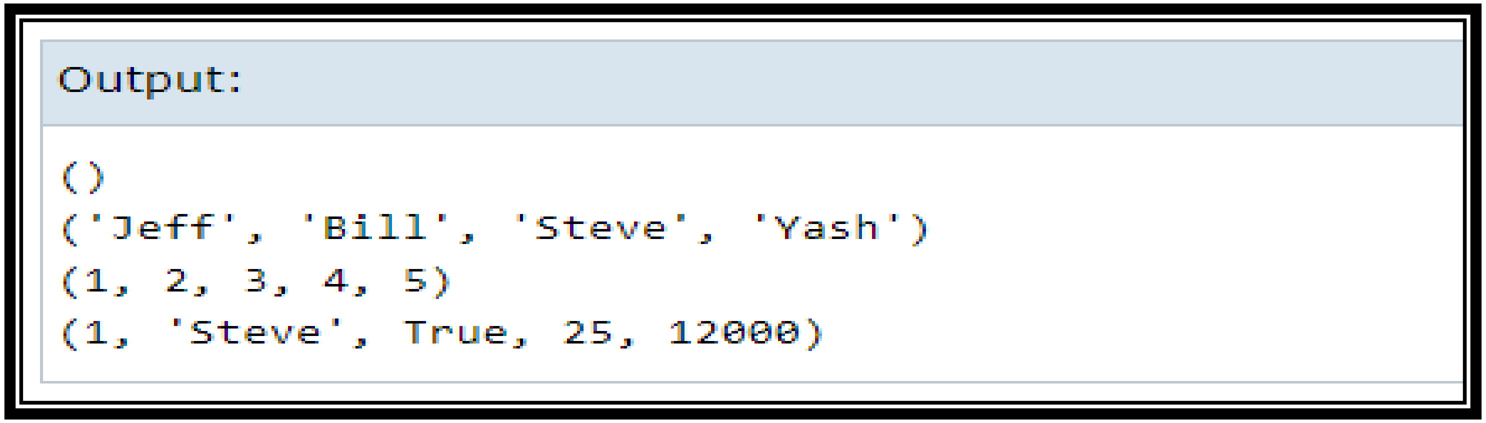
- A tuple in Python is similar to a list. The difference between the two is that we cannot change the elements of a tuple once it is assigned whereas we can change the elements of a list.
- Tuple is an immutable (unchangeable) collection of elements of different data types. It is an ordered collection, so it preserves the order of elements in which they were defined.
- Tuples are defined by enclosing elements in parentheses (), separated by a comma.

Contd..



Example: Tuple Variable Declaration

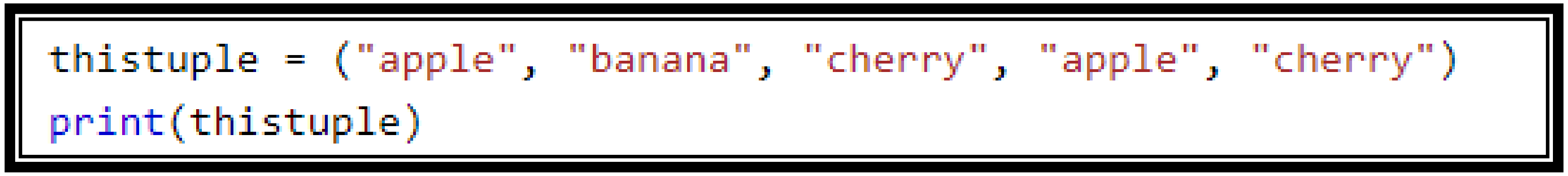
```
tpl=() # empty tuple
print(tpl)

names = ('Jeff', 'Bill', 'Steve', 'Yash') # string tuple
print(names)

nums = (1, 2, 3, 4, 5) # int tuple
print(nums)

employee=(1, 'Steve', True, 25, 12000)  # heterogeneous data tuple
print(employee)
```



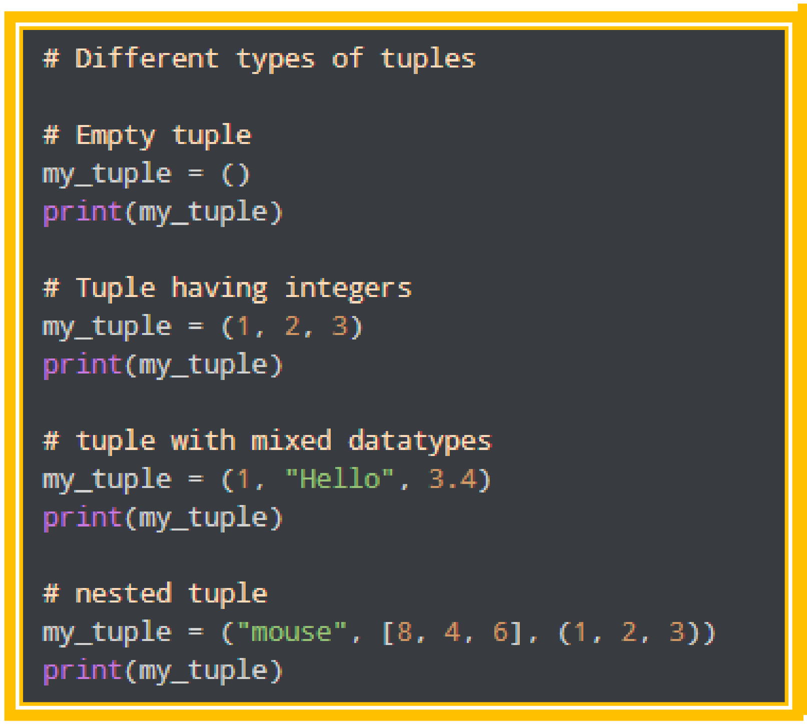
Output:

```
()
('Jeff', 'Bill', 'Steve', 'Yash')
(1, 2, 3, 4, 5)
(1, 'Steve', True, 25, 12000)
```

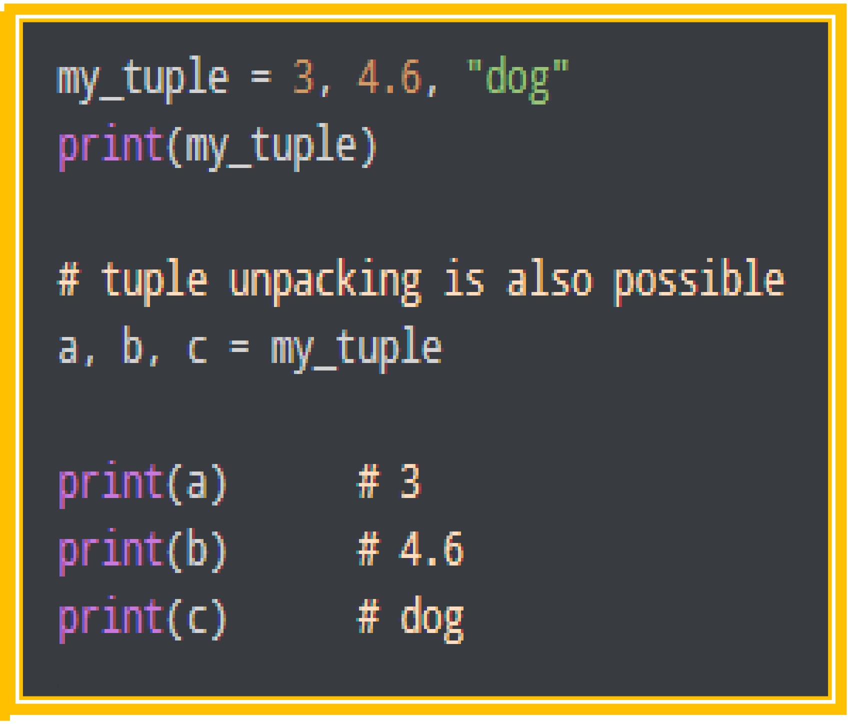
Contd..



```python
thistuple = ("apple", "banana", "cherry", "apple", "cherry")
print(thistuple)
```

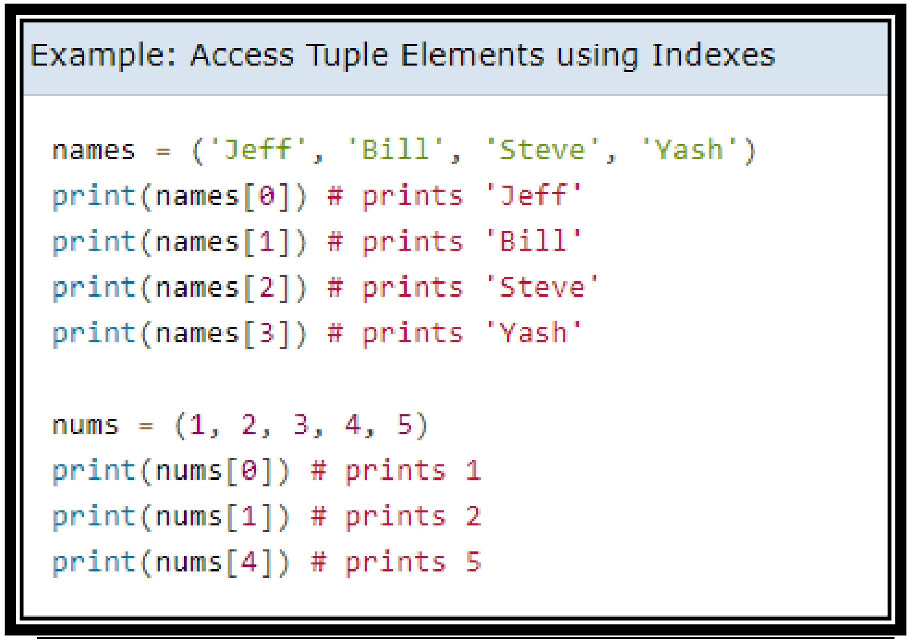


```python
# Different types of tuples

# Empty tuple
my_tuple = ()
print(my_tuple)

# Tuple having integers
my_tuple = (1, 2, 3)
print(my_tuple)

# tuple with mixed datatypes
my_tuple = (1, "Hello", 3.4)
print(my_tuple)

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



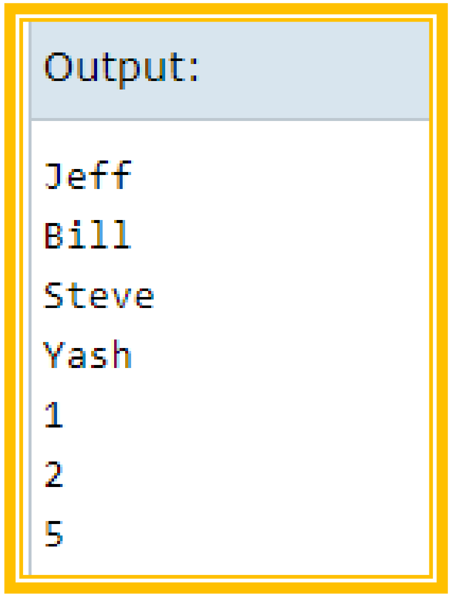
```python
my_tuple = 3, 4.6, "dog"
print(my_tuple)

# tuple unpacking is also possible
a, b, c = my_tuple

print(a)      # 3
print(b)      # 4.6
print(c)      # dog
```

Accessing Tuple Elements



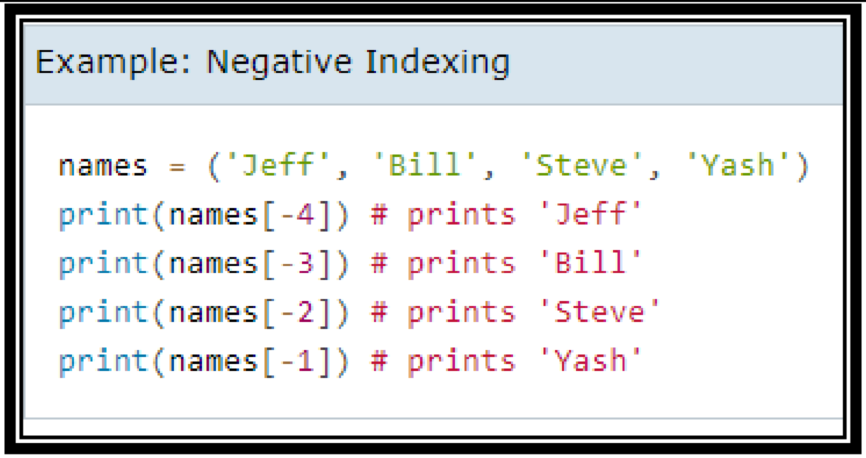
Example: Access Tuple Elements using Indexes

```
names = ('Jeff', 'Bill', 'Steve', 'Yash')
print(names[0]) # prints 'Jeff'
print(names[1]) # prints 'Bill'
print(names[2]) # prints 'Steve'
print(names[3]) # prints 'Yash'

nums = (1, 2, 3, 4, 5)
print(nums[0]) # prints 1
print(nums[1]) # prints 2
print(nums[4]) # prints 5
```



Output:

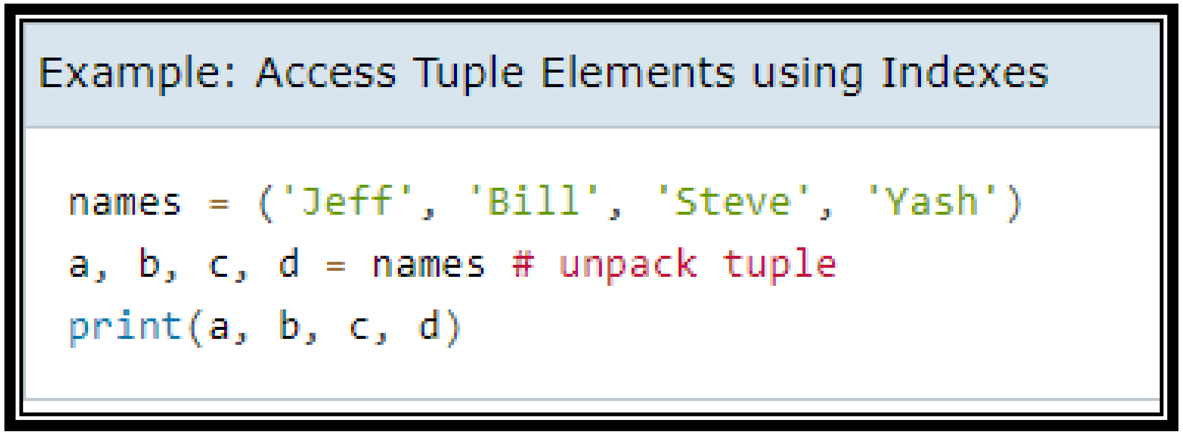
```
Jeff
Bill
Steve
Yash
1
2
5
```



Example: Negative Indexing

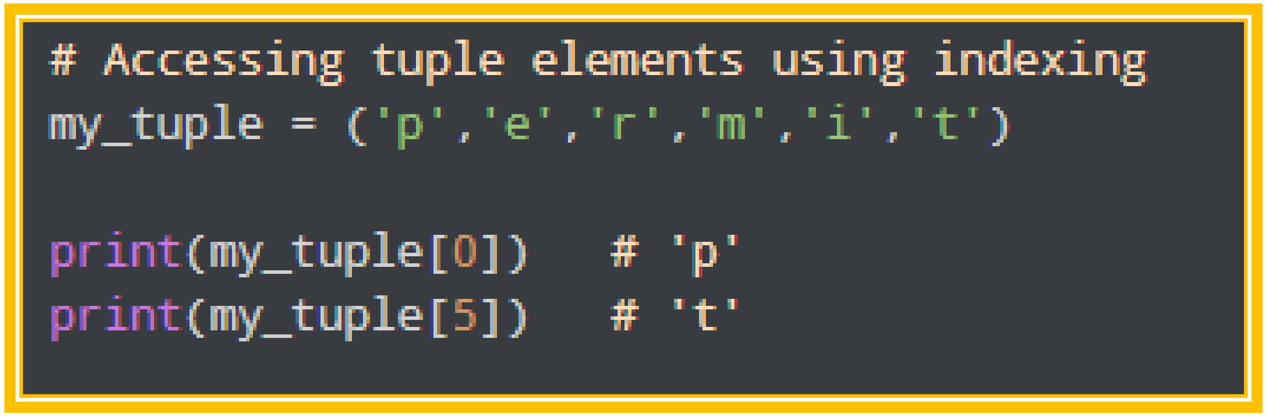
```
names = ('Jeff', 'Bill', 'Steve', 'Yash')
print(names[-4]) # prints 'Jeff'
print(names[-3]) # prints 'Bill'
print(names[-2]) # prints 'Steve'
print(names[-1]) # prints 'Yash'
```

Contd..



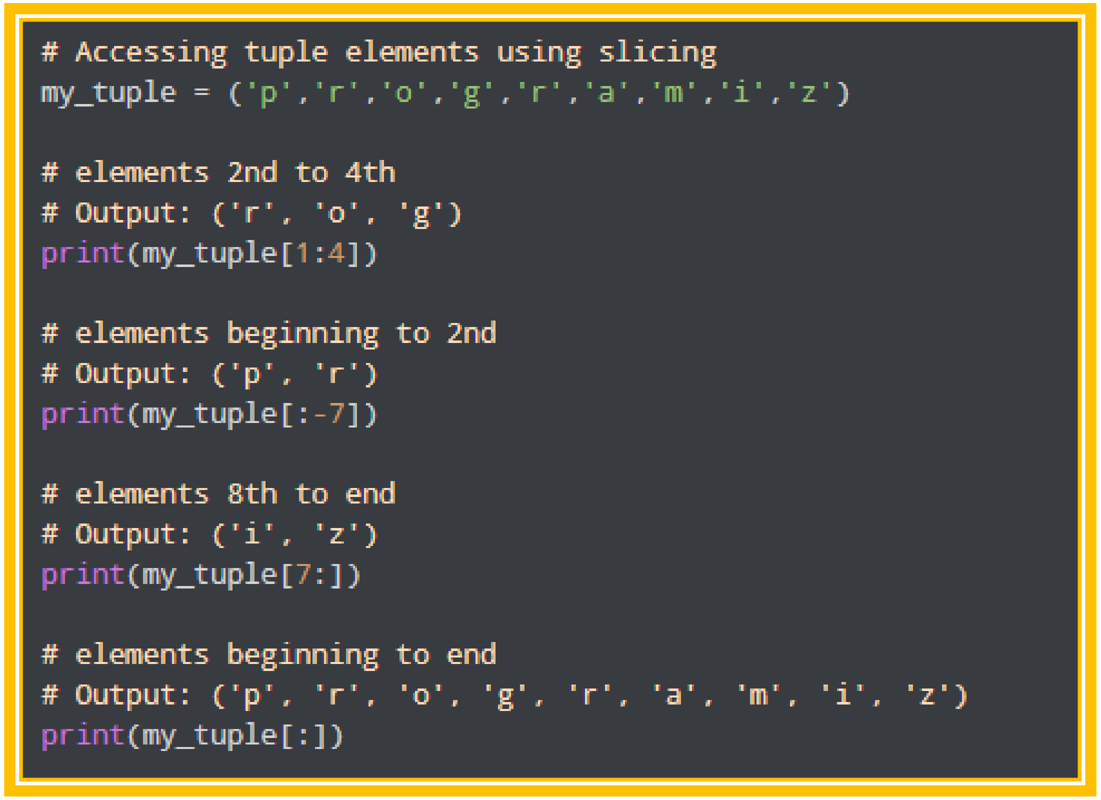
Example: Access Tuple Elements using Indexes

```
names = ('Jeff', 'Bill', 'Steve', 'Yash')
a, b, c, d = names # unpack tuple
print(a, b, c, d)
```

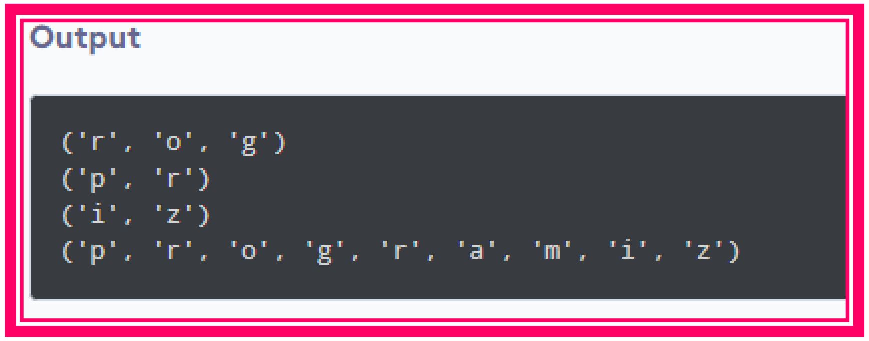


```
# Accessing tuple elements using indexing
my_tuple = ('p','e','r','m','i','t')

print(my_tuple[0])   # 'p'
print(my_tuple[5])   # 't'
```

Tuple Operations



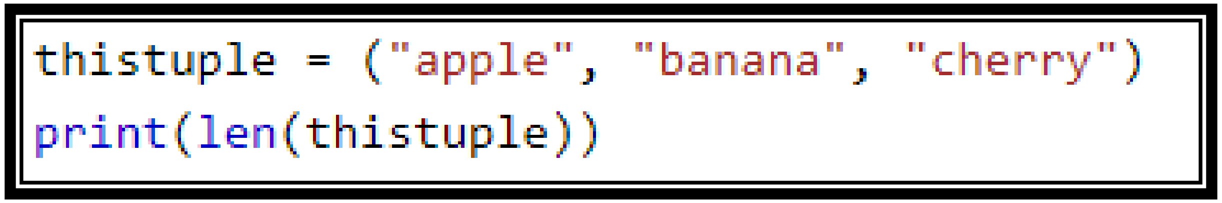
```python
# Accessing tuple elements using slicing
my_tuple = ('p','r','o','g','r','a','m','i','z')

# elements 2nd to 4th
# Output: ('r', 'o', 'g')
print(my_tuple[1:4])

# elements beginning to 2nd
# Output: ('p', 'r')
print(my_tuple[:-7])

# elements 8th to end
# Output: ('i', 'z')
print(my_tuple[7:])

# elements beginning to end
# Output: ('p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'z')
print(my_tuple[:])
```



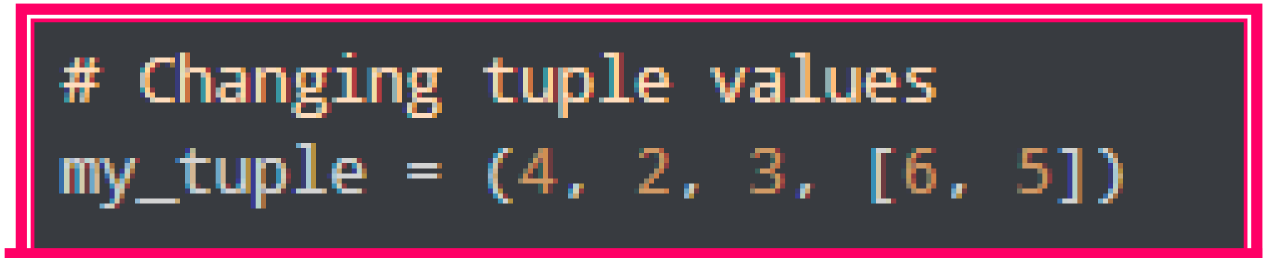
Output

```
('r', 'o', 'g')
('p', 'r')
('i', 'z')
('p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'z')
```

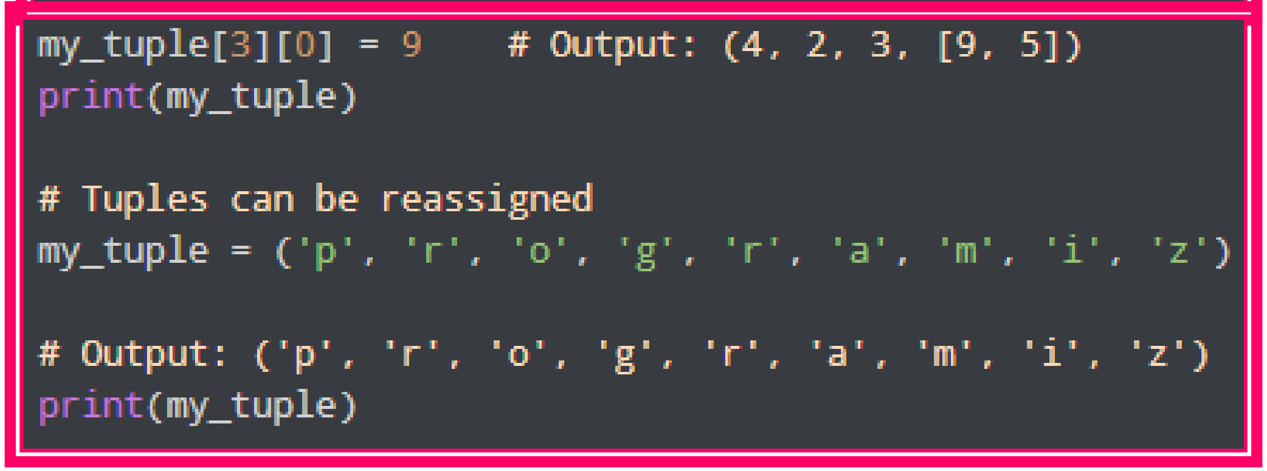
Contd..



```python
thistuple = ("apple", "banana", "cherry")
print(len(thistuple))
```

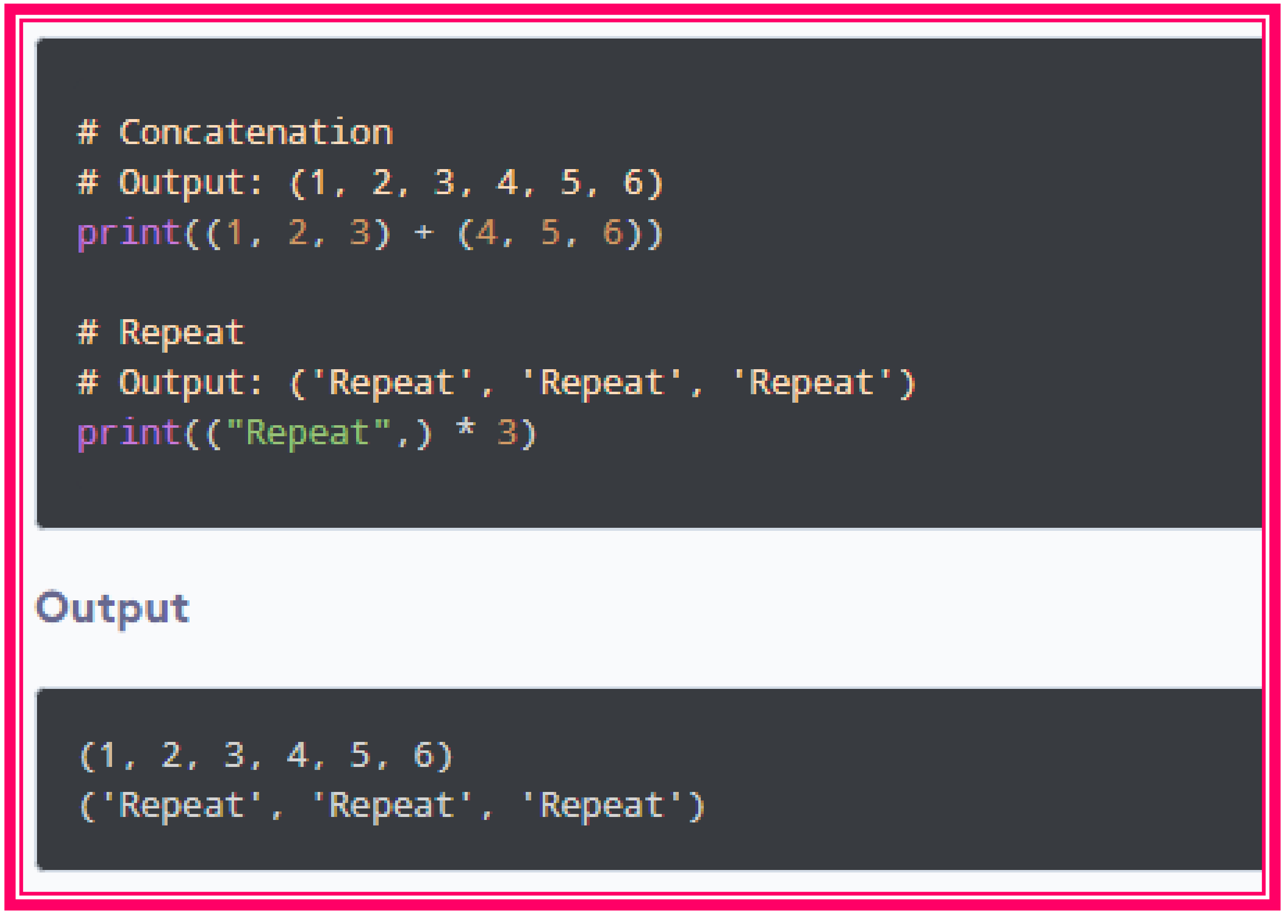


```python
# Changing tuple values
my_tuple = (4, 2, 3, [6, 5])
```



```python
my_tuple[3][0] = 9    # Output: (4, 2, 3, [9, 5])
print(my_tuple)

# Tuples can be reassigned
my_tuple = ('p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'z')

# Output: ('p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'z')
print(my_tuple)
```

Contd..

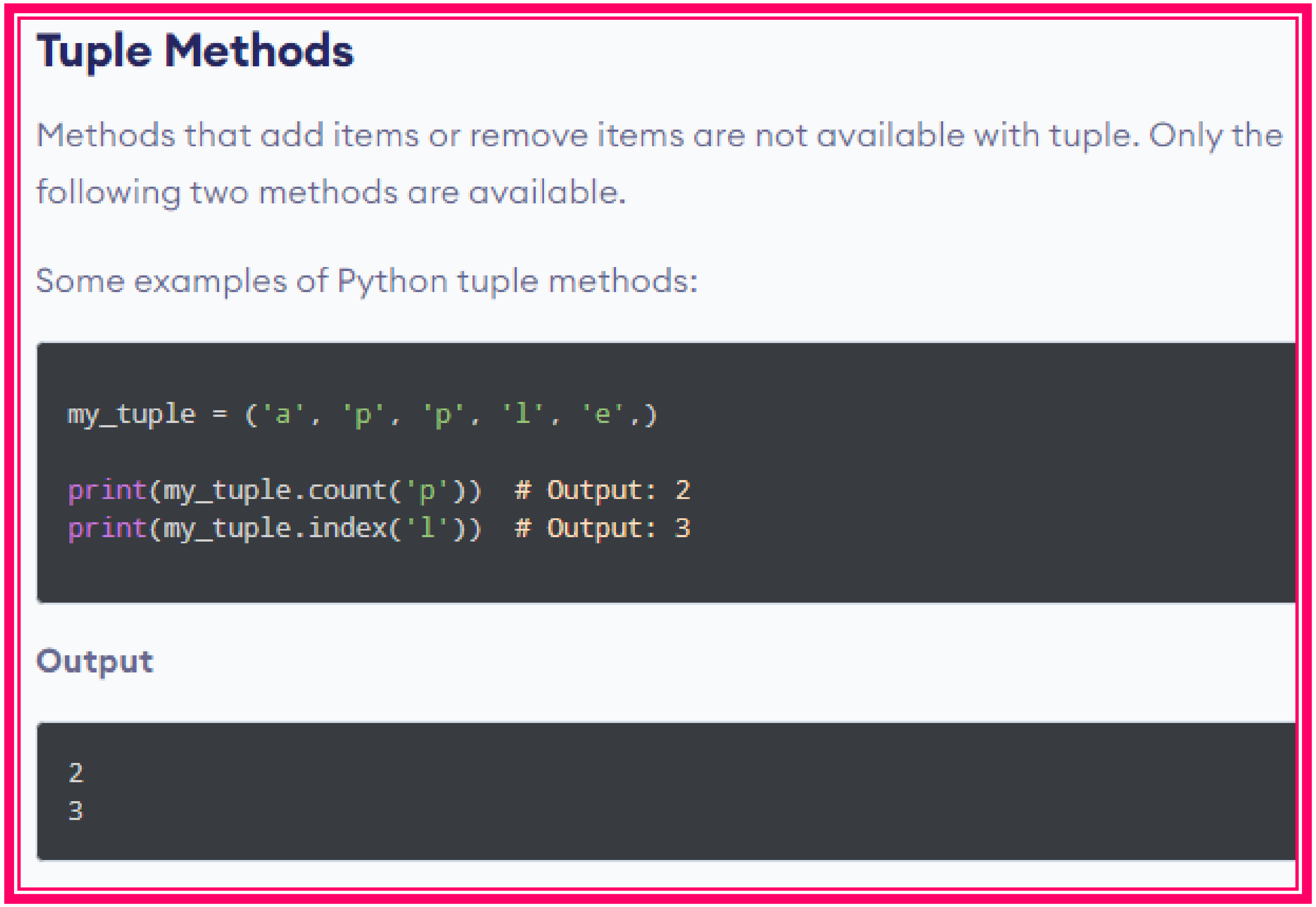


```
# Concatenation
# Output: (1, 2, 3, 4, 5, 6)
print((1, 2, 3) + (4, 5, 6))

# Repeat
# Output: ('Repeat', 'Repeat', 'Repeat')
print(("Repeat",) * 3)
```

Output

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

Tuple Methods



Tuple Methods

Methods that add items or remove items are not available with tuple. Only the following two methods are available.

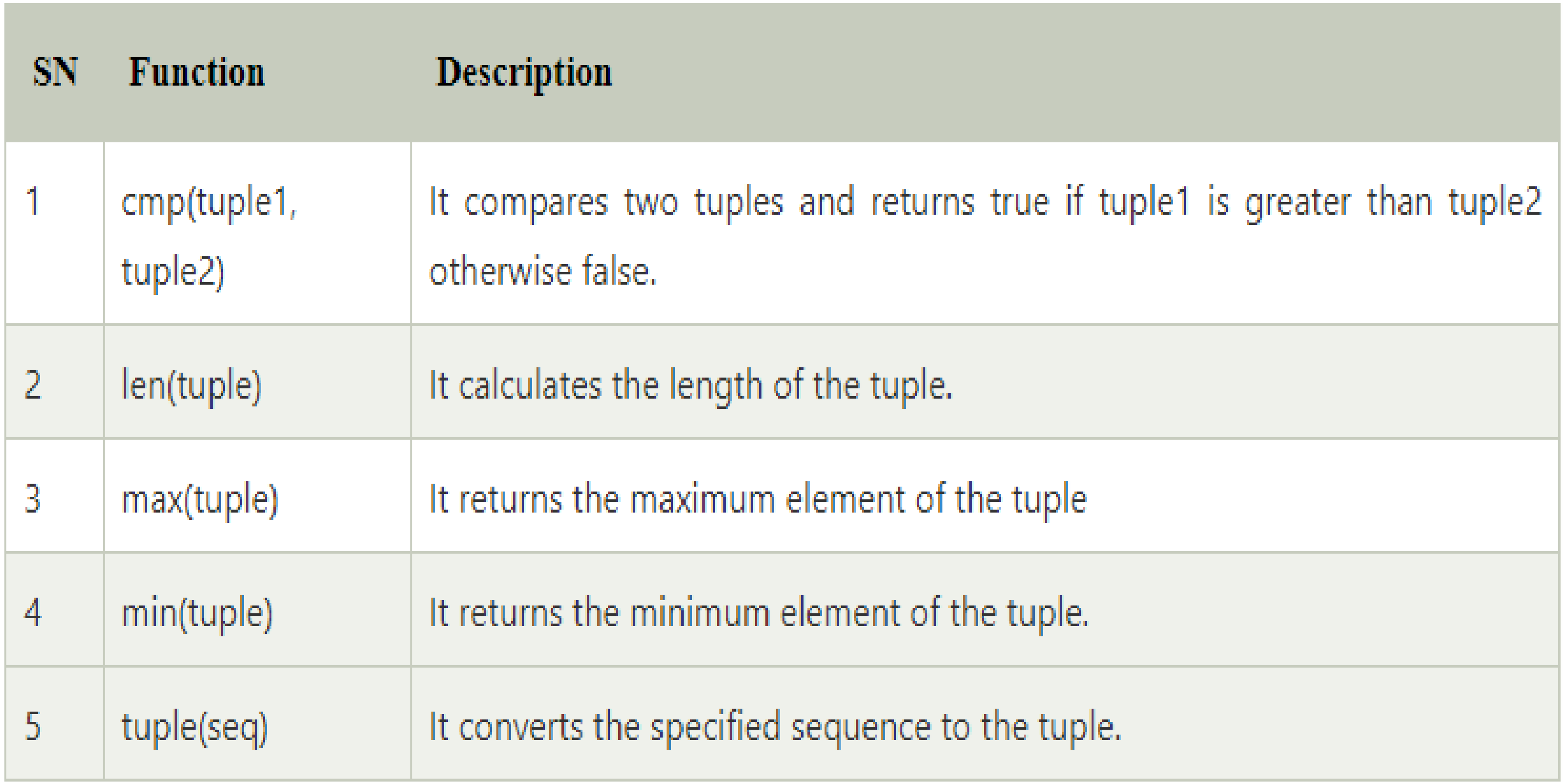
Some examples of Python tuple methods:

```
my_tuple = ('a', 'p', 'p', 'l', 'e',)

print(my_tuple.count('p'))  # Output: 2
print(my_tuple.index('l'))  # Output: 3
```

Output

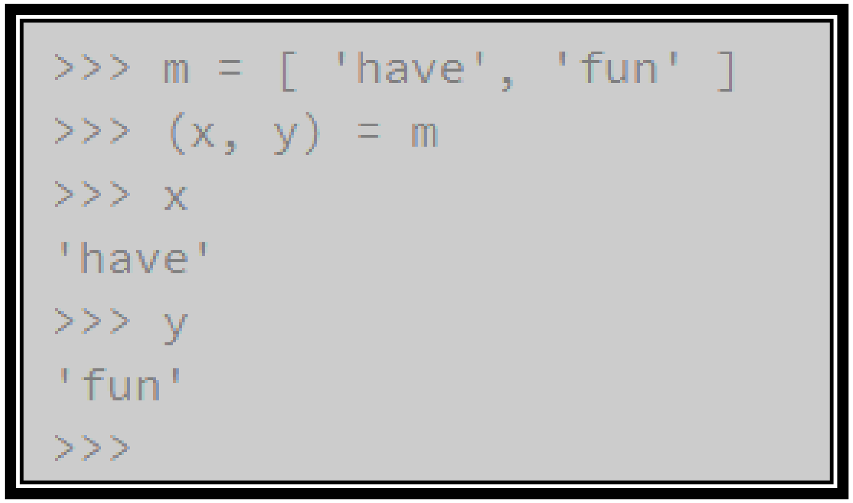
```
2
3
```

Tuple Methods – Inbuilt Functions

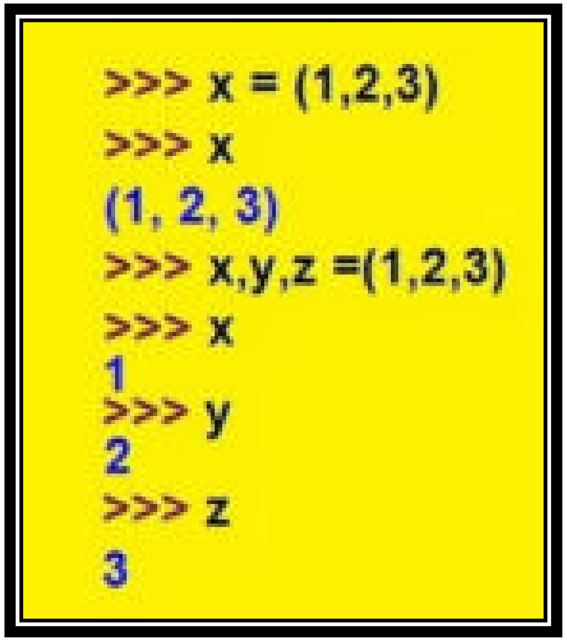


SN	Function	Description
1	cmp(tuple1, tuple2)	It compares two tuples and returns true if tuple1 is greater than tuple2 otherwise false.
2	len(tuple)	It calculates the length of the tuple.
3	max(tuple)	It returns the maximum element of the tuple
4	min(tuple)	It returns the minimum element of the tuple.
5	tuple(seq)	It converts the specified sequence to the tuple.

Tuple Assignment

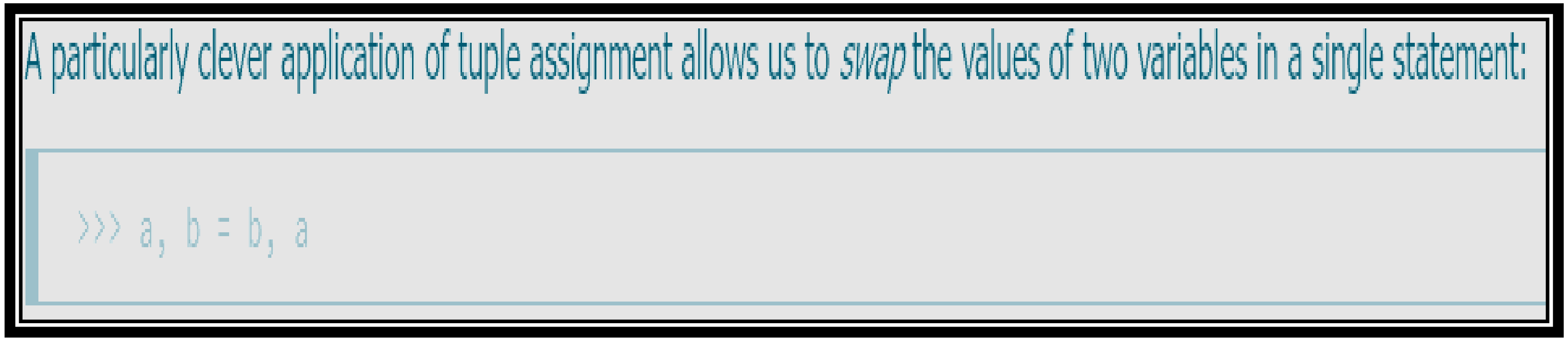


```
>>> m = [ 'have', 'fun' ]
>>> (x, y) = m
>>> x
'have'
>>> y
'fun'
>>>
```



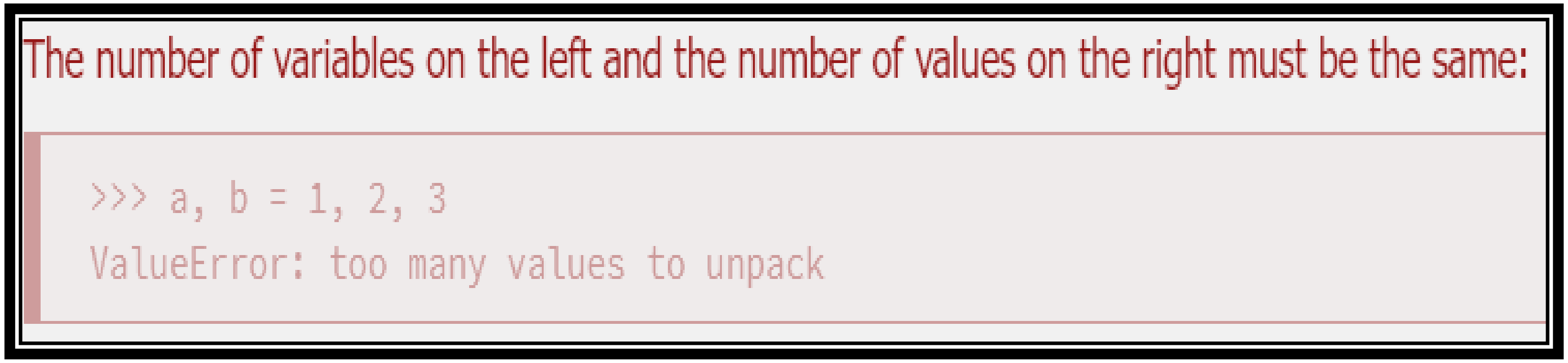
```
>>> x = (1,2,3)
>>> x
(1, 2, 3)
>>> x,y,z =(1,2,3)
>>> x
1
>>> y
2
>>> z
3
```

Contd..



A particularly clever application of tuple assignment allows us to *swap* the values of two variables in a single statement:

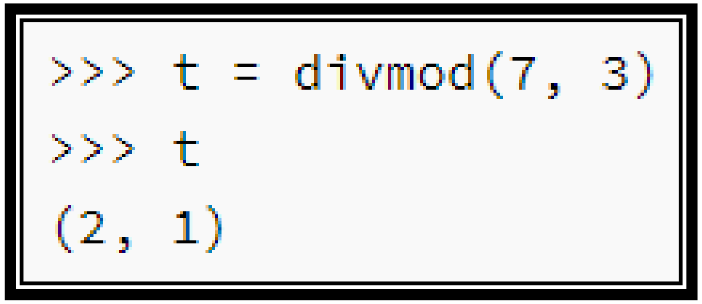
```
>>> a, b = b, a
```



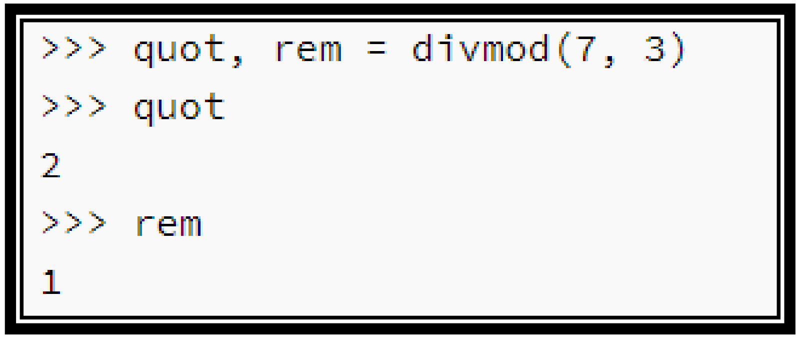
The number of variables on the left and the number of values on the right must be the same:

```
>>> a, b = 1, 2, 3
ValueError: too many values to unpack
```

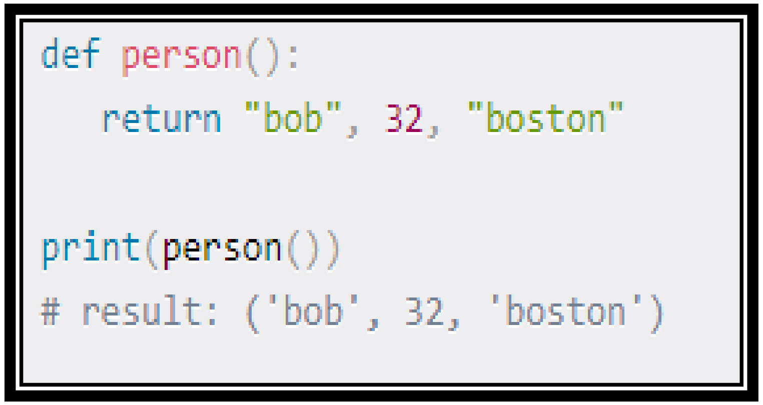
Tuple as return Values



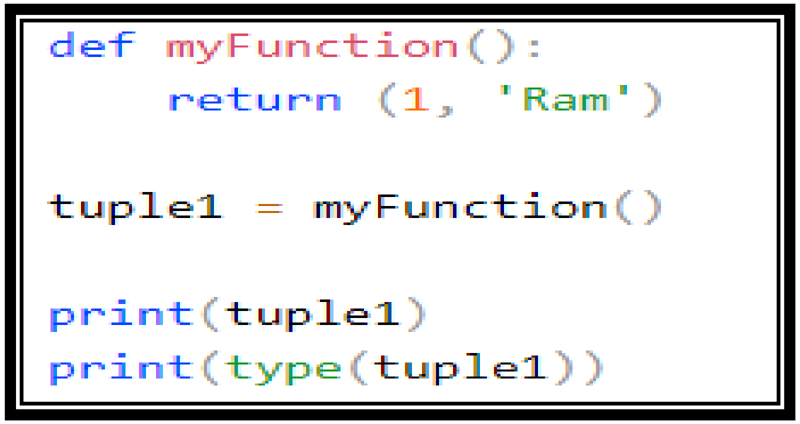
```
>>> t = divmod(7, 3)
>>> t
(2, 1)
```



```
>>> quot, rem = divmod(7, 3)
>>> quot
2
>>> rem
1
```

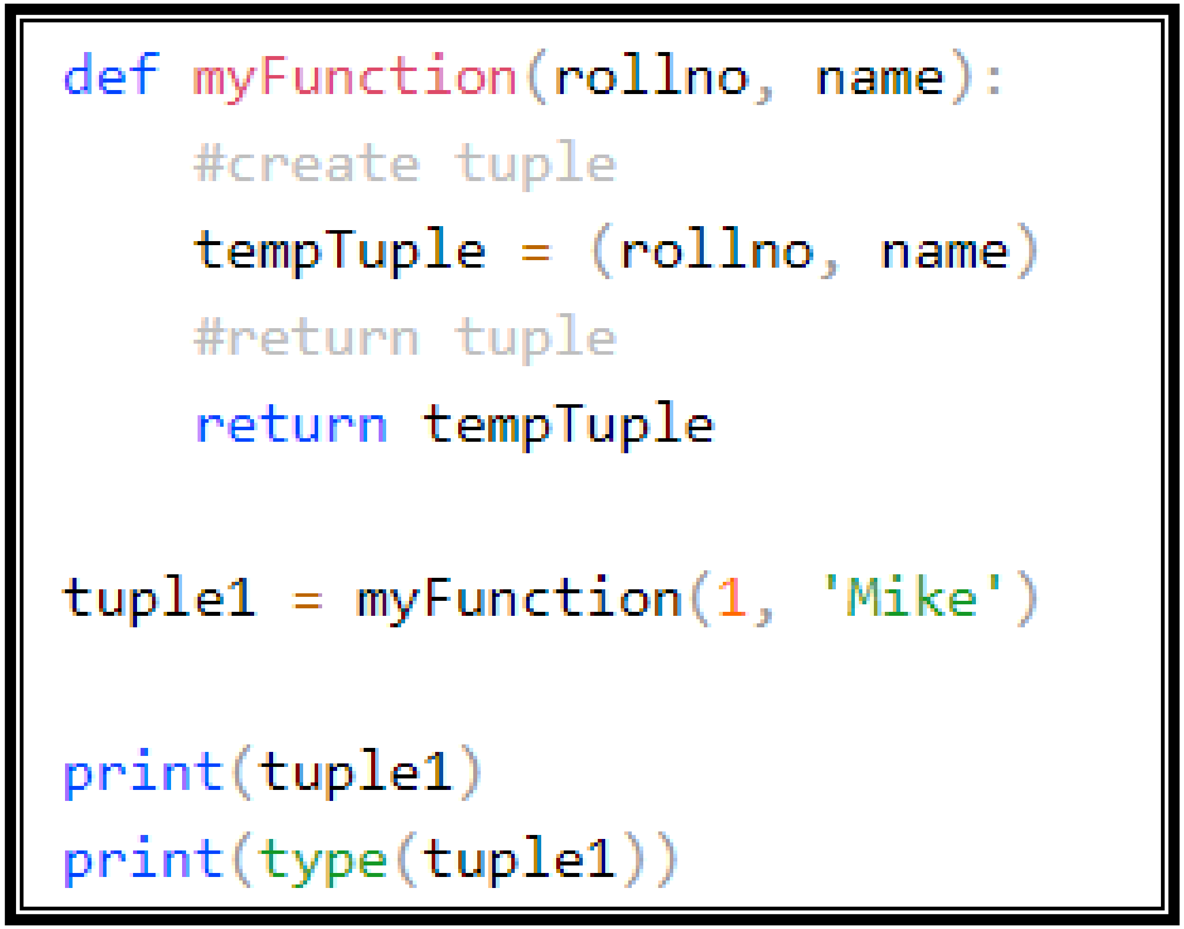


```
def person():
   return "bob", 32, "boston"

print(person())
# result: ('bob', 32, 'boston')
```



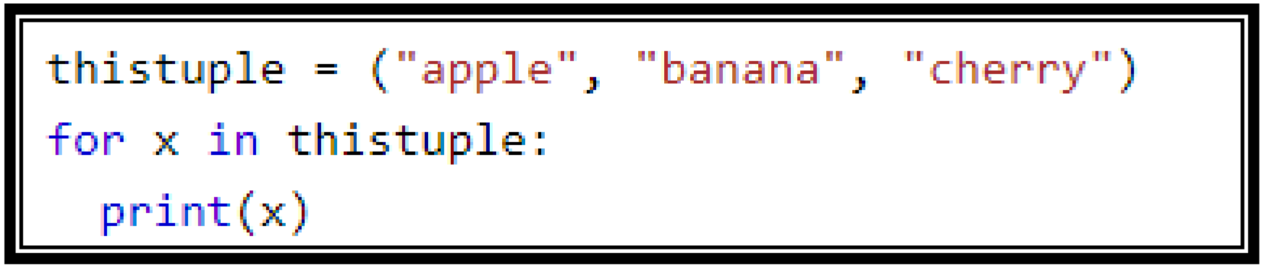
```
def myFunction():
    return (1, 'Ram')

tuple1 = myFunction()

print(tuple1)
print(type(tuple1))
```

Contd..

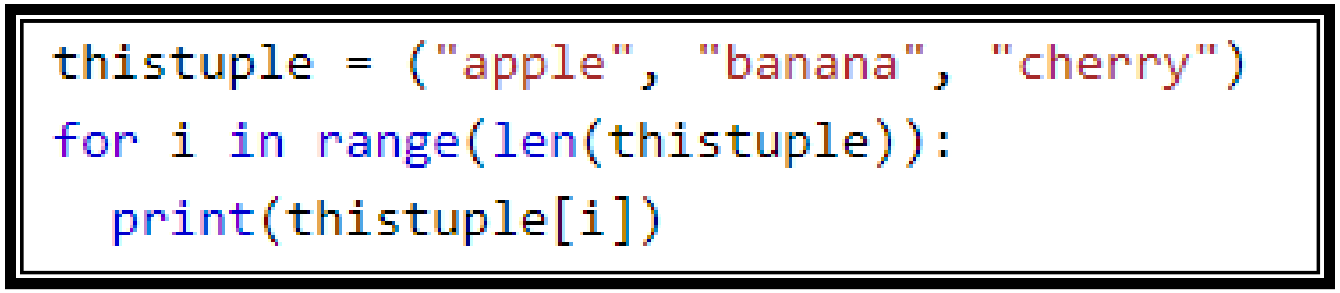


```
def myFunction(rollno, name):
    #create tuple
    tempTuple = (rollno, name)
    #return tuple
    return tempTuple

tuple1 = myFunction(1, 'Mike')

print(tuple1)
print(type(tuple1))
```

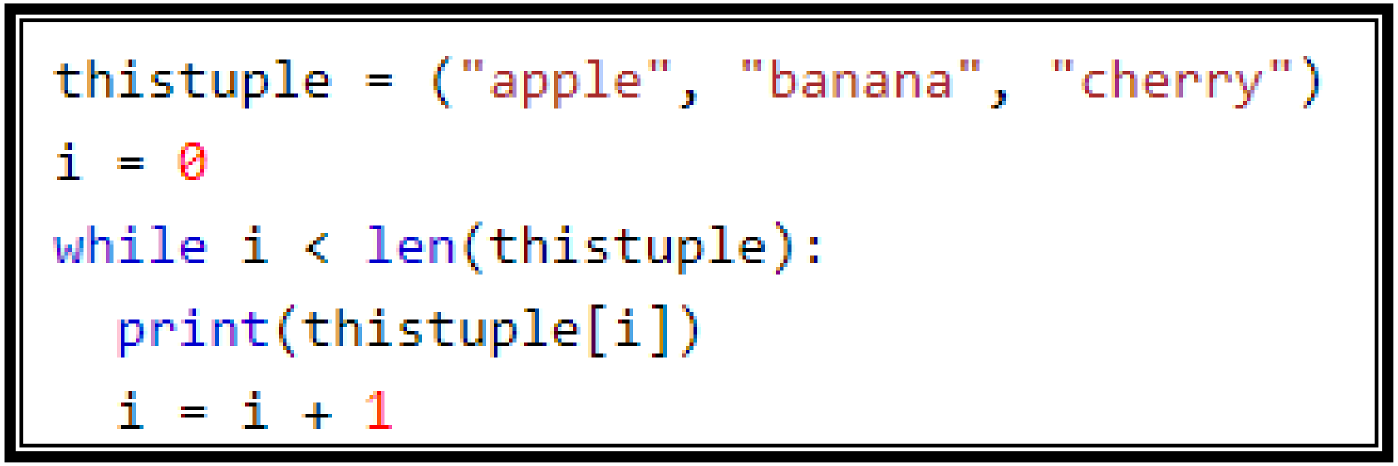
Tuple Traversal



```python
thistuple = ("apple", "banana", "cherry")
for x in thistuple:
  print(x)
```



```python
thistuple = ("apple", "banana", "cherry")
for i in range(len(thistuple)):
  print(thistuple[i])
```



```python
thistuple = ("apple", "banana", "cherry")
i = 0
while i < len(thistuple):
  print(thistuple[i])
  i = i + 1
```

Is Tuple Mutable?

Exception : However, there is an exception in immutability as well. We know that tuple in python is immutable. But the tuple consists of a sequence of names with unchangeable bindings to objects.

Consider a tuple

```
tup = ([3, 4, 5], 'myname')
```

The tuple consists of a string and a list. Strings are immutable so we can't change its value. But the contents of the list can change. **The tuple itself isn't mutable but contain items that are mutable.**

Summary

- Tuple ☐ a sequence datatype holds heterogenous data like List.
- Tuple is immutable ☐ values in the tuple can't be changed. But when a tuple contain list that time tuple becomes mutable as list can be changed.
- Tuple can be traversed through loop
- Tuple have in-built methods to perform operations on its values.
- Tuple assignment is done through using variables and comma separator. Multiple variables can hold a tuple value.

THANK YOU