Python Tuple

A tuple is an ordered collection of values.

Tuples are a lot like lists:

- **Tuples are ordered – Tuples maintains a left-to-right positional ordering among the items they contain.**
- **Accessed by index – Items in a tuple can be accessed using an index.**
- **Tuples can contain any sort of object – It can be numbers, strings, lists and even other tuples.**

except:

- **Tuples are immutable – you can't add, delete, or change items after the tuple is defined.**

Create a Tuple

You can create a tuple by placing a comma-separated sequence of items in parentheses `()`**.**

```
# A tuple of integers
T = (1, 2, 3)


# A tuple of strings
T = ('red', 'green', 'blue')
```

The items of a tuple don't have to be the same type. The following tuple contains an integer, a string, a float, and a boolean.

```
# A tuple with mixed datatypes
T = (1, 'abc', 1.23, True)
```

**A tuple containing zero items is called an empty tuple and you can create one with empty
brackets** `()`

```
# An empty tuple
T = ()
```

Syntactically, a tuple is just a comma-separated list of values.

```
# A tuple without parentheses
T = 1, 'abc', 1.23, True
```

You don't need the parentheses to create a tuple. It's the trailing commas that really define a tuple. But using them doesn't hurt; also they help make the tuple more visible.

Singleton Tuple

If you have only one value in a tuple, you can indicate this by including a trailing comma , just before the closing parentheses.

```
T = (4,)

print(type(T))
# Prints <type 'tuple'>
```

Otherwise, Python will think you've just typed a value inside regular parentheses.

```
# Not a tuple

T = (4)

print(type(T))
# Prints <type 'int'>
```

The tuple() Constructor

You can convert other data types to tuple using Python's tuple() constructor.

```
# Convert a list to a tuple

T = tuple([1, 2, 3])

print(T)
# Prints (1, 2, 3)

# Convert a string to a tuple
```

```
T = tuple('abc')

print(T)
# Prints ('a', 'b', 'c')
```

Nested Tuples

A tuple can contain sub-tuple, which in turn can contain sub-tuples themselves, and so on. This is known as nested tuple. You can use them to arrange data into hierarchical structures.

```
T = ('red', ('green', 'blue'), 'yellow')
```

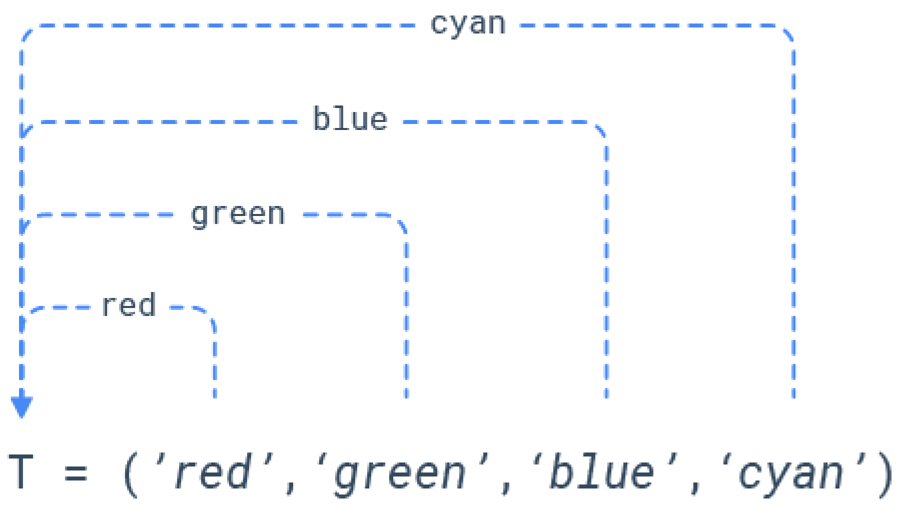
Tuple Packing & Unpacking

Tuple Packing

When a tuple is created, the items in the tuple are packed together into the object.

```
T = ('red', 'green', 'blue', 'cyan')

print(T)
# Prints ('red', 'green', 'blue', 'cyan')
```

In above example, the values ‘red’, ‘green’, ‘blue’ and ‘cyan’ are packed together in a tuple.

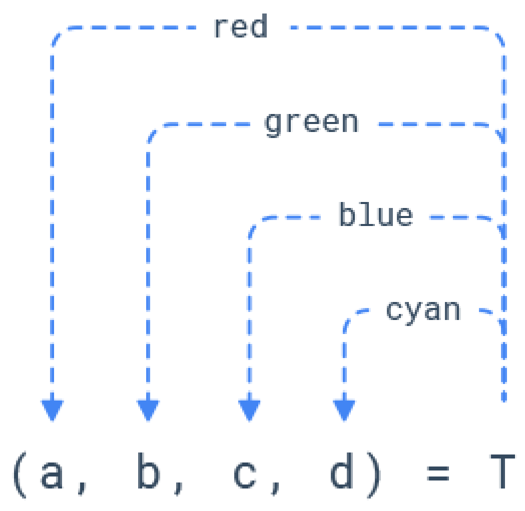


Tuple Unpacking

When a packed tuple is assigned to a new tuple, the individual items are unpacked (assigned to the items of a new tuple).

```
T = ('red', 'green', 'blue', 'cyan')
(a, b, c, d) = T

print(a)
# Prints red

print(b)
# Prints green

print(c)
# Prints blue

print(d)
# Prints cyan
```

In above example, the tuple `T` is unpacked into a, b, c and d variables.



When unpacking, the number of variables on the left must match the number of items in the tuple.

```
# Common errors in tuple unpacking

T = ('red', 'green', 'blue', 'cyan')
(a, b) = T
# Triggers ValueError: too many values to unpack

T = ('red', 'green', 'blue')
(a, b, c, d) = T
# Triggers ValueError: not enough values to unpack (expected 4, got 3)
```

Usage

Tuple unpacking comes handy when you want to swap values of two variables without using a temporary variable.

```
# Swap values of 'a' and 'b'
a = 1
b = 99
```

```
a, b = b, a


print(a)
# Prints 99


print(b)
# Prints 1
```

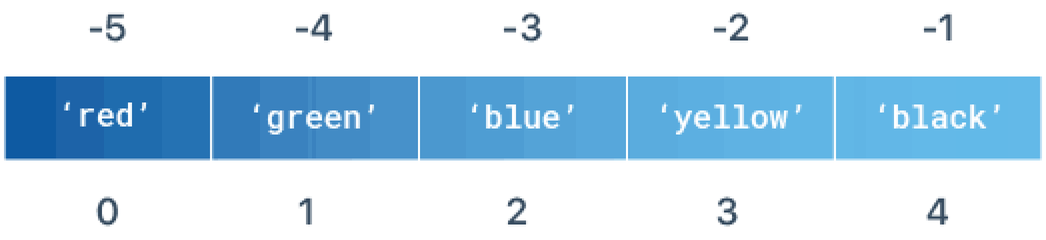
While unpacking a tuple, the right side can be any kind of sequence (tuple, string or list).

```
# Split an email address into a user name and a domain
addr = 'bob@python.org'
user, domain = addr.split('@')


print(user)
# Prints bob


print(domain)
# Prints python.org
```

Access Tuple Items

You can access individual items in a tuple using an index in square brackets. Note that tuple indexing starts from 0.

The indices for the elements in a tuple are illustrated as below:



```
T = ('red', 'green', 'blue', 'yellow', 'black')
```

```
print(T[0])

# Prints red


print(T[2])
# Prints blue
```

You can access a tuple by negative indexing as well. Negative indexes count backward from the end of the tuple. So, `T[-1]` refers to the last item, `T[-2]` is the second-last, and so on.

```
T = ('red', 'green', 'blue', 'yellow', 'black')


print(T[-1])

# Prints black


print(T[-2])
# Prints yellow
```

Tuple Slicing

To access a range of items in a tuple, you need to slice a tuple using a slicing operator. Tuple slicing is similar to list slicing.

```
T = ('a', 'b', 'c', 'd', 'e', 'f')


print(T[2:5])

# Prints ('c', 'd', 'e')


print(T[0:2])

# Prints ('a', 'b')


print(T[3:-1])
# Prints ('d', 'e')
```

Change Tuple Items

Tuples are immutable (unchangeable). Once a tuple is created, it cannot be modified.

```
T = ('red', 'green', 'blue')

T[0] = 'black'
# Triggers TypeError: 'tuple' object does not support item assignment
```

The tuple immutability is applicable only to the top level of the tuple itself, not to its contents. For example, a list inside a tuple can be changed as usual.

```
T = (1, [2, 3], 4)

T[1][0] = 'xx'

print(T)
# Prints (1, ['xx', 3], 4)
```

Delete a Tuple

Tuples cannot be modified, so obviously you cannot delete any item from it. However, you can delete the tuple completely with del keyword.

```
T = ('red', 'green', 'blue')
del T
```

Tuple Concatenation & Repetition

Tuples can be joined using the concatenation operator `+` or Replication operator `*`

```
# Concatenate

T = ('red', 'green', 'blue') + (1, 2, 3)

print(T)

# Prints ('red', 'green', 'blue', 1, 2, 3)
# Replicate
T = ('red',) * 3
print(T)
# Prints ('red', 'red', 'red')
```

Find Tuple Length

To find how many items a tuple has, use len() method.

```
T = ('red', 'green', 'blue')
print(len(T))
# Prints 3
```

Check if item exists in a tuple

To determine whether a value is or isn't in a tuple, you can use in and not in operators with if statement.

```
# Check for presence
T = ('red', 'green', 'blue')
if 'red' in T:
    print('yes')


# Check for absence
T = ('red', 'green', 'blue')
if 'yellow' not in T:
    print('yes')
```

Iterate through a tuple

To iterate over the items of a tuple, use a simple for loop.

```
T = ('red', 'green', 'blue')
for item in T:
    print(item)
# Prints red green blue
```

Tuple Sorting

There are two methods to sort a tuple.

Method 1: Use the built-in sorted() method that accepts any sequence object.

```
T = ('cc', 'aa', 'dd', 'bb')

print(tuple(sorted(T)))
# Prints ('aa', 'bb', 'cc', 'dd')
```

Method 2: Convert a tuple to a mutable object like list (using list constructor), gain access to a sorting method call (sort()) and convert it back to tuple.

```
T = ('cc', 'aa', 'dd', 'bb')

tmp = list(T)    # convert tuple to list

tmp.sort()       # sort list

T = tuple(tmp)   # convert list to tuple
print(T)         # Prints ('aa', 'bb', 'cc', 'dd')
```

Python Tuple Methods

Python has a set of built-in methods that you can call on tuple objects.

Method	Description
count()	**Returns the count of specified item in the tuple**

index()	**Returns the index of first instance of the specified item**

Built-in Functions with Tuple

Python also has a set of built-in functions that you can use with tuple objects.

Method	Description
all()	**Returns True if all tuple items are true**
any()	**Returns True if any tuple item is true**
enumerate()	**Takes a tuple and returns an enumerate object**
len()	**Returns the number of items in the tuple**
max()	**Returns the largest item of the tuple**

min()	**Returns the smallest item of the tuple**
sorted()	**Returns a sorted tuple**
sum()	**Sums items of the tuple**
tuple()	**Converts an iterable (list, string, set etc.) to a tuple**