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

Python Tuple count() Method

Counts the number of occurrences of an item

Usage

Use count() **method to find the number of times the given** item **appears in the tuple.**

Syntax

```
tuple.count(item)
```

Parameter	Condition	Description
Item	Required	Any item (of type string, list, set, etc.) you want to search for.

Examples

```
# Count the number of occurrences of ‘red’
T = ('red', 'green', 'blue')
print(T.count('red'))
# Prints 1
# Count the number of occurrences of number ‘9’
```

```
T = (1, 9, 7, 3, 9, 1, 9, 2)

print(T.count(9))
# Prints 3
```

Count Multiple Items

If you want to count multiple items in a tuple, you can call `count()` in a loop.

This approach, however, requires a separate pass over the tuple for every `count()` call; which can be catastrophic for performance. Use couter() method from class collections, instead.

```
# Count occurrences of all the unique items

T = ('a', 'b', 'c', 'b', 'a', 'a', 'a')

from collections import Counter

print(Counter(T))
# Prints Counter({'a': 4, 'b': 2, 'c': 1})
```

Python Tuple index() Method

Searches the tuple for a given item

Usage

The `index()` method searches for the first occurrence of the given `item` and returns its index. If specified item is not found, it raises 'ValueError' exception.

The optional arguments `start` and `end` limit the search to a particular subsequence of the tuple.

Syntax

```
tuple.index(item,start,end)
```

Parameter	Condition	Description
Item	Required	Any item you want to search for
Start	Optional	An index specifying where to start the search. Default is 0.
End	Optional	An index specifying where to stop the search. Default is the end of the tuple.

Basic Example

```
# Find index of 'green' in a tuple

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

print(T.index('green'))
# Prints 1
```

index() on Duplicate Items

If the tuple has many instances of the specified item, the index() method returns the index of first instance only.

```
# Find first occurrence of character ‘c’

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

print(T.index('c'))
# Prints 2
```

Limit index() Search to Subsequence

If you want to search the tuple from the middle, specify the start parameter.

```
# Find 'c' starting a position 5

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

print(T.index('c',5))
# Prints 8
```

The returned index is computed relative to the beginning of the full sequence rather than the start argument.

You can also specify where to stop the search with end parameter.

```
# Find 'c' in between 5 & 10
T = ('a','b','c','d','e','f','a','b','c','d','e','f')
print(T.index('c',5,10))
# Prints 8
```

index() on Item that Doesn’t Exist

`index()` method raises a ‘ValueError’ if specified `item` is not found in the tuple.

```
T = ('a','b','c','d','e','f','a','b','c','d','e','f')
print(T.index('x'))
# Triggers ValueError: tuple.index(x): x not in tuple


# also within search bound
T = ('a','b','c','d','e','f','a','b','c','d','e','f')
print(T.index('c',4,7))
# Triggers ValueError: tuple.index(x): x not in tuple
```

To avoid such exception, you can check if item exists in a tuple, using in operator inside if statement.

```
T = ('a','b','c','d','e','f','a','b','c','d','e','f')
if 'x' in T:
    print(T.index('x'))
```

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

Python all() Function

Determines whether all items in an iterable are True

Usage

The `all()` function returns True if all items in an iterable are True. Otherwise, it returns False.

If the iterable is empty, the function returns True.

Syntax

```
all(iterable)
```

Parameter	Condition	Description
iterable	**Required**	**An iterable of type (list, string, tuple, set, dictionary etc.)**

Falsy Values

In Python, all the following values are considered False.

- **Constants defined to be false:** None **and** False**.**
- **Zero of any numeric type:** 0**,** 0.0**,** 0j**,** Decimal(0)**,** Fraction(0, 1)
- **Empty sequences and collections:** ''**,** ()**,** []**,** {}**,** set()**,** range(0)

Basic Examples

```
# Check if all items in a list are True

L = [1, 1, 1]
print(all(L))   # Prints True

L = [0, 1, 1]
print(all(L))   # Prints False
```

Here are some scenarios where all() **returns False.**

```
L = [True, 0, 1]
print(all(L))   # Prints False

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

```
print(all(T))   # Prints False


S = {0j, 3+4j}
print(all(S))   # Prints False
```

all() on a Dictionary

When you use `all()` function on a dictionary, it checks if all the keys are true, not the values.

```
D1 = {0: 'Zero', 1: 'One', 2: 'Two'}

print(all(D1))   # Prints False


D2 = {'Zero': 0, 'One': 1, 'Two': 2}
print(all(D2))   # Prints True
```

all() on Empty Iterable

If the iterable is empty, the function returns True.

```
# empty iterable

L = []

print(all(L))   # Prints True


# iterable with empty items

L = [[], []]
print(all(L))   # Prints False
```

Python any() Function

Determines whether any item in an iterable is True

Usage

The any() **function returns True if any item in an** iterable **is True. Otherwise, it returns False.**

If the iterable is empty, the function returns False.

Syntax

```
any(iterable)
```

Parameter	**Condition**	**Description**
iterable	**Required**	**An iterable of type (list, string, tuple, set, dictionary etc.)**

Falsy Values

In Python, all the following values are considered False.

- **Constants defined to be false:** None **and** False**.**
- **Zero of any numeric type:** 0**,** 0.0**,** 0j**,** Decimal(0)**,** Fraction(0, 1)
- **Empty sequences and collections:** ''**,** ()**,** []**,** {}**,** set()**,** range(0)

Basic Examples

```
# Check if any item in a list is True
```

```
L = [0, 0, 0]
print(any(L))   # Prints False


L = [0, 1, 0]
print(any(L))   # Prints True
```

Here are some scenarios where `any()` returns True.

```
L = [False, 0, 1]
print(any(L))   # Prints True


T = ('', [], 'green')
print(any(T))   # Prints True


S = {0j, 3+4j, 0.0}
print(any(S))   # Prints True
```

any() on a Dictionary

When you use `any()` function on a dictionary, it checks if any of the keys is true, not the values.

```
D1 = {0: 'Zero', 0: 'Nil'}
print(any(D1))   # Prints False


D2 = {'Zero': 0, 'Nil': 0}
print(any(D2))   # Prints True
```

any() on Empty Iterable

If the iterable is empty, the function returns False.

```
L = []
print(any(L))   # Prints False
```

Python enumerate() Function

Adds a counter to an iterable

Usage

The `enumerate()` **function adds a counter to an** iterable **and returns it as an enumerate object.**

By default, `enumerate()` **starts counting at 0 but if you add a second argument** start**, it'll start from that number instead.**

Syntax

```
enumerate(iterable,start)
```

Parameter	**Condition**	**Description**
iterable	**Required**	**An iterable (e.g. list, tuple, string etc.)**
start	**Optional**	**A number to start counting from. Default is 0.**

Basic Example

```
# Create a list that can be enumerated
L = ['red', 'green', 'blue']
x = list(enumerate(L))
print(x)
# Prints [(0, 'red'), (1, 'green'), (2, 'blue')]
```

Specify Different Start

By default, `enumerate()` starts counting at 0 but if you add a second argument `start`, it'll start from that number instead.

```
# Start counter from 10
L = ['red', 'green', 'blue']
x = list(enumerate(L, 10))
print(x)
# Prints [(10, 'red'), (11, 'green'), (12, 'blue')]
```

Iterate Enumerate Object

When you iterate an enumerate object, you get a tuple containing (counter, item)

```
L = ['red', 'green', 'blue']
for pair in enumerate(L):
    print(pair)
# Prints (0, 'red')
# Prints (1, 'green')
# Prints (2, 'blue')
```

You can unpack the tuple into multiple variables as well.

```
L = ['red', 'green', 'blue']
for index, item in enumerate(L):
    print(index, item)
# Prints 0 red
# Prints 1 green
# Prints 2 blue
```

Python len() Function

Returns the number of items of an object

Usage

The `len()` function returns the number of items of an object.

The object may be a sequence (such as a string, tuple, list, or range) or a collection (such as a dictionary, set, or frozen set).

Syntax

```
len(object)
```

Parameter	Condition	Description
object	**Required**	**A sequence or a collection.**

len() on Sequences

```
# number of characters in a string

S = 'Python'

x = len(S)

print(x)
# Prints 6

# number of items in a list

L = ['red', 'green', 'blue']

x = len(L)

print(x)
# Prints 3

# number of items in a tuple

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

x = len(T)

print(x)
# Prints 3
```

len() on Collections

```
# number of key:value pairs in a dictionary

D = {'name': 'Bob', 'age': 25}

x = len(D)

print(x)
# Prints 2

# number of items in a set

S = {'red', 'green', 'blue'}

x = len(S)

print(x)
# Prints 3
```

Python max() Function

Returns the largest item

Usage

The `max()` function can find

- the largest of two or more values (such as numbers, strings etc.)
- the largest item in an iterable (such as list, tuple etc.)

With optional `key` parameter, you can specify custom comparison criteria to find maximum value.

Syntax

```
max(val1,val2,val3… ,key)
```

Parameter	Condition	Description
val1,val2,val3…	Required	Two or more values to compare
key	Optional	A function to specify the comparison criteria. Default value is None.

– OR –

```
max(iterable,key,default)
```

Parameter	Condition	Description
iterable	**Required**	**Any iterable, with one or more items to compare**
key	**Optional**	**A function to specify the comparison criteria. Default value is None.**
default	**Optional**	**A value to return if the iterable is empty. Default value is False.**

Find Maximum of Two or More Values

If you specify two or more values, the largest value is returned.

```
x = max(10, 20, 30)

print(x)
# Prints 30
```

If the values are strings, the string with the highest value in alphabetical order is returned.

```
x = max('red', 'green', 'blue')

print(x)
# Prints red
```

You have to specify minimum two values to compare. Otherwise, TypeError exception is raised.

Find Maximum in an Iterable

If you specify an Iterable (such as list, tuple, set etc.), the largest item in that iterable is returned.

```
L = [300, 500, 100, 400, 200]

x = max(L)

print(x)
# Prints 500
```

If the iterable is empty, a ValueError is raised.

```
L = []

x = max(L)

print(x)
# Triggers ValueError: max() arg is an empty sequence
```

To avoid such exception, add default **parameter. The** default **parameter specifies a value to return if the provided iterable is empty.**

```
# Specify default value '0'

L = []

x = max(L, default='0')

print(x)
# Prints 0
```

Find Maximum with Built-in Function

With optional key **parameter, you can specify custom comparison criteria to find maximum value. A** key **parameter specifies a function to be executed on each iterable's item before making comparisons.**

For example, with a list of strings, specifying `key=len` **(the built-in [len()](#) function) finds longest string.**

```
L = ['red', 'green', 'blue', 'black', 'orange']

x = max(L, key=len)
```

```
print(x)
# Prints orange
```

Python min() Function

Returns the smallest item

Usage

The `min()` function can find

- **the smallest of two or more values (such as numbers, strings etc.)**
- **the smallest item in an iterable (such as list, tuple etc.)**

With optional `key` parameter, you can specify custom comparison criteria to find minimum value.

Syntax

```
min(val1,val2,val3… ,key)
```

Parameter	Condition	Description
val1,val2,val3…	**Required**	**Two or more values to compare**

key	**Optional**	**A function to specify the comparison criteria. Default value is None.**

– OR –

```
min(iterable,key,default)
```

Parameter	Condition	Description
iterable	**Required**	**Any iterable, with one or more items to compare**
key	**Optional**	**A function to specify the comparison criteria. Default value is None.**
default	**Optional**	**A value to return if the iterable is empty. Default value is False.**

Find Minimum of Two or More Values

If you specify two or more values, the smallest value is returned.

```
x = min(10, 20, 30)

print(x)
# Prints 10
```

If the values are strings, the string with the lowest value in alphabetical order is returned.

```
x = min('red', 'green', 'blue')

print(x)
# Prints blue
```

You have to specify minimum two values to compare. Otherwise, TypeError exception is raised.

Find Minimum in an Iterable

If you specify an Iterable (such as list, tuple, set etc.), the smallest item in that iterable is returned.

```
L = [300, 500, 100, 400, 200]

x = min(L)

print(x)
# Prints 100
```

If the iterable is empty, a ValueError is raised.

```
L = []

x = min(L)

print(x)
# Triggers ValueError: min() arg is an empty sequence
```

To avoid such exception, add `default` **parameter. The** `default` **parameter specifies a value to return if the provided iterable is empty.**

```
# Specify default value '0'

L = []

x = min(L, default='0')

print(x)
```

```
# Prints 0
```

Find Minimum with Built-in Function

With optional `key` parameter, you can specify custom comparison criteria to find minimum value. A `key` parameter specifies a function to be executed on each iterable's item before making comparisons.

For example, with a list of strings, specifying `key=len` (the built-in len() function) finds shortest string.

```
L = ['red', 'green', 'blue']
x = min(L, key=len)
print(x)
# Prints red
```

Python sorted() Function

Sorts the items of an iterable

Usage

The `sorted()` method sorts the items of any `iterable`

You can optionally specify parameters for sort customization like sorting order and sorting criteria.

Syntax

```
sorted(iterable,key,reverse)
```

The method has two optional arguments, which must be specified as keyword arguments.

Parameter	Condition	Description
iterable	**Required**	**Any iterable (list, tuple, dictionary, set etc.) to sort.**
key	**Optional**	**A function to specify the sorting criteria. Default value is None.**
reverse	**Optional**	**Settting it to True sorts the list in reverse order. Default value is False.**

Return Value

The method returns a new sorted list from the items in iterable**.**

Sort Iterables

sorted() **function accepts any** iterable **like list, tuple, dictionary, set, string etc.**

```
# strings are sorted alphabetically
L = ['red', 'green', 'blue', 'orange']
x = sorted(L)
print(x)
# Prints ['blue', 'green', 'orange', 'red']
# numbers are sorted numerically
L = [42, 99, 1, 12]
x = sorted(L)
print(x)
# Prints [1, 12, 42, 99]
```

If you want to sort the list in-place, use built-in sort() method.

sorted() **is actually faster than** sorted() **as it doesn't need to create a new list.**

```
# Sort a tuple
L = ('cc', 'aa', 'dd', 'bb')
x = sorted(L)
print(x)
# Prints ['aa', 'bb', 'cc', 'dd']
```

sorted() **function sorts a dictionary by keys, by default.**

```
D = {'Bob':30, 'Sam':25, 'Max':35, 'Tom':20}
x = sorted(D)
print(x)
# Prints ['Bob', 'Max', 'Sam', 'Tom']
```

To sort a dictionary by values use the sorted() **function along with the values() method.**

```
D = {'Bob':30, 'Sam':25, 'Max':35, 'Tom':20}
x = sorted(D.values())
print(x)
# Prints [20, 25, 30, 35]
```

Sort in Reverse Order

You can also sort an iterable **in reverse order by setting** reverse **to true.**

```
L = ['cc', 'aa', 'dd', 'bb']
x = sorted(L, reverse=True)
```

```
print(x)
# Prints ['dd', 'cc', 'bb', 'aa']
```

Sort with Key

Use key parameter for more complex custom sorting. A key parameter specifies a function to be executed on each list item before making comparisons.

For example, with a list of strings, specifying `key=len` (the built-in len() function) sorts the strings by length, from shortest to longest.

```
L = ['orange', 'red', 'green', 'blue']

x = sorted(L, key=len)

print(x)
# Prints ['red', 'blue', 'green', 'orange']
```

Python sum() Function

Sums items of an iterable

Usage

The `sum()` function sums the items of an iterable and returns the total.

If you specify an optional parameter start, it will be added to the final sum.

This function is created specifically for numeric values. For other values, it will raise TypeError.

Syntax

sum(iterable,start)

Parameter	Condition	Description
iterable	**Required**	**An iterable (such as list, tuple etc.)**
start	**Optional**	**A value to be added to the final sum. Default is 0.**

Examples

```
# Return the sum of all items in a list
L = [1, 2, 3, 4, 5]
x = sum(L)
print(x)
# Prints 15
```

If you specify an optional parameter start**, it will be added to the final sum.**

```
# Start with '10' and add all items in a list
L = [1, 2, 3, 4, 5]
x = sum(L, 10)
print(x)
# Prints 25
```

Python tuple() Function

Creates a tuple from an iterable

Usage

The `tuple()` **function creates a tuple from an** iterable**.**

The iterable may be a sequence (such as a string, list or range) or a collection (such as a dictionary, set or frozen set)

Syntax

```
tuple(iterable)
```

Parameter	Condition	Description
iterable	**Required**	**A sequence or a collection**

Examples

`tuple()` **with no arguments creates an empty tuple.**

```
T = tuple()
print(T)
# Prints ()
```

You can convert any sequence (such as a string, list or range) into a tuple using a `tuple()` **method.**

```
# string into tuple
T = tuple('abc')
```

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

# list into tuple
T = tuple([1, 2, 3])
print(T)
# Prints (1, 2, 3)

# sequence into tuple
T = tuple(range(0, 4))
print(T)
# Prints (0, 1, 2, 3)
```

You can even convert any collection (such as a dictionary, set or frozen set) into a tuple.

```
# dictionary keys into tuple
T = tuple({'name': 'Bob', 'age': 25})
print(T)
# Prints ('age', 'name')

# set into tuple
L = tuple({1, 2, 3})
print(L)
# Prints (1, 2, 3)
```