Tuples
Reference

Operator [ ]

Usage

```
tuple[index]
tuple[:]
tuple[index:]
tuple[:index]
tuple[index1:index2]
```

Description

This operator can be used in the following contexts:

- Expression `tuple[index]` returns the element at position `index` in the given tuple.
- Expression `tuple[:]` returns a copy of the entire tuple.
- Expression `tuple[index:]` returns a new tuple containing all elements of the given tuple, starting at the position given in `index`.
- Expression `tuple[:index]` returns a new tuple containing all elements of the given tuple starting with the first and ending with the element at position `index` − 1.
- Expression `tuple[index1:index2]` returns a new tuple containing all elements between positions `index1` and `index2` − 1.

Indices can be expressed as positive or negative values. Index 0 represents the first element in the tuple; index 1 is the second element, etc. Index −1 represents the last element in the tuple; index −2 represents the second to last, etc.

Example

```
# Create tuple
x = ('John', 'Mary', 'Adam', 'Susan')

# Using positive indices
print(x[1])              # Prints 'Mary'
print(x[:])              # Prints ('John', 'Mary', 'Adam', 'Susan')
print(x[2:])             # Prints ('Adam', 'Susan')
print(x[:3])             # Prints ('John', 'Mary', 'Adam')
print(x[1:3])            # Prints ('Mary', 'Adam')

# Using negative indices
print(x[-1], x[-2])      # Prints 'Susan Adam'
print(x[-3:-1])          # Prints ('Mary', 'Adam')
```

Operator +

Usage

```
tuple1 + tuple2
```

Description

When the `+` operator is surrounded with two tuples, it serves the purpose of tuple concatenation. A new tuple is created containing all elements in `tuple1` followed by the elements in `tuple2`.

Example

```
men = ('John', 'Adam')
women = ('Mary', 'Susan')
persons = men + women
print(persons)   # Prints ('John', 'Adam', 'Mary', 'Susan')
```

Operator *

Usage

```
tuple * value
```

Description

When the `*` operator is used with a tuple on its left and an integer number on its right, it serves the purpose of concatenating `tuple` with itself, as many times as indicated by `value`. This expression returns a new tuple containing all elements in `tuple` repeated `value` times.

Example

```
persons = ('John', 'Mary')
many_persons = persons * 3
print(many_persons)   # Prints ('John', 'Mary', 'John', 'Mary', 'John', 'Mary')
```

Operator `del`

Usage

```
del tuple
```

Description

When the `del` operator is followed by a tuple name (or a variable name of any type, for that matter), it frees the entire tuple (or variable). Referencing that tuple (or variable) after that will cause a `NameError` exception.

Example

```
# Create tuple
x = ('John', 'Mary', 'Susan')

# Remove entire tuple
del x
print(x)         # Raises 'NameError' exception
```

Built-in function `all()`

Usage

```
all(tuple)
```

Description

The `all` built-in function takes a tuple as an argument and returns a Boolean value indicating whether all elements in *`tuple`* are `True`. If any element in the tuple is not a Boolean value, it is first converted using the same rules as the `bool()` type conversion function uses.

If the tuple is empty, the `all()` function returns `True`.

Example

```
print(all( () ))                            # Prints True (empty tuple)
print(all((True, True, True)))              # Prints True
print(all((True, False, True)))             # Prints False
print(all((1, 'x', True)))                  # Prints True
```

Built-in function `any()`

Usage

```
any(tuple)
```

Description

The `any` built-in function takes a tuple as an argument and returns a Boolean value indicating whether at least one of the elements in *`tuple`* is `True`. If any element in the tuple is not a Boolean value, it is first converted using the same rules as the `bool()` type conversion function uses.

If the tuple is empty, the `any()` function returns `False`.

Example

```
print(any( () ))                            # Prints False (empty tuple)
print(any((True, False, False)))            # Prints True
print(any((False, False, False)))           # Prints False
print(any((0, '', False)))                  # Prints False
```

Built-in function `len()`

Usage

```
len(tuple)
```

Description

The `len()` built-in function returns the number of elements present in argument *`tuple`*.

Example

```
x = (2, 4, 6)
print(len(x))    # Prints 3
```

Built-in function `max()`

Usage

```
max(tuple)
```

Description

The `max()` built-in function returns the maximum value among all elements present in the tuple.

Example

```
x = (2, 5, 3)
print(max(x))    # Prints 5
```

Built-in function `min()`

Usage

```
min(tuple)
```

Description

The `min()` built-in function returns the minimum value among all elements present in the tuple.

Example

```
x = (2, 5, 3])
print(min(x))    # Prints 2
```

Built-in function `sum()`

Usage

```
sum(tuple)
```

Description

Obtain the sum of all elements in the tuple. If the tuple is empty, this function returns 0.

Example

```
print(sum((2, 5, 3)))    # Prints 10
print(sum( () ))         # Prints 0 for empty tuple
```

Method `count()`

Usage

```
tuple.count(value)
```

Description

Return the number of occurrences of *`value`* in *`tuple`*. If the given value is not present in the tuple, this method returns 0.

Example

```
x = ('Adam', 'John', 'Adam', 'Mary')
print(x.count('Adam'))     # Prints 2
print(x.count('Susan'))    # Prints 0
```

Method `index()`

Usage

```
tuple.index(value, [start], [end])
```

Description

Return the position of the first occurrence of *`value`* in *`tuple`*.

Arguments *`start`* and *`end`* are both optional. If they are given, the search space is limited between index *`start`* and *`end`* – 1.

If *`value`* is not present in the tuple within the search space (or in the entire tuple if *`start`* and *`end`* are not given), this function raises a `ValueError` exception.

Example

```
names = ('Alice', 'Kevin', 'Susan', 'Kevin', 'Taylor')
print(names.index('Kevin'))           # Prints 1
print(names.index('Susan', 1, 4))     # Prints 2
print(names.index('Taylor', 1, 4))    # Raises 'ValueError' exception
```