

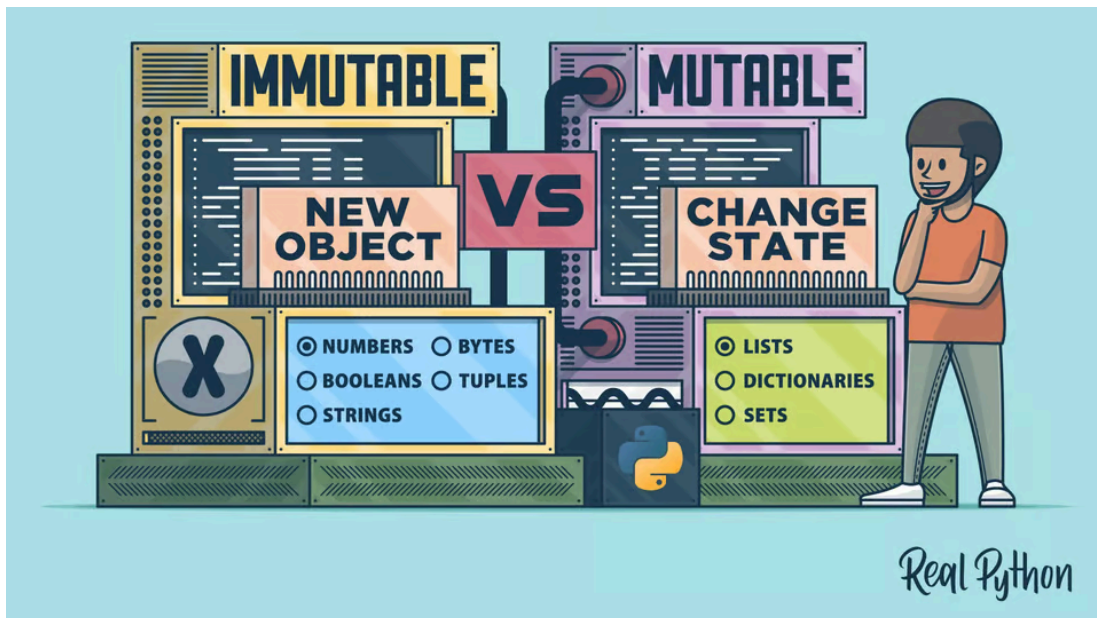
4. Tuple

In this section, we'll explore the concept of tuples in Python.

Tuples are ordered, immutable sequences that can store various types of data. We'll learn how to create, access, and manipulate tuples effectively.

We'll learn about the following topics:

- 4.1. Creating Tuples
- 4.2. Built-in Tuple Functions
- 4.3. Tuple Indexing and Slicing
- 4.4. Tuple Properties
- 4.5. Tuple Operators
- 4.6. Built-in Tuple Methods
- 4.7. Nesting Tuples



Name	Type in Python	Description	Example
Tuples	tuple	Comma-separated ordered immutable sequence of objects in parentheses ().	(10,'hello',15.9)

```
In [1]: type((23, 98.3, 'hello'))
```

```
Out[1]: tuple
```

4.1. Creating Tuples:

Tuples in Python are created using parentheses `()`. You can include elements within the parentheses, separated by commas.

```
In [2]: a = ('one', 23, 98.3, 'hello')
```

If you try to define a tuple with one item.

```
In [3]: type((2))
```

```
Out[3]: int
```

To tell Python that you really want to define a singleton tuple, include a trailing comma `(,)` just before the closing parenthesis.

```
In [4]: type((2,))
```

```
Out[4]: tuple
```

This is another way to define a tuple's items.

```
In [5]: x1, x2, x3 = 4, 5, 6
        t = x1, x2, x3
        print(t)
        type(t)
```

```
(4, 5, 6)
```

```
Out[5]: tuple
```

4.2. Built-in Tuple Functions:

- **len()**: Python's built-in `len()` function determines the total number of elements within a tuple's sequence.

```
In [6]: a = ('one', 23, 98.3, 'hello')
```

```
In [7]: len(a)
```

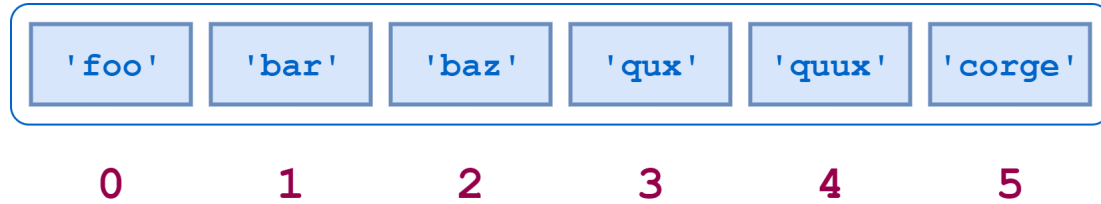
```
Out[7]: 4
```

4.3. Tuple Indexing and Slicing:

```
In [8]: print(a)
```

```
('one', 23, 98.3, 'hello')
```

Elements in a tuple can be accessed individually by using square brackets and an index, just like accessing individual characters in a string. All lists, tuples, and strings indexing follow a zero-based approach.



```
In [9]: #Grab element at index 0  
a[0]
```

```
Out[9]: 'one'
```

```
In [10]: #Grab index 1 and everything past it  
a[1:]
```

```
Out[10]: (23, 98.3, 'hello')
```

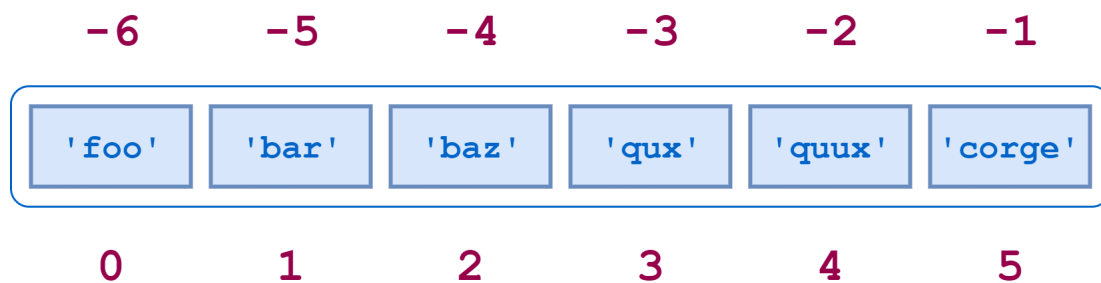
```
In [11]: #Grab everything UP TO index 2  
a[:2]
```

```
Out[11]: ('one', 23)
```

```
In [12]: a[2:len(a)]
```

```
Out[12]: (98.3, 'hello')
```

Negative indexing allows counting from the end of the tuple.



```
In [13]: a[-1]
```

```
Out[13]: 'hello'
```

```
In [14]: a[:-3]
```

```
Out[14]: ('one',)
```

[start:stop:step]

- start: numerical index for the slice index
- stop: index you will go up to but not include
- step: the size of jump you take

```
In [15]: a[:2]
```

```
Out[15]: ('one', 98.3)
```

You can specify a negative step value as well, in which case Python steps backward through the tuple. In that case, the starting/first index should be greater than the ending/second index.

```
In [16]: a
```

```
Out[16]: ('one', 23, 98.3, 'hello')
```

```
In [17]: a[4:0:-2]
```

```
Out[17]: ('hello', 23)
```

This is a common paradigm for reversing a tuple.

```
In [18]: a[::-1]
```

```
Out[18]: ('hello', 98.3, 23, 'one')
```

4.4. Tuple Properties:

- **Ordered:** A tuple in Python is not just a collection of objects; it is an ordered collection where the sequence in which elements are specified upon tuple creation becomes an inherent characteristic of that tuple, preserved throughout its lifetime.

```
In [19]: a = ('one', 23, 98.3, 'hello')
```

```
b = (23, 98.3, 'one', 'hello')
```

```
In [20]: a is b
```

```
Out[20]: False
```

```
In [21]: a == b
```

```
Out[21]: False
```

- **Immutability:** Tuples are immutable meaning that when a tuple is created, the elements within it cannot be changed or replaced.

This is the biggest difference between Tuples and Lists.

In [22]: a

Out[22]: ('one', 23, 98.3, 'hello')

In [23]: a[0] = 2

```
-----
TypeError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_18304\4221126460.py in <module>
----> 1 a[0] = 2

TypeError: 'tuple' object does not support item assignment
```

In [24]: a.append('new')

```
-----
AttributeError                            Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_18304\2298863157.py in <module>
----> 1 a.append('new')

AttributeError: 'tuple' object has no attribute 'append'
```

In [25]: a = a + ('new')

```
-----
TypeError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_18304\1405506363.py in <module>
----> 1 a = a + ('new')

TypeError: can only concatenate tuple (not "str") to tuple
```

In [26]: a.remove('one')

```
-----
AttributeError                            Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_18304\3225995218.py in <module>
----> 1 a.remove('one')

AttributeError: 'tuple' object has no attribute 'remove'
```

Nothing can be added/removed/replaced in tuples unlike lists.

In [27]: del a[0]

```
-----
TypeError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_18304\3074760810.py in <module>
----> 1 del a[0]

TypeError: 'tuple' object doesn't support item deletion
```

- **Repetition:** We can use the multiplication symbol to create repetition.

In [28]: a * 2

```
Out[28]: ('one', 23, 98.3, 'hello', 'one', 23, 98.3, 'hello')
```

4.5. Tuple Operators:

- **in** : Python also provides a membership operator that can be used with tuples. The in operator returns True if the first operand is contained within the second, and False otherwise.

```
In [29]: 'one' in a
```

```
Out[29]: True
```

- **not in** : Python also provides a membership operator that can be used with tuples. The not in operator returns True if the first operand is not contained within the second, and False otherwise.

```
In [30]: 'one' not in a
```

```
Out[30]: False
```

4.6. Built-in Tuple Methods:

Method	Description
<code>index(m,a(optional),b(optional))</code>	the index of first occurrence of m for a given substring indicated by a and b
<code>count(m,a(optional),b(optional))</code>	number of occurrences of m within the substring indicated by a and b

```
In [31]: a.index(23)
```

```
Out[31]: 1
```

```
In [32]: a.count('one')
```

```
Out[32]: 1
```

To be honest, tuples are not used as often as lists in programming, but are used when immutability is necessary. If in your program you are passing around an object and need to make sure it does not get changed, then a tuple becomes your solution. It provides a convenient source of data integrity.

4.7. Nesting Tuples:

```
In [33]: b = ([1,'hello'], (26.3, 56))
```

```
In [34]: print(b)
```

```
([1, 'hello'], (26.3, 56))
```

```
In [35]: b[0][1]
```

```
Out[35]: 'hello'
```

```
In [36]: b[1][1]
```

```
Out[36]: 56
```