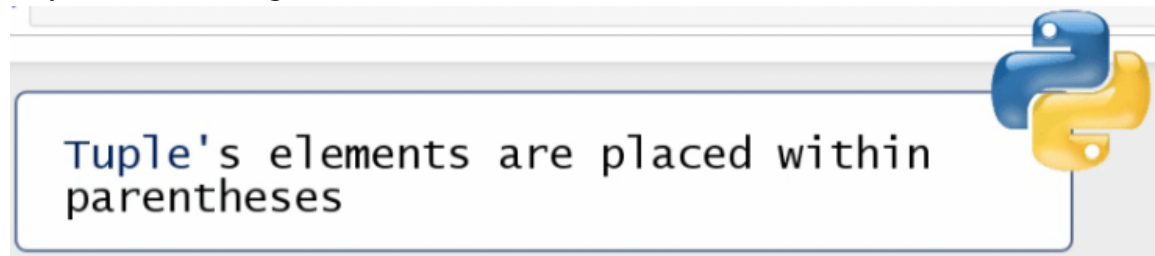# Tuples

In Python tuples are very similar to lists, however, unlike lists they are *immutable* meaning they can not be changed.

Tuple's elements are placed within parentheses

You would use tuples to present things that shouldn't be changed, such as days of the week, or dates on a calendar.

In this section, we will get a brief overview of the following:

```
1.) Constructing Tuples
2.) Basic Tuple Methods
3.) Immutability
4.) When to Use Tuples
```

You'll have an intuition of how to use tuples based on what you've learned about lists. We can treat them very similarly with the major distinction being that tuples are immutable.

	Mutable	Ordered	Indexing / Slicing	Duplicate Elements
List	✓	✓	✓	✓
Tuple	✗	✓	✓	✓
Set	✓	✗	✗	✗

Constructing Tuples

The construction of a tuples use () with elements separated by commas. For example:

```
In [4]: a, b, c = 1, 2, 3    # Assigning values to multiple variables

print(a)
print(c)
```

```
1
3
```

In [1]:
```python
t = (1,2,3)
```

In [2]:
```python
print(t)
```
```
(1, 2, 3)
```

In [5]:
```python
print(len(t))
```
```
3
```

the default sequence type

if multiple values are enlist the new variable will be will perceived as a tuple.

In [3]:
```python
t1 = 1,2,3

print(t1)
```
```
(1, 2, 3)
```


In [6]:
```python
# Can also mix object types
t = ('one', 2)

print(t)
```
```
('one', 2)
```

In [7]:
```python
print(t[0])
```
```
one
```

In [8]:
```python
print(t[-1])
```
```
2
```

Basic Tuple Methods

Tuples have built-in methods, but not as many as lists do. Let's look at two of them:

In [9]:
```python
# Use .index to enter a value and return the index
print(t.index('one'))
```
```
0
```

In [10]:
```python
# Use .count to count the number of times a value appears
print(t.count('one'))
```
```
1
```

```python
In [13]: t = ('one',2, 'one')
         print(t.count('one'))
```

```
2
```

```python
In [10]: (a,b) = 4,7
```

```python
In [11]: (a, b) = "4,7"
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-11-0d01118de6c2> in <module>
----> 1 (a, b) = "4,7";

ValueError: too many values to unpack (expected 2)
```

```python
In [13]: (a, b) = "4,7".split(",")
```

```python
In [14]: print(a);
         print(b);
```

```
4
7
```

Immutability

It can't be stressed enough that tuples are immutable. To drive that point home:

```python
In [14]: t
```

```
Out[14]: ('one', 2, 'one')
```

```python
In [21]: t[0]= 'change'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-21-fab90dbf634c> in <module>
----> 1 t[0]= 'change';

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

Because of this immutability, tuples can't grow. Once a tuple is made we can not add to it.

```python
In [10]: t.append('nope')
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[10], line 1
----> 1 t.append('nope')

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

```python
In [11]: t.extend('nope')
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[11], line 1
----> 1 t.extend('nope')

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

```
In [17]:    a = 3
            type(a)
```

Out[17]:   int

```
In [18]:    a = float(a)
            type(a)
```

Out[18]:   float

```
In [19]:    a
```

Out[19]:   3.0

```
In [23]:    t1
```

Out[23]:   (1, 2, 3)

```
In [24]:    t1.append(4)
```

AttributeError   Traceback (most recent call last)
Cell In[24], line 1
----> **1** t1.append(4)

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

```
In [16]:    Lst = list(t1)
            Lst
```

Out[16]:   [1, 2, 3]

```
In [25]:    Lst.append(4)
            Lst
```

Out[25]:   [1, 2, 3, 4]

```
In [20]:    Lst2 = [5, 8, 9]
            Lst2
```

Out[20]:   [5, 8, 9]

```
In [22]:    t3 = tuple(Lst2)
            t3
```

Out[22]:   (5, 8, 9)

When to use Tuples

You may be wondering, "Why bother using tuples when they have fewer available methods?" 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.

You should now be able to create and use tuples in your programming as well as have an understanding of their immutability.

Up next Files!

In [ ]: