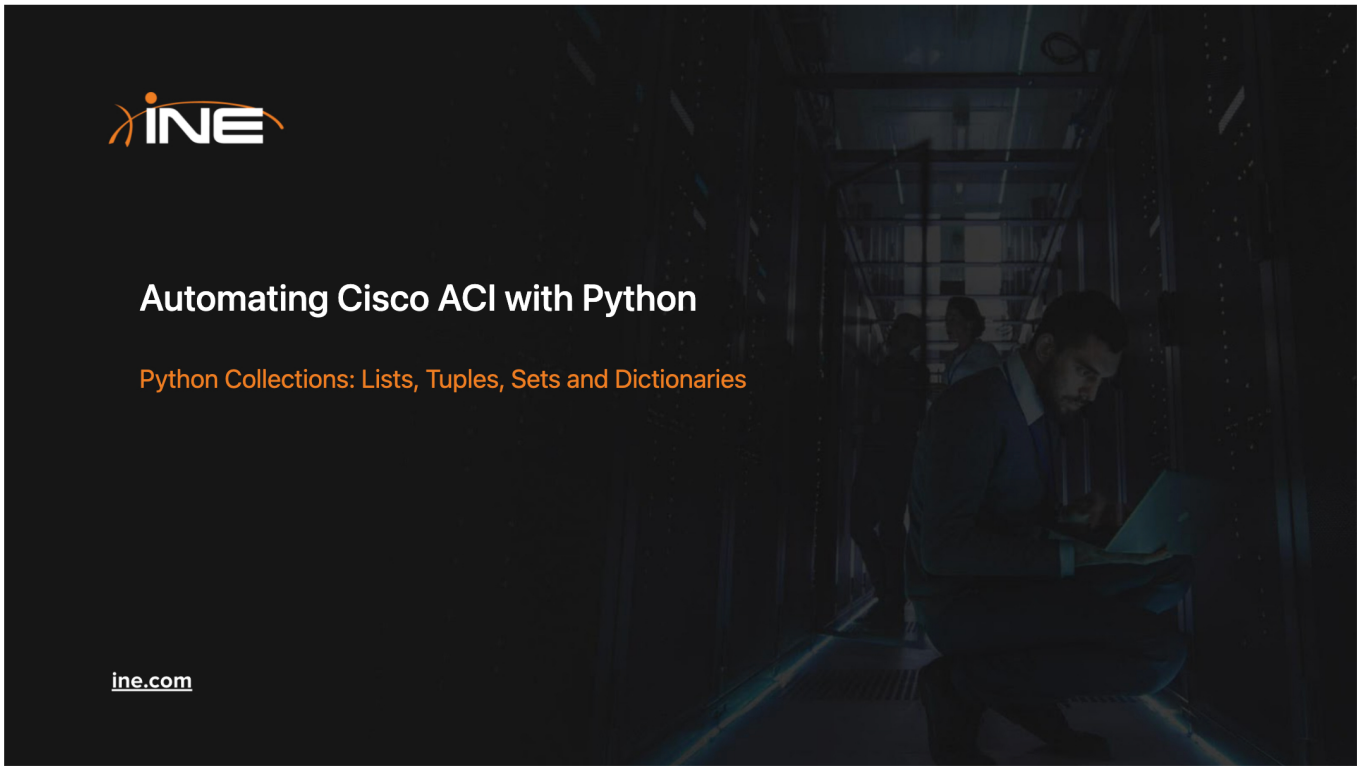

Python has multiple versatile collection types, each with different features and capabilities. These are the most common collections we'll explore:

- Lists
- Tuples
- Dictionaries
- Sets

Even though they all have different capabilities, there is one common property to all of them, and it's that Python collections are heterogeneous, that is, you can mix multiple types. That **doesn't mean we should** mix types, usually it's better to have a consistent collection. But it's still possible.

Lists

Lists (and other collections) are used to store *collections* (sorry, captain obvious) of objects. **Lists** are the most commonly used Python collection.

Strings and lists share many similarities with each other. After all, they're both sequences. A ***List is a sequence of objects***, and a ***String is a sequence of characters***.

Python lists are created using square brackets (`[]`).

Examples:

In [115…
```
animal_1 = 'dog'
animal_2 = 'cat'
animal_3 = 'duck'
```

In [116…
```
animal_1
```

Out[116… `'dog'`

In [117…
```
type(animal_1)
```

Out[117… `str`

In [118…
```
animals = ['dog', 'cat', 'duck']
```

In [119…
```
animals
```

Out[119… `['dog', 'cat', 'duck']`

In [120…
```
type(animals)
```

Out[120… `list`

In [121…
```
numbers = [5, 10, 12, 2]
```

In [122…
```
numbers
```

Out[122…
```
[5, 10, 12, 2]
```

In [123…
```
type(numbers)
```

Out[123…
```
list
```

In [124…
```
booleans = [True, True, False, True]

booleans
```

Out[124…
```
[True, True, False, True]
```

In [125…
```
type(booleans)
```

Out[125…
```
list
```

Strings to list

In [26]:
```
long_string = "This is my super long string"
```

In [27]:
```
long_string.split(" ")
```

Out[27]:
```
['This', 'is', 'my', 'super', 'long', 'string']
```

In [33]:
```
super_long_string = "Bibendum ad ac mus senectus donec mauris ligula habitant, platea venenatis fringilla auctor sem per habitasse potenti
```

In [34]:
```
super_long_string
```

Out[34]:
```
'Bibendum ad ac mus senectus donec mauris ligula habitant, platea venenatis fringilla auctor sem per habitasse potenti vel, libero digniss
im rutrum in porttitor tincidunt eleifend. Curabitur orci massa mattis eget pellentesque fermentum quam pretium libero venenatis, consequa
t aenean taciti semper viverra in nunc velit mollis. Mollis rhoncus leo phasellus litora cubilia id mus, senectus quis per rutrum nascetur
faucibus nec platea, commodo vehicula tempus tempor luctus suspendisse parturient, non facilisis taciti natoque habitant malesuada. Partur
ient accumsan purus vulputate maecenas facilisis nisl sociis viverra mollis quam arcu leo augue, tempus urna aenean porta dapibus sociosqu
mattis etiam rhoncus condimentum dignissim.'
```

In [36]:
```
super_long_string.split(".")
```

```
['Bibendum ad ac mus senectus donec mauris ligula habitant, platea venenatis fringilla auctor sem per habitasse potenti vel, libero dignis
sim rutrum in porttitor tincidunt eleifend', ' Curabitur orci massa mattis eget pellentesque fermentum quam pretium libero venenatis, cons
equat aenean taciti semper viverra in nunc velit mollis', ' Mollis rhoncus leo phasellus litora cubilia id mus, senectus quis per rutrum n
ascetur faucibus nec platea, commodo vehicula tempus tempor luctus suspendisse parturient, non facilisis taciti natoque habitant malesuad
a', ' Parturient accumsan purus vulputate maecenas facilisis nisl sociis viverra mollis quam arcu leo augue, tempus urna aenean porta dapi
bus sociosqu mattis etiam rhoncus condimentum dignissim', '']
```

In [37]:
```
len(super_long_string.split("."))
```

Out[37]:
```
5
```

Heterogeneous

All Python collections are heterogeneous; that means, you can store any type of value that you want in them:

In [126…
```
my_list = ["Hello", 3, True, 2.5]
```

In [127…
```
my_list
```

Out[127…
```
['Hello', 3, True, 2.5]
```

As you can see in the previous example, we can mix strings, with integers or booleans in the same Python list. You can store any type of object in a list (even other lists 😱).

> Even though we *CAN* store any type of element, it doesn't mean we *SHOULD* do it.

Mixing different types of elements in a list makes our less predictable and errors are more likely to occur.

We're obligated to tell you that lists don't have a limitation of types, but you'll see that good code will keep consistent the data types used in collections.

Ordered

Lists are *ordered sequences*. That means that they'll "remember" the order that you chose for the elements.

If we create a list with `"John"` first and `"Jane"` second, that order will be preserved:

In [128…
```
['John Doe', 'Jane Doe']
```

Out[128…
```
['John Doe', 'Jane Doe']
```

We know that this *"ordered"* trait might sound a little bit obvious. But if we're mention it, is because you'll see later that other collections will **NOT** respect order.

Mutable

You'll be able to change (*mutate*) lists: add, remove or update elements.

As I mentioned before, if we're pointing out this trait, is because you'll see other collections that will be immutable (you won't be able to change them).

In [129…
```
l = ['a', 'b', 'c']
l
```

Out[129…
```
['a', 'b', 'c']
```

Adding new items

In [130…
```
l.append('d')
```

In [131…
```
l
```

Out[131…
```
['a', 'b', 'c', 'd']
```

Removing items

In [132…
```
l.remove('b')
```

In [133…
```
l
```

Out[133…
```
['a', 'c', 'd']
```

Updating items by position

> Remember that in Python first position is `0`.

In [134…
```
l[2] = 'x'
```

In [135…
```
l
```

Out[135…
```
['a', 'c', 'x']
```

Getting list/string length

In [136…
```
my_list = ['dog', 'cat', 'duck']
my_string = 'Learning Python with INE'
```

In [137…
```
len(my_list)
```

Out[137…
```
3
```

In [138…
```
len(my_string)
```

Out[138…
```
24
```

Indexing and Slicing

Accessing individual elements and slicing lists will work in the same way as with strings.

> Remember that in Python first position is `0`.

In [38]:
```
l = ['A', 'B', 'C', 'D', 'E']
#    0    1    2    3    4     (Regular Indexes)
#   -5   -4   -3   -2   -1     (Negative Indexes)
```

Indexing

Getting / reading individual elements by their integer position.

In [140…
```
my_list[0]
```

Out[140…
```
'dog'
```

In [141…
```
my_list[1]
```

Out[141…
```
'cat'
```

In [142…
```
my_list[2]
```
Out[142…
```
'duck'
```
In [143…
```
my_list[-1]
```
Out[143…
```
'duck'
```
In [144…
```
my_list[len(my_list) - 1]
```
Out[144…
```
'duck'
```
In [145…
```
my_list[2] == my_list[len(my_list) - 1] == my_list[-1] == 'duck'
```
Out[145…
```
True
```

Slicing

Getting a "sub-list", a range of objects.

In [55]:
```
letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g']
```
In [147…
```
letters[0:5]
```
Out[147…
```
['a', 'b', 'c', 'd', 'e']
```
In [148…
```
letters[2:7]
```
Out[148…
```
['c', 'd', 'e', 'f', 'g']
```
In [149…
```
letters[0:2]
```
Out[149…
```
['a', 'b']
```
In [150…
```
letters[2:]
```
Out[150…
```
['c', 'd', 'e', 'f', 'g']
```
In [151…
```
letters[:5]
```
Out[151…
```
['a', 'b', 'c', 'd', 'e']
```
In [152…
```
letters[-3:-1]
```
Out[152…
```
['e', 'f']
```
In [153…
```
letters[:]
```
Out[153…
```
['a', 'b', 'c', 'd', 'e', 'f', 'g']
```
In [154…
```
letters[::]
```
Out[154…
```
['a', 'b', 'c', 'd', 'e', 'f', 'g']
```
In [155…
```
letters[::1]
```
Out[155…
```
['a', 'b', 'c', 'd', 'e', 'f', 'g']
```
In [156…
```
letters[::2]
```
Out[156…
```
['a', 'c', 'e', 'g']
```

Count occurrences inside the list:

In [59]:
```
letters.count('b')
```
Out[59]:
```
1
```

Tuples

Tuples are very similar to lists, but with a huge difference: **they're immutable**. That means, once a tuple is created, it can't be further modified:

In [47]:
```
t = (3, 'Hello World', True)
```

Indexing them works in the same way:

In [48]:
```
t[0]
```

Out[48]:
```
3
```

In [49]:
```
t[-1]
```

Out[49]:
```
True
```

In [50]:
```
'Hello World' in t
```

Out[50]:
```
True
```

But there's no way of modifying them.

Dictionaries

Dictionaries are map-like collections that store values under a user-defined key. The key must be an immutable object; we usually employ strings for keys. Dictionaries are mutable, and more importantly, **unordered**.

In [2]:
```
user = {
    "name": "Mary Smith",
    "email": "mary@example.com",
    "age": 30,
    "subscribed": True
}
```

In [3]:
```
user
```

Out[3]:
```
{'name': 'Mary Smith',
 'email': 'mary@example.com',
 'age': 30,
 'subscribed': True}
```

Access is by key, also using square brackets:

In [4]:
```
user['email']
```

Out[4]:
```
'mary@example.com'
```

In [5]:
```
'age' in user
```

Out[5]:
```
True
```

In [6]:
```
'last_name' in user
```

Out[6]:
```
False
```

In [7]:
```
user.keys()
```

Out[7]:
```
dict_keys(['name', 'email', 'age', 'subscribed'])
```

In [10]:
```
for k in user.keys():
    print(k)
```

```
name
email
age
subscribed
```

In [11]:
```
user.values()
```

Out[11]:
```
dict_values(['Mary Smith', 'mary@example.com', 30, True])
```

In [12]:
```
for v in user.values():
    print(v)
```

```
Mary Smith
mary@example.com
30
True
```

Sets

Sets are unordered collection which the unique characteristic that they only contain unique elements:

In [19]:
```
s = {3, 1, 3, 7, 9, 1, 3, 1}
```

In [20]:
```
s
```

Out[20]:
```
{1, 3, 7, 9}
```

Adding elements is done with the `add` method:

In [21]:
```
s.add(10)
```

In [22]:
```
s
```

Out[22]:
```
{1, 3, 7, 9, 10}
```

Removing elements can be done with `pop()`:

In [23]:
```
s.pop()
```

Out[23]:
```
1
```

In [24]:
```
s
```

Out[24]:
```
{3, 7, 9, 10}
```

Membership operation:

In [25]:
```
9 in s
```

Out[25]:
```
True
```

Iterating collections

We can use Python's `for` loop to iterate over collections. Let's see a quick example as an intro to what we will be seeing next:

In [44]:
```
l = [3, 'Hello World', True]
```

In [45]:
```
for elem in l:
    print(elem)
```

```
3
Hello World
True
```

In [94]:
```
user
```

Out[94]:
```
{'name': 'Mary Smith',
 'email': 'mary@example.com',
 'age': 30,
 'subscribed': True}
```

In [42]:
```
for key in user.keys():
    print(key.title(), '=>', user[key])
```

```
Name => Mary Smith
Email => mary@example.com
Age => 30
Subscribed => True
```

In [43]:
```
for key in user:
    print(key.title(), '=>', user[key])
```

```
Name => Mary Smith
Email => mary@example.com
Age => 30
Subscribed => True
```

In [98]:
```
for key, value in user.items():
    print(key, value)
```

```
name Mary Smith
email mary@example.com
age 30
subscribed True
```

In [99]:
```
for i in range(5):
    print(i)
```

```
0
1
2
3
4
```

Comprehensions

In [61]:
```
initial = list(range(100))

print(initial)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36,
37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 7
1, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99]
```

In [66]:
```
new_list = []
for elem in initial:
    new_list.append(elem*2)

print(new_list)
```

```
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46, 48, 50, 52, 54, 56, 58, 60, 62, 64, 66, 68, 7
0, 72, 74, 76, 78, 80, 82, 84, 86, 88, 90, 92, 94, 96, 98, 100, 102, 104, 106, 108, 110, 112, 114, 116, 118, 120, 122, 124, 126, 128, 130,
132, 134, 136, 138, 140, 142, 144, 146, 148, 150, 152, 154, 156, 158, 160, 162, 164, 166, 168, 170, 172, 174, 176, 178, 180, 182, 184, 18
6, 188, 190, 192, 194, 196, 198]
```

In [69]:
```
new_list = [n*2 for n in initial]

print(new_list)
```

```
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46, 48, 50, 52, 54, 56, 58, 60, 62, 64, 66, 68, 7
0, 72, 74, 76, 78, 80, 82, 84, 86, 88, 90, 92, 94, 96, 98, 100, 102, 104, 106, 108, 110, 112, 114, 116, 118, 120, 122, 124, 126, 128, 130,
132, 134, 136, 138, 140, 142, 144, 146, 148, 150, 152, 154, 156, 158, 160, 162, 164, 166, 168, 170, 172, 174, 176, 178, 180, 182, 184, 18
6, 188, 190, 192, 194, 196, 198]
```

In [70]:
```
new_list = [n for n in initial if n % 7 == 0]

print(new_list)
```

```
[0, 7, 14, 21, 28, 35, 42, 49, 56, 63, 70, 77, 84, 91, 98]
```

In [72]:
```
new_list = [n*3 for n in initial if n % 7 == 0 or n % 5 == 0]

print(new_list)
```

```
[0, 15, 21, 30, 42, 45, 60, 63, 75, 84, 90, 105, 120, 126, 135, 147, 150, 165, 168, 180, 189, 195, 210, 225, 231, 240, 252, 255, 270, 273,
285, 294]
```

In [77]:
```
other_list = [1, 2, 1, 0]
new_set = {n for n in other_list}

print(new_set)
```

```
{0, 1, 2}
```

In [78]:
```
other_list = [1, 2, 1, 0]
new_set = {n*2 for n in other_list}

print(new_set)
```

```
{0, 2, 4}
```

In [86]:
```
new_dict = {n:n**3 for n in initial if n < 10}

print(new_dict)
```

```
{0: 0, 1: 1, 2: 8, 3: 27, 4: 64, 5: 125, 6: 216, 7: 343, 8: 512, 9: 729}
```