



Collection Types

- List
- Tuple
- Dictionary
- Set





Lists

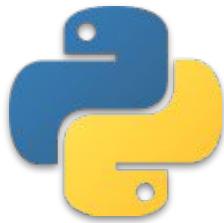


Table of Contents



- ▶ Introduction
- ▶ Creating a List
- ▶ Basic Operations with Lists



1

Introduction

```
fruit = [ 'apple', 'orange', 'Banana' ]  
best = [ 'Clarusway' ]  
list()
```



What do you know
about [lists]?



Students, write your response!

Introduction



▶ What is a collection type?

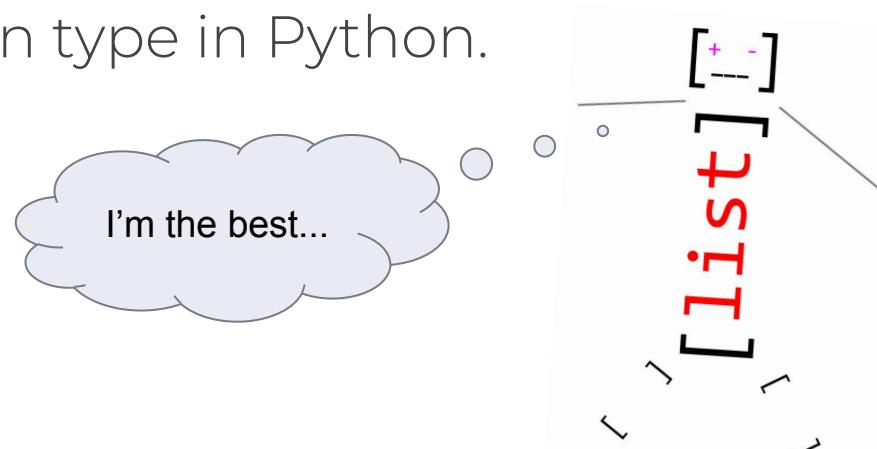
- ▷ There are various **collection types** in Python. While types such as `int` and `str` hold a single value, collection types hold *multiple values*.
- ▷ In your programs, you usually need to group several items to render as a single object. We use **collection types** of data to do this job.

Introduction



► What is a **list** ?

- ▷ One of the most useful collections in Python is a **list**.
- ▷ In Python, a **list** is only an ordered collection of valid Python values.
- ▷ The **list** type is probably the most commonly used collection type in Python.





2

Creating a list

Creating a list

- ▶ A **list** can be created by enclosing values, separated by commas, in square brackets  [].
- ▶ Another way to create a **list** is to call the **list()** function.

- []
- **list()**

Creating a list

- ▶ A **list** can be created by enclosing values, separated by commas, in square brackets  [].
- ▶ Another way to create a **list** is to call the **list()** function.

- []
- **list()**



```
list_1 = ['h', 'a', 'p', 'p', 'y']
word = 'happy'
list_2 = list(word)
print(list_1)
print(list_2)
```

What is the output? Try to figure out in your mind...

Creating a list

- ▶ A **list** can be created by enclosing values, separated by commas, in square brackets  [].
- ▶ Another way to create a **list** is to call the **list()** function.

- []
- **list()**



```
list_1 = ['h', 'a', 'p', 'p', 'y']
word = 'happy'
list_2 = list(word)
print(list_1)
print(list_2)
```

```
['h', 'a', 'p', 'p', 'y']
['h', 'a', 'p', 'p', 'y']
```

Creating a list (review of pre-class)

- Here is another example of creating a **list**:

```
1 country = ['USA', 'Brasil', 'UK', 'Germany', 'Turkey', 'New Zealand']
2
3 print(country)
4
```

Creating a list (review of pre-class)

Here is another example of creating a **list**:

```
1 country = ['USA', 'Brasil', 'UK', 'Germany', 'Turkey', 'New Zealand']
2
3 print(country)
4
```

```
1 ['USA', 'Brasil', 'UK', 'Germany', 'Turkey', 'New Zealand']
2
```

Creating a **list** (review of pre-class)

Here is another example of creating a **list**:

```
1 country = ['USA', 'Brasil', 'UK', 'Germany', 'Turkey', 'New Zealand']
2
3 print(country)
4
```

```
1 ['USA', 'Brasil', 'UK', 'Germany', 'Turkey', 'New Zealand']
2
```



Tips:

- All the country names are printed in the same order as they were stored in the list because lists are **ordered**.

Creating a `list` (review of pre-class)

- ▶ Here is another example of creating a `list` using `list()`.
- ▶ We can do it when we want to create a `list` from an iterable object: that is, type of object whose elements you can import individually. The `lists` are iterable like other collections and `string` types.

```
1 string_1 = 'I quit smoking'  
2  
3 new_list_1 = list(string_1) # we created multi element list  
4 print(new_list_1)  
5  
6 new_list_2 = [string_1] # this is a single element list  
7 print(new_list_2)  
8
```

What is the output? Try to figure out in your mind...



Students, write your response!

REINVENT YOURSELF

Creating a list (review of pre-class)

- ▶ Here is another example of creating a `list` using `list()`.
- ▶ We can do it when we want to create a `list` from an iterable object: that is, type of object whose elements you can import individually. The `lists` are iterable like other collections and `string` types.

```
1 string_1 = 'I quit smoking'  
2  
3 new_list_1 = list(string_1) # we created multi element list  
4 print(new_list_1)  
5  
6 new_list_2 = [string_1] # this is a single element list  
7 print(new_list_2)  
8
```

```
1 ['I', ' ', 'q', 'u', 'i', 't', ' ', 's', 'm', 'o', 'k', 'i', 'n', 'g']  
2 ['I quit smoking']  
3
```

Creating a list

Tips:

- Note that, using `list()` function, all characters of `string_1` including spaces was moved into a `new_list_1`.
- If you noticed, **lists** can contain **more than one** of the **same** value.

```
1 string_1 = 'I quit smoking'  
2  
3 new_list_1 = list(string_1) # we created multi element list  
4 print(new_list_1)  
5  
6 new_list_2 = [string_1] # this is a single element list  
7 print(new_list_2)  
8
```

```
1 ['I', ' ', 'q', 'u', 'i', 't', ' ', 's', 'm', 'o', 'k', 'i', 'n', 'g']  
2 ['I quit smoking']  
3
```

Creating a list

Tips:

- Note that, using `list()` function, all characters of `string_1` including spaces was moved into a new_`list_1`.
- If you noticed, **lists** can contain **more than one** of the **same** value.

```
1 string_1 = 'I quit smoking'  
2  
3 new_list_1 = list(string_1) # we created multi element list  
4 print(new_list_1)  
5  
6 new_list_2 = [string_1] # single element list  
7 print(new_list_2)  
8
```

! It has only one element.

single element list

! It has 14 elements.

```
1 ['I', ' ', 'q', 'u', 'i', 't', ' ', 's', 'm', 'o', 'k', 'i', 'n', 'g']  
2 ['I quit smoking']  
3
```

Creating a list

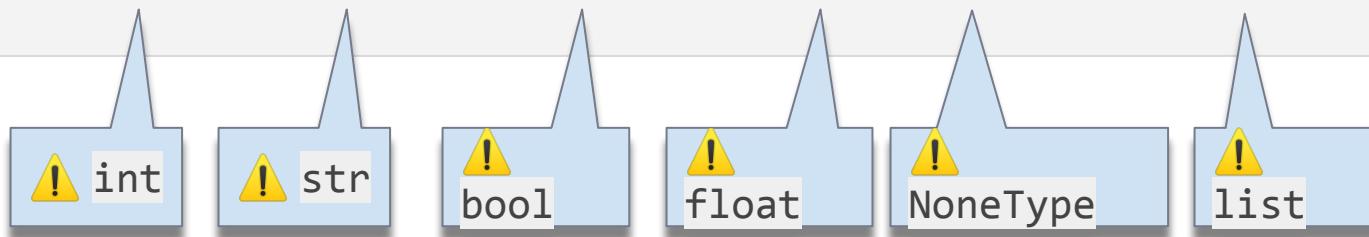
- ▶ The components of a **list** are not limited to a single data type, given that Python is a dynamic language.
- ▶ Here is an example :

```
mixed_list = [11, 'Joseph', False, 3.14, None, [1, 2, 3]]
```

Creating a list

- ▶ The components of a **list** are not limited to a single data type, given that Python is a dynamic language.
- ▶ Here is an example :

```
mixed_list = [11, 'Joseph', False, 3.14, None, [1, 2, 3]]
```



Creating a list



- ▶ Task:
 - ▶ Try to figure out the output of these list operations in your mind.

```
my_list = ['Joseph', 'Clarusway', 2020]
new_list1 = list(my_list) # what will be the output?
new_list2 = [my_list]   # what will be the output?
print(new_list1)
print(len(new_list1))  # what is the length of the variable?

print(new_list2)
print(len(new_list2))
```



Students, write your response!

Pear Deck Interactive Slide
Do not remove this bar

Creating a list

- ▶ The output:

```
['Joseph', 'Clarusway', 2020]  
3  
[['Joseph', 'Clarusway', 2020]]  
1
```

Creating a list

- ▶ Task:
 - ▷ Create a list from **string** value of "2023's perfect".
 - ▷ Use both **list()** function and square brackets [].



Creating a list

- ▶ The code :

```
my_list1 = ["2023's perfect"]  
my_list2 = list("2023's perfect")
```

```
print(my_list1)  
print(my_list2)
```

Each item/char including spaces
came into the list

```
["2023's perfect"]  
['2','0','2','0','',' ','s',' ',' ','p','e','r','f','e','c','t']
```



3

Basic Operations with Lists

Basic Operations with lists

- ▶ We can also measure the length of the list by using `len()` function. Take a look at this pre-class example



```
1 city = ['Addis Ababa', 'Istanbul', 'New York', 'Seoul', 'Stockholm', 'Sydney']
2 print(len(city))
3
```

```
1 6
2
```

Basic Operations with lists



- In most cases, we'll have to make an empty **list** to fill it later with the data you want.

How to create an empty list?



Students choose an option

REINVENT YOURSELF

Pear Deck Interactive Slide
Do not remove this bar

Basic Operations with lists

- In most cases, we'll have to make an empty `list` to fill it later with the data you want.

```
empty_list_1 = []
```

```
empty_list_2 = list()
```

Two methods for creating
an empty list.

Basic Operations with lists



- ▶ We can add an element into a list using `.append()` or `.insert()` methods.
- ▶ `.append()` appends an object to the end of a `list`.

- `.append()`
- `.insert()`

Basic Operations with lists



- ▶ We can add an element into a list using `.append()` or `.insert()` methods.
- ▶ `.append()` appends an object to the end of a `list`.

- `.append()`
- `.insert()`



```
numbers = [1, 4, 7]
numbers.append(9)
numbers.append('9')
print(numbers)
```

What is the output? Try to figure out in your mind...

Basic Operations with lists



- ▶ We can add an element into a list using `.append()` or `.insert()` methods.
- ▶ `.append()` appends an object to the end of a **list**.

- `.append()`
- `.insert()`



```
numbers = [1, 4, 7]
numbers.append(9)
numbers.append('9')
print(numbers)
```

```
[1, 4, 7, 9, '9']
```

Basic Operations with lists

- ▶ Take a look at the example 

```
1 empty_list = []
2 empty_list.append(6666)
3 empty_list.append('Multiverse')
4 empty_list.append([0])
5
6 print(empty_list)
7
8 |
```

What is the output? Try to figure out in your mind...



USWY[®]
Students, write your response!

REINVENT YOURSELF

Pear Deck Interactive Slide
Do not remove this bar

Basic Operations with lists

- ▶ Take a look at the example 

```
1 empty_list = []
2 empty_list.append(6666)
3 empty_list.append('Multiverse')
4 empty_list.append([0])
5
6 print(empty_list)
7
8 |
```

Output

```
[6666, 'Multiverse', [0]]
```

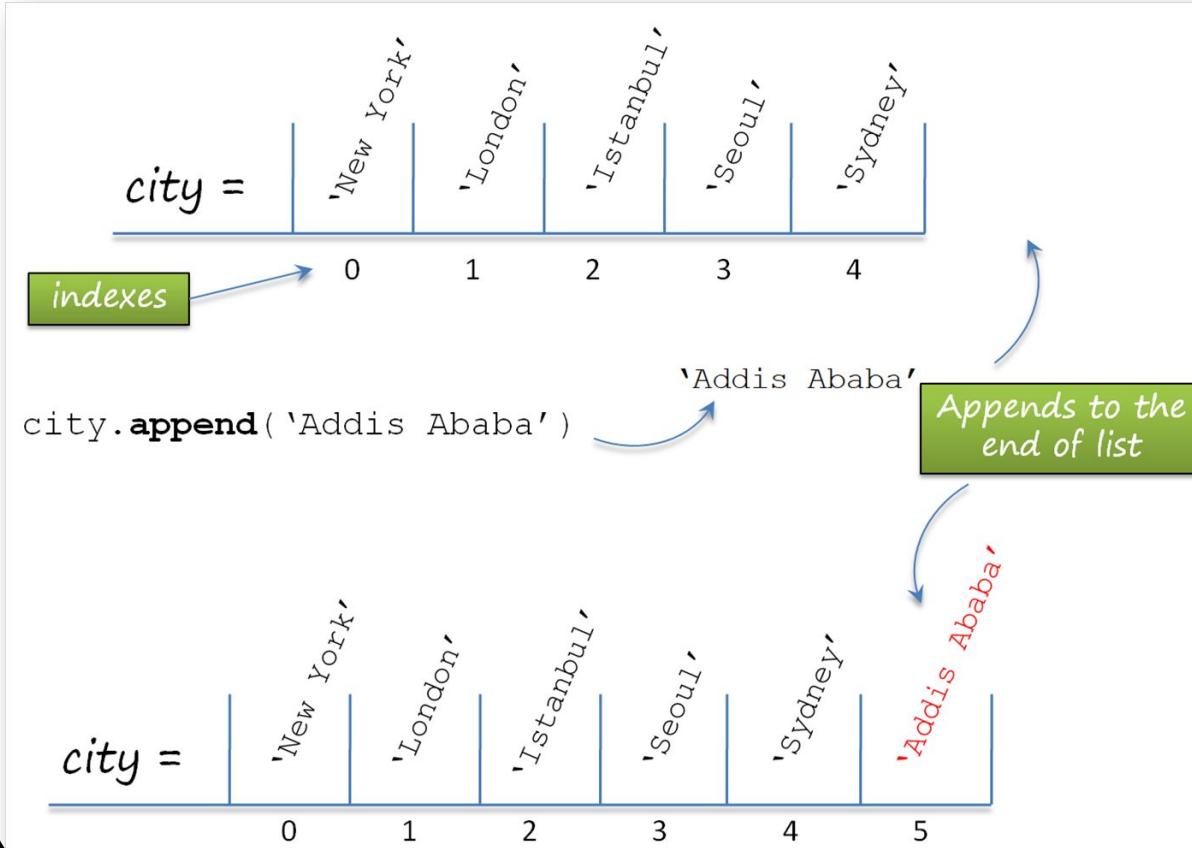
Basic Operations with lists

- ▶ Here's the pre-class example



```
1 city = ['New York', 'London', 'Istanbul', 'Seoul', 'Sydney']
2 city.append('Addis Ababa')
3
4 print(city)
5
```

Basic Operations with lists



Basic Operations with lists

- ▶ Here's an another example 

```
1 city = ['New York', 'London', 'Istanbul', 'Seoul', 'Sydney']
2 city.append('Addis Ababa')
3
4 print(city)
5
```

```
1 ['New York', 'London', 'Istanbul', 'Seoul', 'Sydney', 'Addis Ababa']
2
```

Basic Operations with lists



▶ Task :

- ▷ Create an empty `list` and then collect the `int` numbers (1 - 4) one by one into the `list` you created using `.append()` method.

Basic Operations with lists

- The code can be like :

```
numbers = []
numbers.append(1)
numbers.append(2)
numbers.append(3)
numbers.append(4)
```

```
print(numbers)
```

```
[1, 2, 3, 4]
```

Basic Operations with lists



- ▶ `.insert()` adds a new object to the `list` at a specific (given) `index`.

- `.append()`
- `.insert()`

Basic Operations with lists



- ▶ `.insert()` adds a new object to the `list` at a specific (given) index.

- `.append()`
- `.insert()`



```
numbers = [1, 4, 7]
numbers.insert(2, 9)
print(numbers)
numbers.insert(2, 6)
print(numbers)
```

What is the output? Try to figure out in your mind...



Basic Operations with lists



- ▶ `.insert()` adds a new object to the `list` at a specific (given) index.

- `.append()`
- `.insert()`



```
numbers = [1, 4, 7]
numbers.insert(2, 9)
print(numbers)
numbers.insert(2, 6)
print(numbers)
```

Adds into index '2'

[1, 4, 9, 7]
[1, 4, 6, 9, 7]

0	1	2	3	4



Basic Operations with lists

- ▶ Consider this pre-class example



```
1 city = ['New York', 'London', 'Istanbul', 'Seoul', 'Sydney', 'Addis Ababa']
2 city.insert(2, 'Stockholm')
3
4 print(city)
5
```

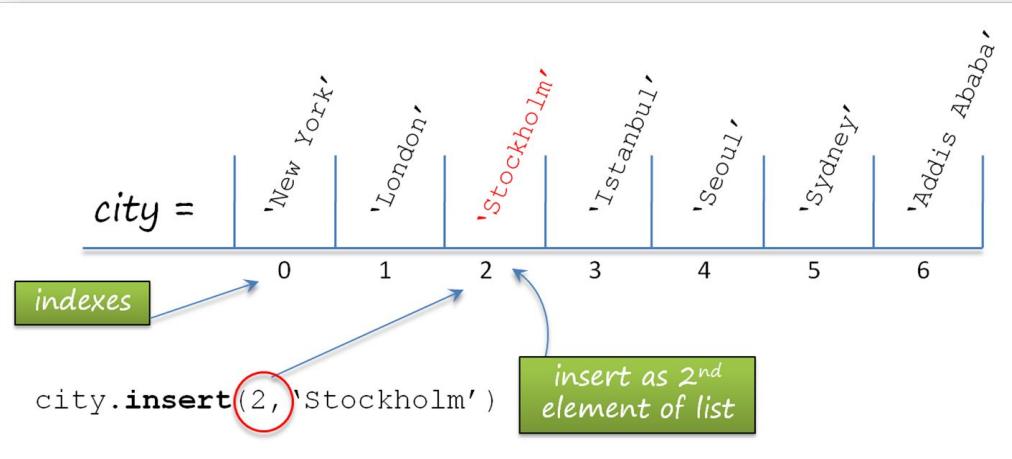
Basic Operations with lists

- ▶ Consider this pre-class example



```
1 city = ['New York', 'London', 'Istanbul', 'Seoul', 'Sydney', 'Addis Ababa']
2 city.insert(2, 'Stockholm')
3
4 print(city)
5
```

```
1 ['New York', 'London', 'Stockholm', 'Istanbul', 'Seoul', 'Sydney', 'Addis Ababa']
2
```



Basic Operations with lists



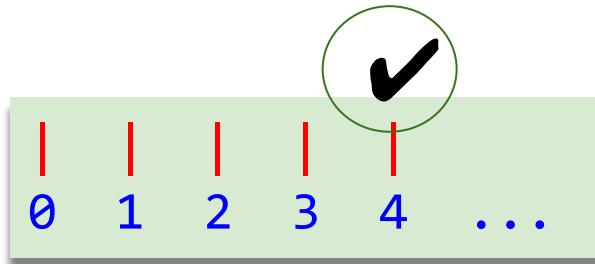
▶ Task :

- ▷ Create a **list** which consists of the **int** numbers (1 - 4) and then insert number **5** at the **end** of the **list** using **.insert()** method.

Basic Operations with lists



- The code can be like :



```
numbers = [1, 2, 3, 4]  
numbers.insert(4, 5)
```

```
print(numbers)
```

index

```
[1, 2, 3, 4, 5]
```

Basic Operations with lists



- ▶ We can remove and sort the elements of the **list**.
- ▶ **.remove()** removes the elements from the **list**.

- **.remove()**
- **.sort()**

Basic Operations with lists



- We can remove and sort the elements of the **list**.
- **.remove()** removes the elements from the **list**.

- **.remove()**
- **.sort()**



```
numbers = [1, 4, 7, 9]
numbers.remove(7)
print(numbers)
```



Basic Operations with lists



- ▶ We can remove and sort the elements of the **list**.
- ▶ **.remove()** removes the elements from the **list**.

- **.remove()**
- **.sort()**



```
numbers = [1, 4, 7, 9]
numbers.remove(7)
print(numbers)
```

```
[1, 4, 9]
```

Basic Operations with lists

- ▶ Consider this pre-class example



```
1 city = ['New York', 'London', 'Stockholm', 'Istanbul', 'Seoul', 'Sydney', 'Addis Ababa']
2 city.remove('London')
3 print(city) # we have deleted 'London'
4
```

Basic Operations with lists



- ▶ Consider this pre-class example



```
1 city = ['New York', 'London', 'Stockholm', 'Istanbul', 'Seoul', 'Sydney', 'Addis Ababa']
2 city.remove('London')
3 print(city) # we have deleted 'London'
4
```

```
1 ['New York', 'Stockholm', 'Istanbul', 'Seoul', 'Sydney', 'Addis Ababa']
2
```

Basic Operations with lists



- ▶ `.sort()` sorts the elements in the `list`.

- `.remove()`
- `.sort()`

Basic Operations with lists



- ▶ `.sort()` sorts the elements in the `list`.

- `.remove()`
- `.sort()`



```
numbers = [4, 1, 9, 7]
numbers.sort()
print(numbers)
```



USWY[®]
Students, write your response!

REINVENT YOURSELF

Pear Deck Interactive Slide
Do not remove this bar



Basic Operations with lists

- ▶ `.sort()` sorts the elements in the `list`.

- `.remove()`
- `.sort()`



```
numbers = [4, 1, 9, 7]
numbers.sort()
print(numbers)
```

```
[1, 4, 7, 9]
```

Basic Operations with lists

- ▶ Here's the pre-class example



```
1 city = ['New York', 'Stockholm', 'Istanbul', 'Seoul', 'Sydney', 'Addis Ababa']
2 city.sort() # lists the items in alphabetical order
3 print(city)
4
```

Basic Operations with lists

- ▶ Here's an example



```
1 city = ['New York', 'Stockholm', 'Istanbul', 'Seoul', 'Sydney', 'Addis Ababa']
2 city.sort() # lists the items in alphabetical order
3 print(city)
4
```

```
1 ['Addis Ababa', 'Istanbul', 'New York', 'Seoul', 'Stockholm', 'Sydney']
2
```

Basic Operations with lists



- ▶ `.sort()` sorts the elements in the `list`.

```
mix_list = ['d', 1, 'a', 7]
mix_list.sort()
print(mix_list)
```

What is the output? Try to figure out in your mind...



Students, write your response!

REINVENT YOURSELF

Basic Operations with lists



- ▶ `.sort()` sorts the elements in the `list`.

```
mix_list = ['d', 1, 'a', 7]
mix_list.sort()
print(mix_list)
```

```
TypeError
last)
<ipython-input-7-ad44188425eb> in <module>
      1 mix_list = ['d', 1, 'a', 7]
----> 2 mix_list.sort()
      3 print(mix_list)
```

```
Traceback (most recent call
```

```
TypeError: '<' not supported between instances of 'int' and 'str'
```



Basic Operations with lists

- ▶ `.sort()` sorts the elements in the `list`.

```
mix_li  
mix_li  
print()  
  
TypeError:  
last)  
<ipython  
1 mix_list = [ 0 , 1 ,  a , 7 ]  
----> 2 mix_list.sort()  
3 print(mix_list)
```

The items to be sorted in the list should be the same type.

TypeError: '<' not supported between instances of 'int' and 'str'

recent call

Basic Operations with lists



- ▶ Sort these elements in the **lists**. Do not play with Playground, push your brains!..

```
list_1 = ['one', 'four', 'nine']
list_2 = ['*-', '@', 'False']
list_3 = [True, False]
list_4 = [[3], [44], [-12]]
list_5 = [[1, 3], [44, 0], [-12, 1]]
```

Basic Operations with lists



- Now! You can Play with Playgrounds. 😊

```
list_1 = ['one', 'four', 'nine']
list_2 = ['@', '*-', 'False']
list_3 = [True, False]
list_4 = [[3], [44], [-12]]
list_5 = [[1, 3], [44, 0], [-12, 1]]
```

Basic Operations with lists

- ▶ We can also measure the length of the list by using `len()` function. Take a look at this pre-class example



```
1 city = ['Addis Ababa', 'Istanbul', 'New York', 'Seoul', 'Stockholm', 'Sydney']
2 print(len(city))
3
```

Basic Operations with lists



- ▶ Calculate the length of these lists. Do not play with Playground, push your brains!..

```
list_1 = ['one', 'four', 'nine']
list_2 = ['*-', 4, 'False', 0]
list_3 = [True, False]
```

Circle how you are feeling:



Students, draw anywhere on this slide!

THANKS!

End of the Lesson (Lists)

next Lesson

Accessing Lists

click above

