APS106

UNIVERSITY OF TORONTO

tuples and sets.

Week 8 | Lecture 1 (8.1)

if nothing else, write **#cleancode**.

This Week's Content

- **Lecture 8.1**
  - **tuples and sets**
  - **Reading: 11**
- **Lecture 8.2**
  - dictionaries
  - Reading: 11
- **Lecture 8.3**
  - Review for Midterm 2 #jeopardy

Tuples

- Tuples are an ordered sequence of items similar to lists.
- Ordered Sequences:
  - Strings
  - Lists
  - `range()`
  - Tuples

Common Sequence Operations

Operation	Result	Notes
`x in s`	`True` if an item of *s* is equal to *x*, else `False`	(1)
`x not in s`	`False` if an item of *s* is equal to *x*, else `True`	(1)
`s + t`	the concatenation of *s* and *t*	(6)(7)
`s * n` or `n * s`	equivalent to adding *s* to itself *n* times	(2)(7)
`s[i]`	*i*th item of *s*, origin 0	(3)
`s[i:j]`	slice of *s* from *i* to *j*	(3)(4)
`s[i:j:k]`	slice of *s* from *i* to *j* with step *k*	(3)(5)
`len(s)`	length of *s*	
`min(s)`	smallest item of *s*	
`max(s)`	largest item of *s*	
`s.index(x[, i[, j]])`	index of the first occurrence of *x* in *s* (at or after index *i* and before index *j*)	(8)
`s.count(x)`	total number of occurrences of *x* in *s*	

```
birthday = (20,01,1985)
```

Tuples

- The general syntax of a tuple is as follows:

```
(expr1, expr2, ..., exprN)
```

- Tuples are represented with parentheses `()` while lists are represented by `[]`.
- To avoid ambiguity, a tuple with a single element is written as `(expr,)` to not be confused with arithmetic operations.
  - (1 + 1) / 2
  - (1) / 2

Open your notebook

Click Link:
1. Creating Tuples

Immutable

Once assigned, the tuple cannot be changed.

- Tuples are basically immutable lists meaning everything works as with lists excepts methods that modify the tuple.
  - `.append()`
  - `.sort()`
  - `.pop()`
- Immutable means that the item reference addresses contained in a tuple cannot be changed after the tuple has been created.
- You've seen this with strings (immutable sequence of characters).

Immutability Rules

1. Can only ever have 3 items.

2. Must always point to these references id's.

2. Must always be in this original order.

Immutable

Once assigned, the tuple cannot be changed.

```
birthday = (20,01,1985)
```

- Tuples are basically immutable lists meaning everything works as with lists excepts methods that modify the tuple.
  - `.append()`
  - `.sort()`
  - `.pop()`
- Immutable means that the item reference addresses contained in a tuple cannot be changed after the tuple has been created.
- You've seen this with strings (immutable sequence of characters).

Open your notebook

Click Link:
2. Tuples Are Immutable

Breakout Session 1

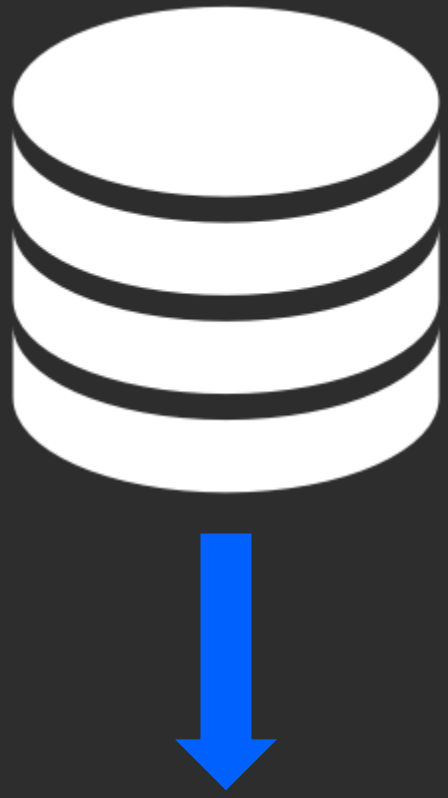
- Complete the exercises in the notebook.

Open your notebook

Click Link:
3. Breakout Session 1

Why Tuples?

- **Reason 1**
  - Tuples makes your code safer and less prone to bugs by providing write protection.
  - Consider that you're reading data from a database and saving it into memory.
  - Example: Imagine if you're telling the doctor what the symptoms are for a certain disease. If these symptoms were stored in a list, they could be changed, which could lead to negative outcomes for patients.



Why Tuples?

- **Reason 2**
  - Performance increase. Processing a tuple is faster than processing a list. Great for large data sets.
  - Since a tuple's size is fixed, it can be stored more compactly than lists which need to over-allocate to make `append()` operations efficient.

```
>>> sys.getsizeof((1, 2, 3, 4, 5))
88 bytes

>>> sys.getsizeof([1, 2, 3, 4, 5])
104 bytes
```

Why Tuples?

- **Reason 3**
  - You can always unpack tuples successfully because you always know how many items are in them (Immutability).

This will always work

```
data = (20,01,1985)
day, month, year = data
```

This will not always work

```
data = [20,01,1985]
day, month, year = data
```

Unpacking Tuples

- **Tuple Packing**
- The values on the right are ‘packed’ together in the tuple.
- `>>> record = ("Joe", 19, "CIV")`
- **Tuple Unpacking**
- The values in a tuple on the right are ‘unpacked’ into the variables on the left.
- `>>> name, age, studies = record`
- `>>> name`
- `‘Joe’`

Open your notebook

Click Link:
4. Unpacking Tuples

Tuples as `return` Values

- Functions can only return a single value, but by making that value a `tuple`, we can effectively group together as many values as we like (`tuple` packing), and `return` them together.

```
def func_name(parameters):
    return (expr1, expr2, ...
```

- When we call the function we can unpack the `tuple` into multiple variables.

```
var1, var2, ... = func_name(args)
```

Open your notebook

Click Link:
5. Tuples as return Values

Breakout Session 2

- In this Breakout Session, you'll loop through a collection of some of my favorite albums and print the content.

Open your notebook

Click Link:
6. Breakout Session 2

Sets

- A set **{exp1, exp2, ...}** is an unordered collection of distinct items that does not record element position or order of insertion.
- Accordingly, sets do not support indexing, slicing, or other sequence-like behavior.
- Their primary purpose is to hold distinct items: there are no duplicates in sets.

List

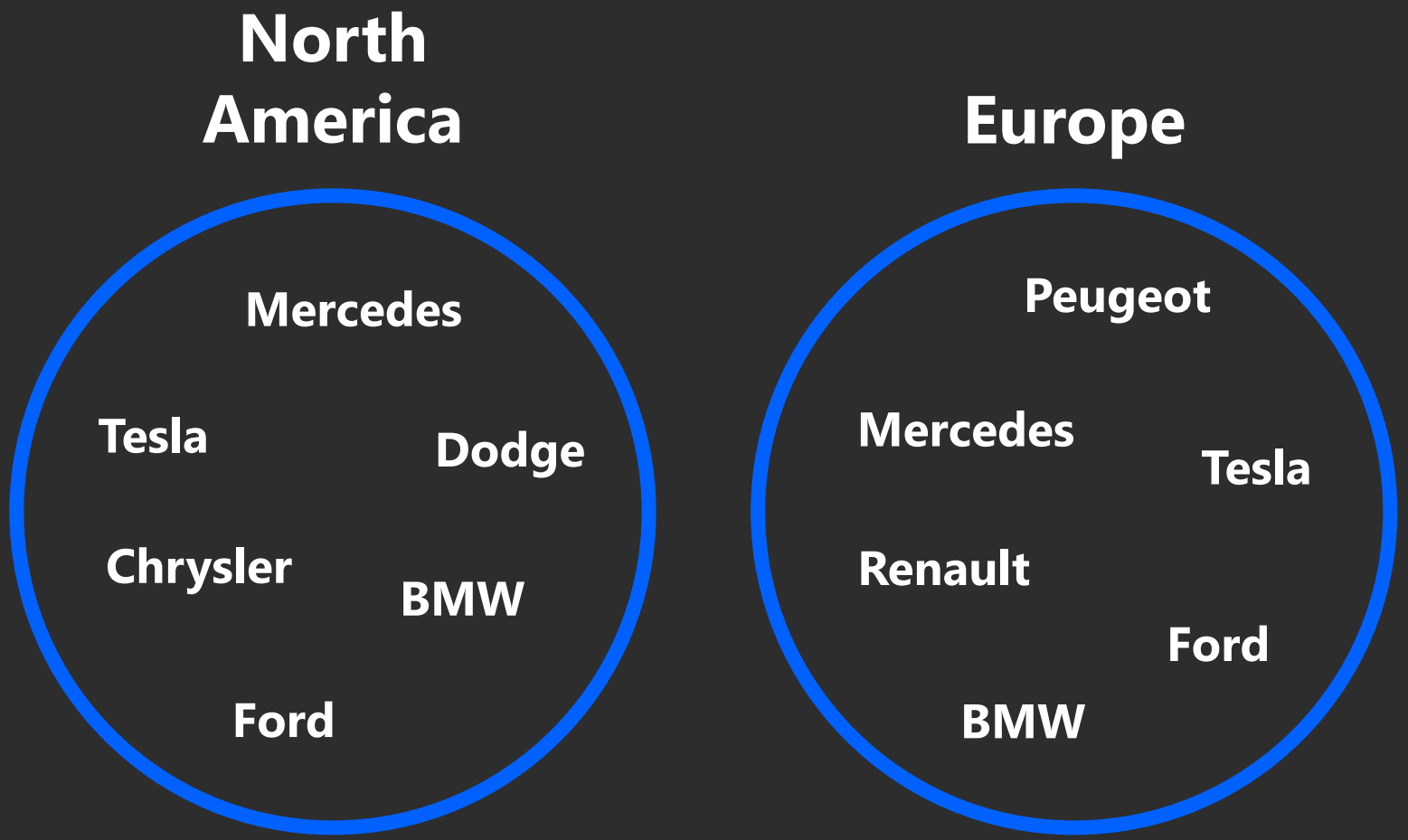
```
['ford', 'tesla', 'dodge', 'tesla']

>>> cars[0:2]
'ford', 'tesla'
```

Set

```
{'ford', 'tesla', 'dodge'}
>>> cars[0:2]
Error
```

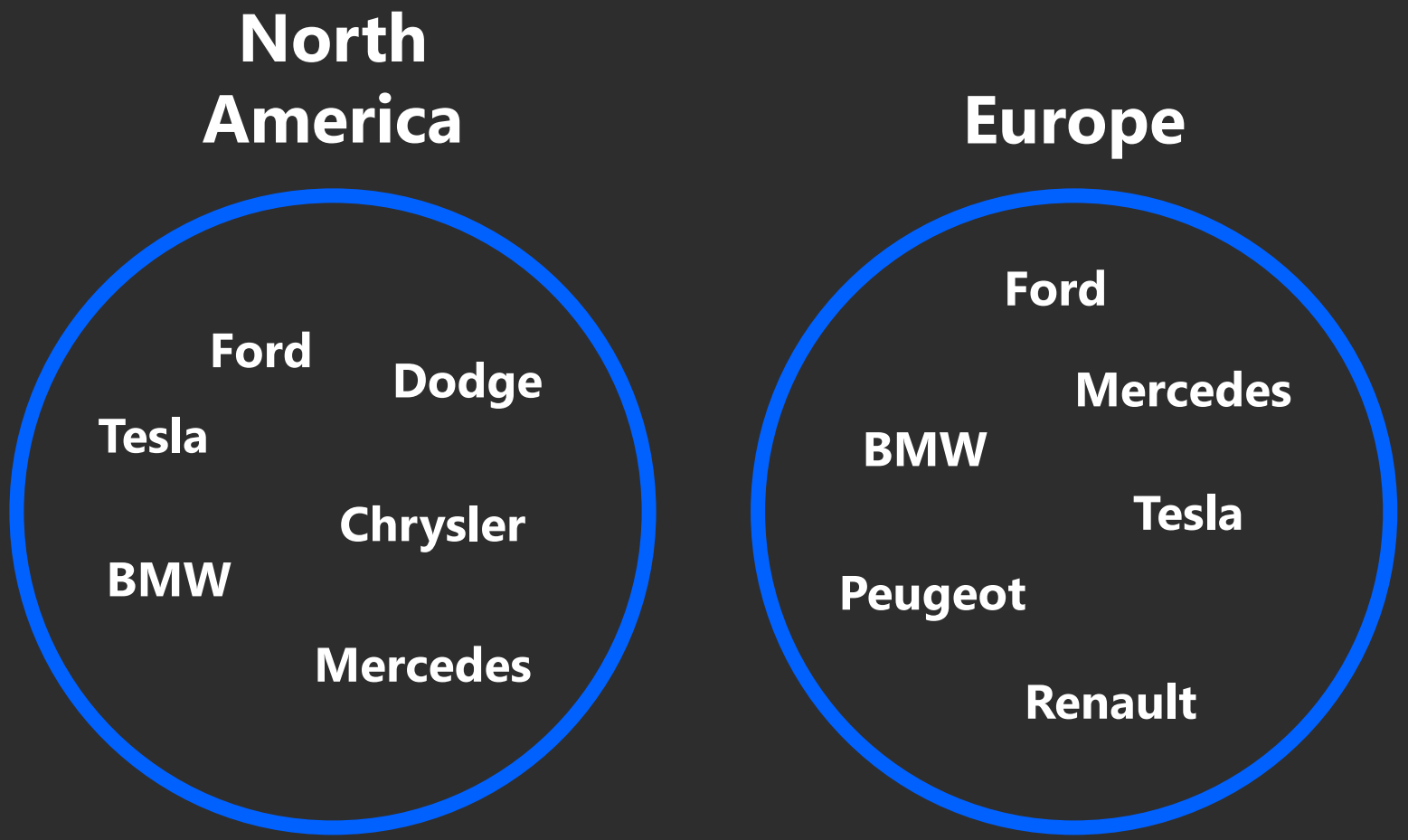
Sets

- Here we have two Sets.
- Cars sold in North America and cars sold in Europe.
- From this graphic, its easy to see that Sets are unordered.



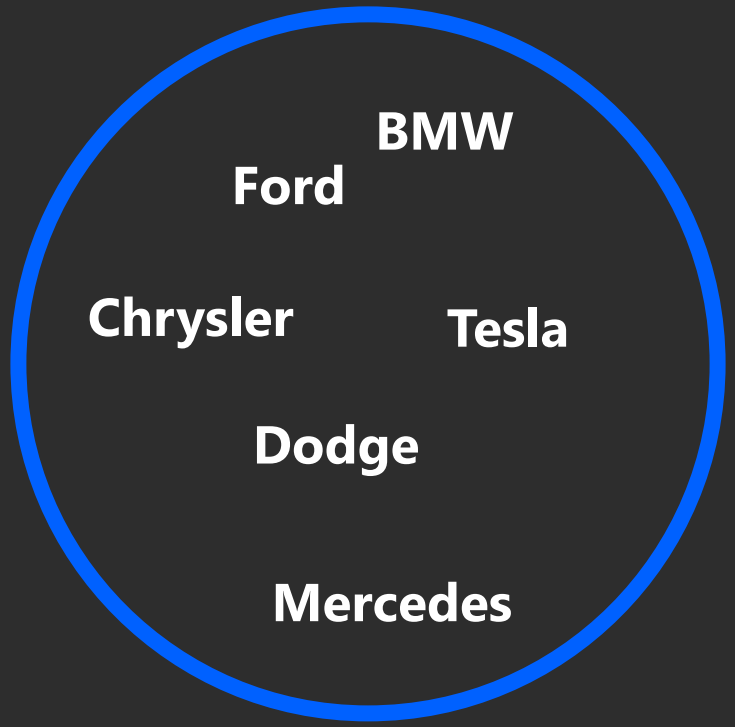
Sets

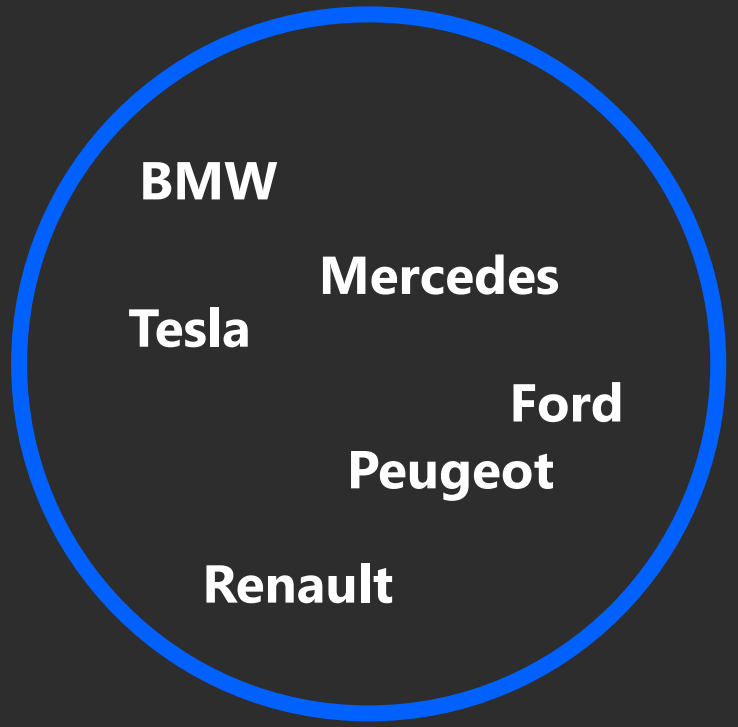
- Here we have two Sets.
- Cars sold in North America and cars sold in Europe.
- From this graphic, its easy to see that Sets are unordered.



Sets

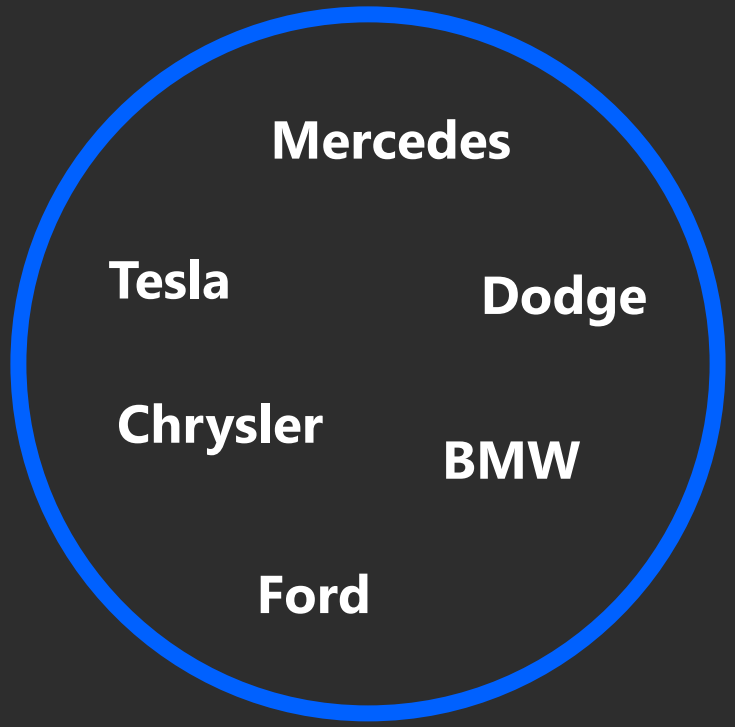
- Here we have two Sets.
- Cars sold in North America and cars sold in Europe.
- From this graphic, its easy to see that Sets are unordered.

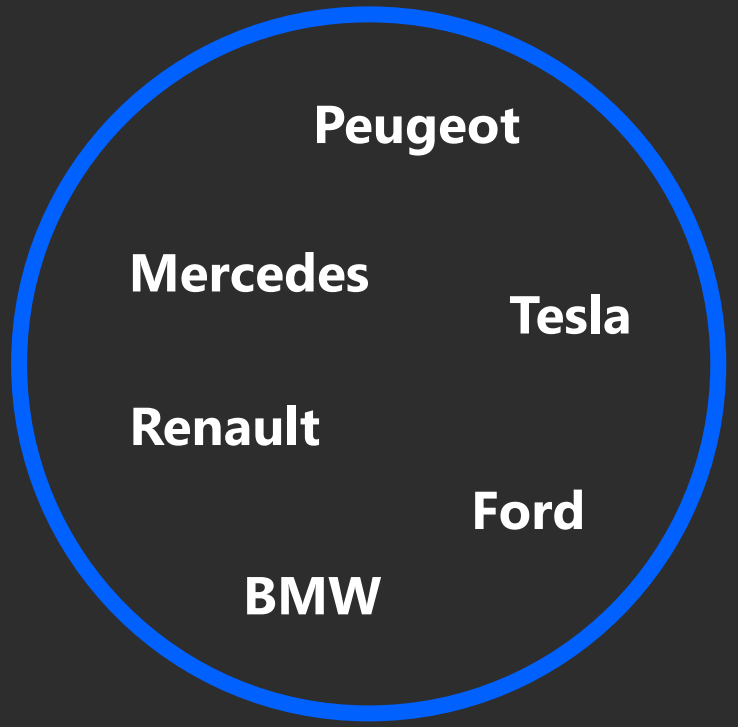




Membership

- Testing for membership is a common operation to perform on a Set.
- Mercedes, Tesla, Dodge, Chrysler, BMW, and Ford are members of the North America Set.
- Similar to lists and tuples, you can test for membership using the `in` operator.

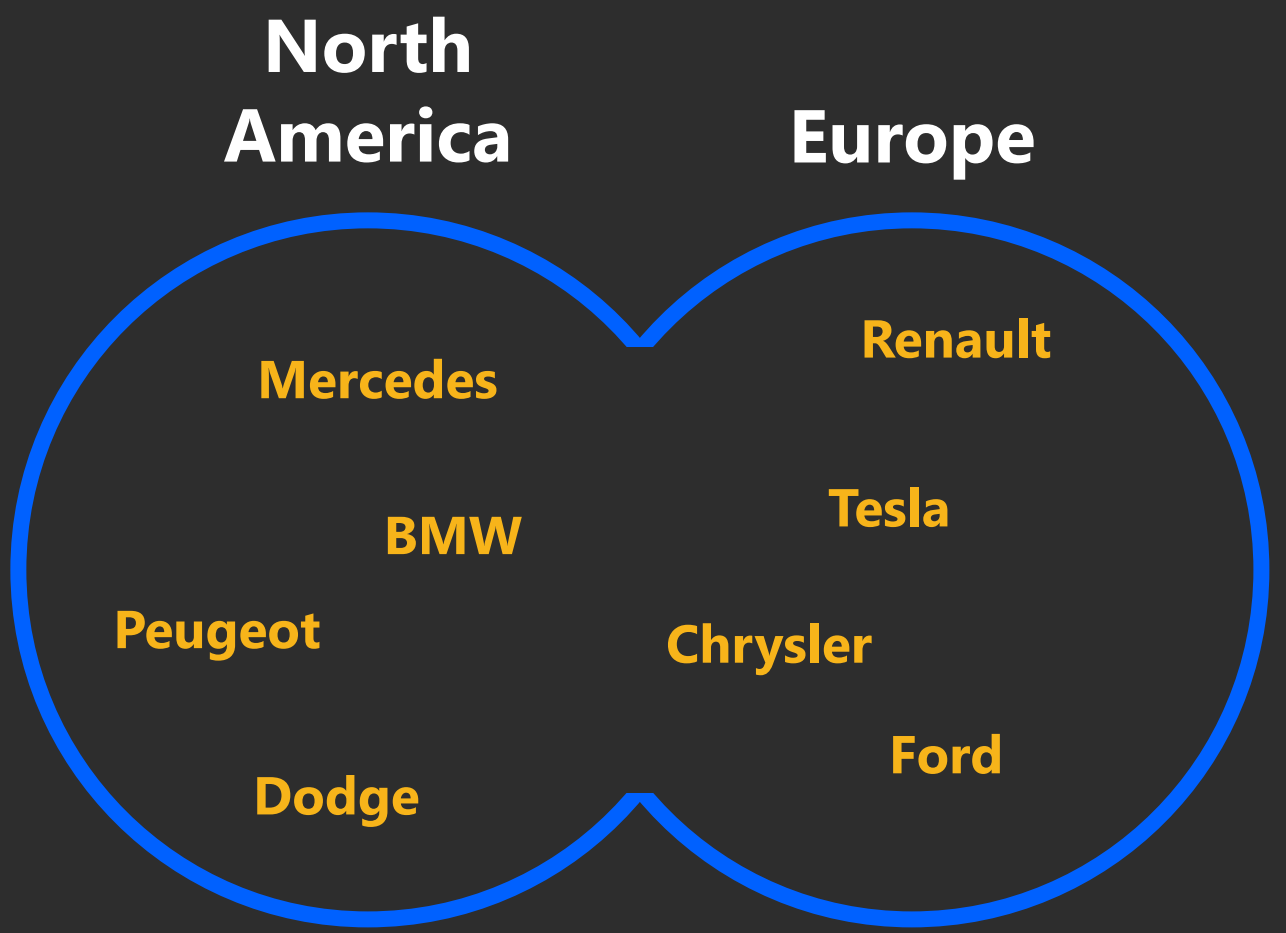





```
>>> 'ford' in north_america
True
```

Union

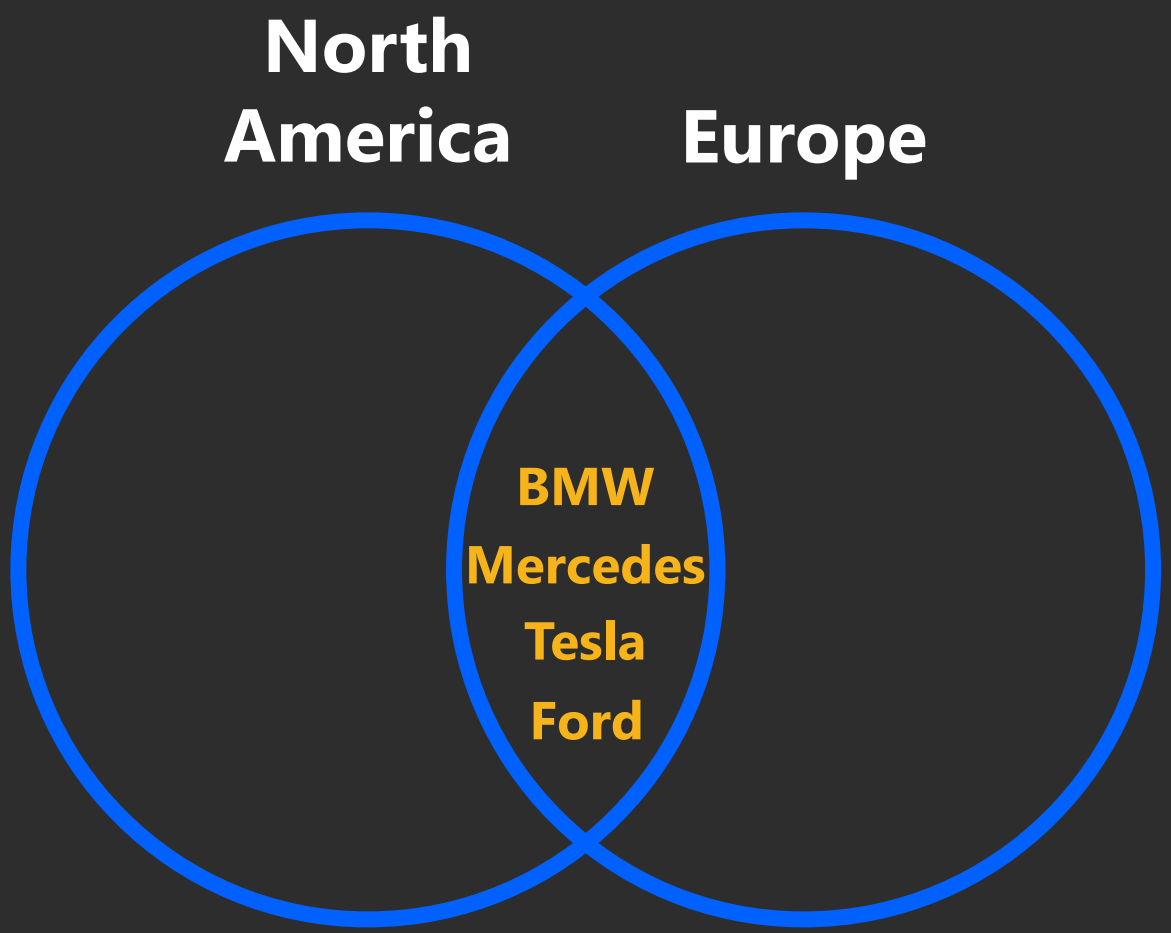
- The Union of two or more Sets is the Set of all items that appear across all Sets.
- Items appear once.
- `north_america.union(europe)`
- `europe.union(north_america)`
- `north_america | europe`
- `europe | north_america`




```
>>> north_america.union(europe)
{'Mercedes', 'BMW', 'Ford', 'Tesla', 'Peugeot', 'Chrysler', 'Renault', 'Dodge'}
```

Intersection

- The Intersection of two or more Sets is the Set of all items that are in each Sets.
- Items appear once.
  - `north_america.intersection(europe)`
  - `europe.intersection(north_america)`
  - `north_america & europe`
  - `europe & north_america`




```
>>> north_america.intersection(europe)
{'Mercedes', 'BMW', 'Ford', 'Tesla'}
```

Sets

- Let's work through some problems with Sets.

Open your notebook

Click Link:
7. Sets

Lecture Recap

- `Tuples` are immutable lists.
- `Tuples`: assignments (packing and unpacking).
- `Sets`: an unordered collection of distinct items.
- `Sets`: set have many methods and operations.
- See **Chapter 11** of the Gries textbook for more on `Tuples` and `Sets`.

APS106

UNIVERSITY OF TORONTO

tuples and sets.

Week 8 | Lecture 1 (8.1)

if nothing else, write **#cleancode**.