Iterator

- Objectives
  - Filter and transform data with list comprehensions
  - Add custom iteration to your classes

List comprehensions

- Python has a concise and local mechanism to
  - transform iterables
  - filter iterables

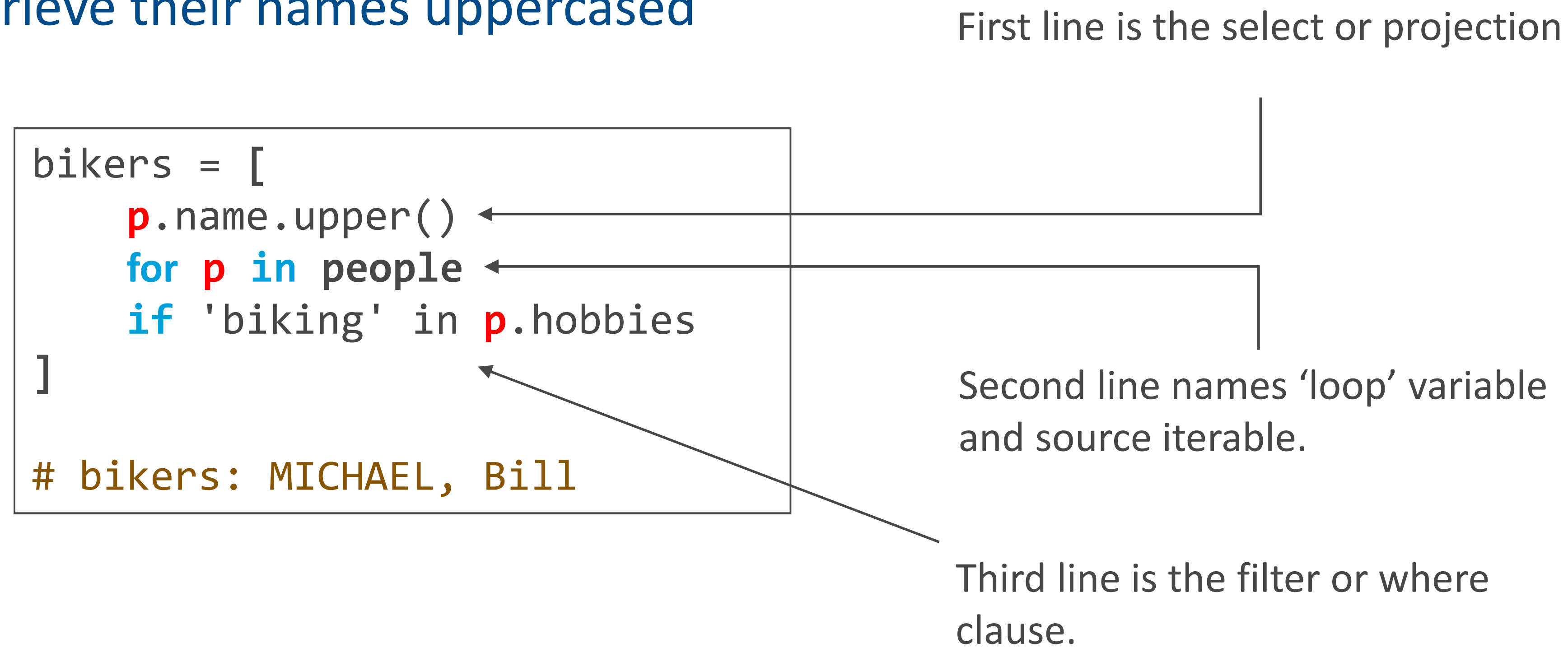
```
# working data for the next few examples

class Person:
    def __init__(self, name, age, hobbies):
        self.name = name
        self.age = age
        self.hobbies = hobbies

people = [
    Person("Jeff", 42, ['tennis', 'hockey', 'football']),
    Person("Michael", 40, ['biking', 'hiking', 'motocross']),
    Person(“Bill", 39, ['biking', 'kite boarding']),
    Person("Stacey", 32, ['skiing']),
]
```

List comprehensions [definition]

- How would we find people who:
  - have biking as a hobby
  - retrieve their names uppercased



```
bikers = [
    p.name.upper()
    for p in people
    if 'biking' in p.hobbies
]

# bikers: MICHAEL, Bill
```

First line is the select or projection

Second line names ‘loop’ variable and source iterable.

Third line is the filter or where clause.

The type of bikers is a **list**.

for in loops [custom classes]

- Some classes (e.g. `dict`) are iterable
- What happens if we try it with our custom classes?

```
class Cart(object):
    # ...

cart = Cart()
cart.add('album', 7.99)
cart.add('book ', 19.99)

for item in cart:
   print( item )

# Boom!
# TypeError: 'Cart' object is not iterable
```

for in loops [implementing iteration]

```
class Cart:
    def __init__(self):
        self.__items = []

    def add(self, cartItem):
        self.__items.append(cartItem)

    def __iter__(self):
        return self.__items.__iter__()
```

Iterable classes define a __iter__ method

We can now iterate over our class.

```
cart = Cart()
cart.add(CartItem('Tesla', 63000))
cart.add(CartItem('BMW', 42000))

total = 0
for item in cart:
    total += item.price

# total is 105,000
```

Summary

- Use the __iter__ magic method to enable iteration
- List comprehensions provide a query language for iterable objects