University of
South Australia

Problem Solving and Programming

Week 13 – Objects

Python Books

Course Textbook

Gaddis, Tony. 2015, *Starting Out with Python*, 3rd edition, Pearson Education, Inc.

Free Electronic Books

There are a number of good free on-line Python books. I recommend that you look at most and see if there is one that you enjoy reading. I find that some books just put me to sleep, while others I enjoy reading. You may enjoy quite a different style of book to me, so just because I say I like a book does not mean it is the one that is best for you to read.

- The following three books start from scratch - they don't assume you have done any prior programming:
  - The free on-line book "**How to think like a Computer Scientist: Learning with Python**", by Peter Wentworth, Jeffrey Elkner, Allen B. Downey and Chris Meyers, provides a good introduction to programming and the Python language. I recommend that you look at this book.
  - There is an on-line book "**A Byte of Python**" that is quite reasonable. See the home page for the book, or you can go directly to the on-line version for Python 3, or download a PDF copy of the book. This book is used in a number of Python courses at different universities and is another I recommend you look at.
  - Another good on-line book is "**Learning to Program**" by Alan Gauld. You can download the whole book in easy to print PDF format, and this is another book that would be good for you to look at.
- If you have done some programming before, you may like to look at the following:
  - **The Python Tutorial** - this is part of Python's documentation and is updated with each release of Python. This is not strictly an e-Book, but is book-sized.
  - **Dive into Python 3**, by Mark Pilgrim is a good book for those with some programming experience. I recommend you have a look at it. You can download a PDF copy.

UniSA

Objects

- An object is a software entity that contains both data and procedures.
- The data contained in an object is known as the object's data attributes.
  - An object's data attributes are variables that reference data.
- The procedures that an object performs are called methods. An object's methods are functions that perform operations on the object's data attributes.
- An object is a self-contained unit that consists of data attributes and methods that operate on the data attributes.

Objects – Under the hood

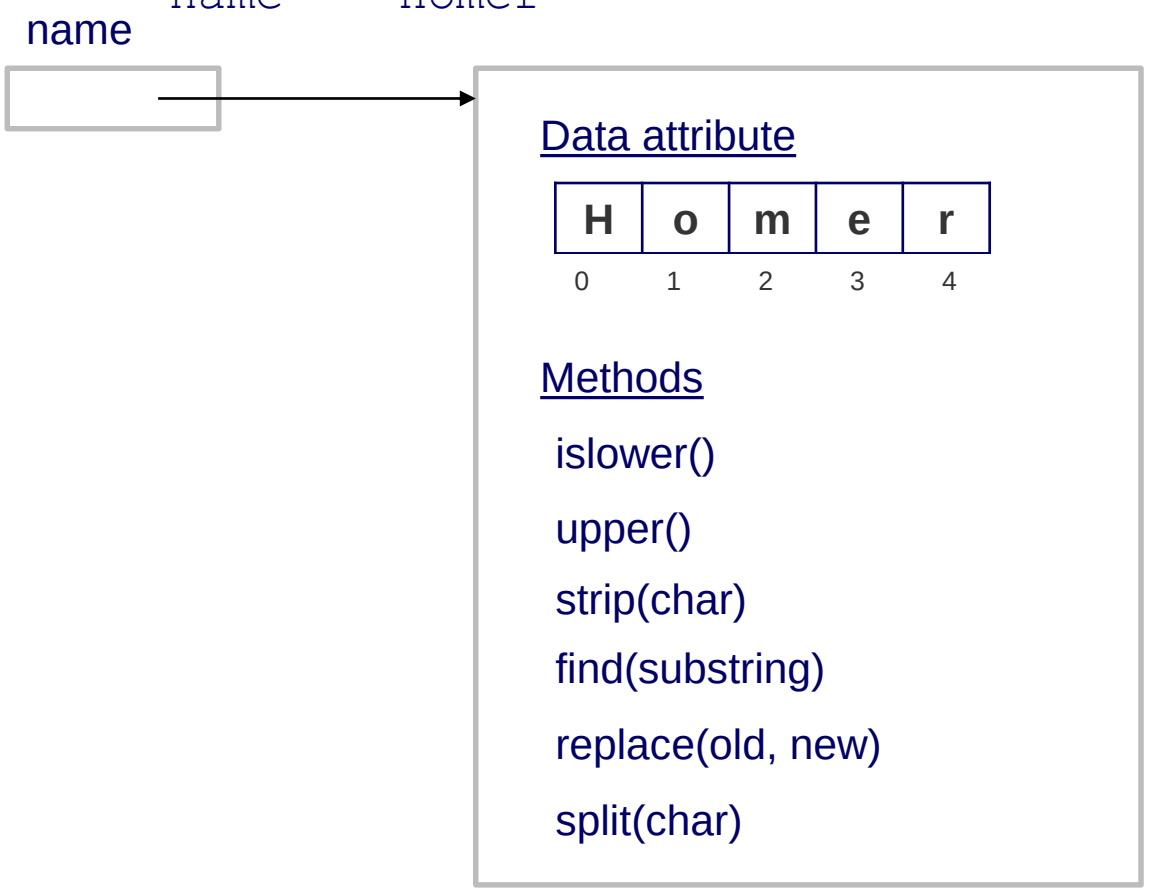
Objects

- We have used List and String objects:
  - A list is an object that contains multiple data items, stored one after the other. Lists are mutable.
  - A String is an object that contains multiple data items (characters), stored one after the other. Strings are immutable.
- List and String objects have data and methods that operate on the data.
  - A method is a function that belongs to an object and performs some operation on that object.
- For example…

Objects

- For example:
  - The following variable definition results in the variable `name` referencing a String object.

```
name = "Homer"
```



** See Python docs for all String methods.*

Objects

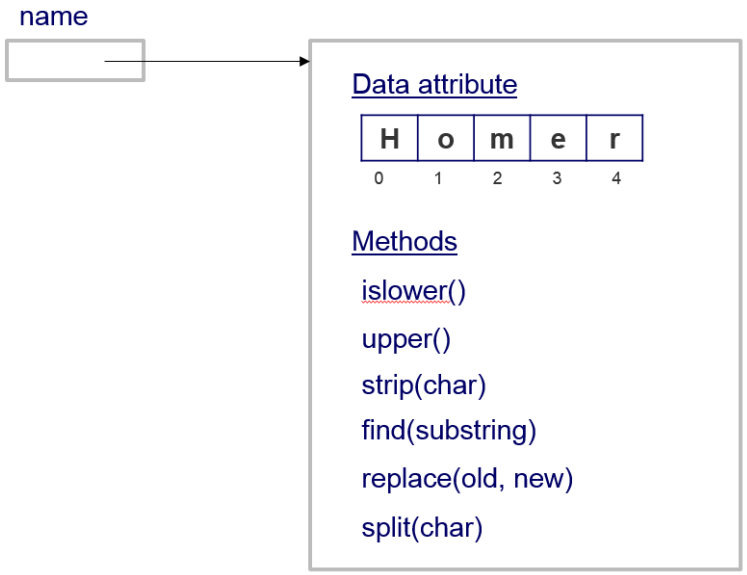
- For example:
  - The following variable definition results in the variable `name` referencing a String object.

    ```
    name = "Homer"
    ```

  - To call a method that belongs to a String object, use dot notation, i.e.:
    If `name` references a String object, then call method islower() like so:

    ```
    name.islower()
    ```

    …call the islower() method that belongs to the `name` (String) object. The method islower() returns `True` if the characters of name are in lowercase, `False` otherwise.



  - Another example:

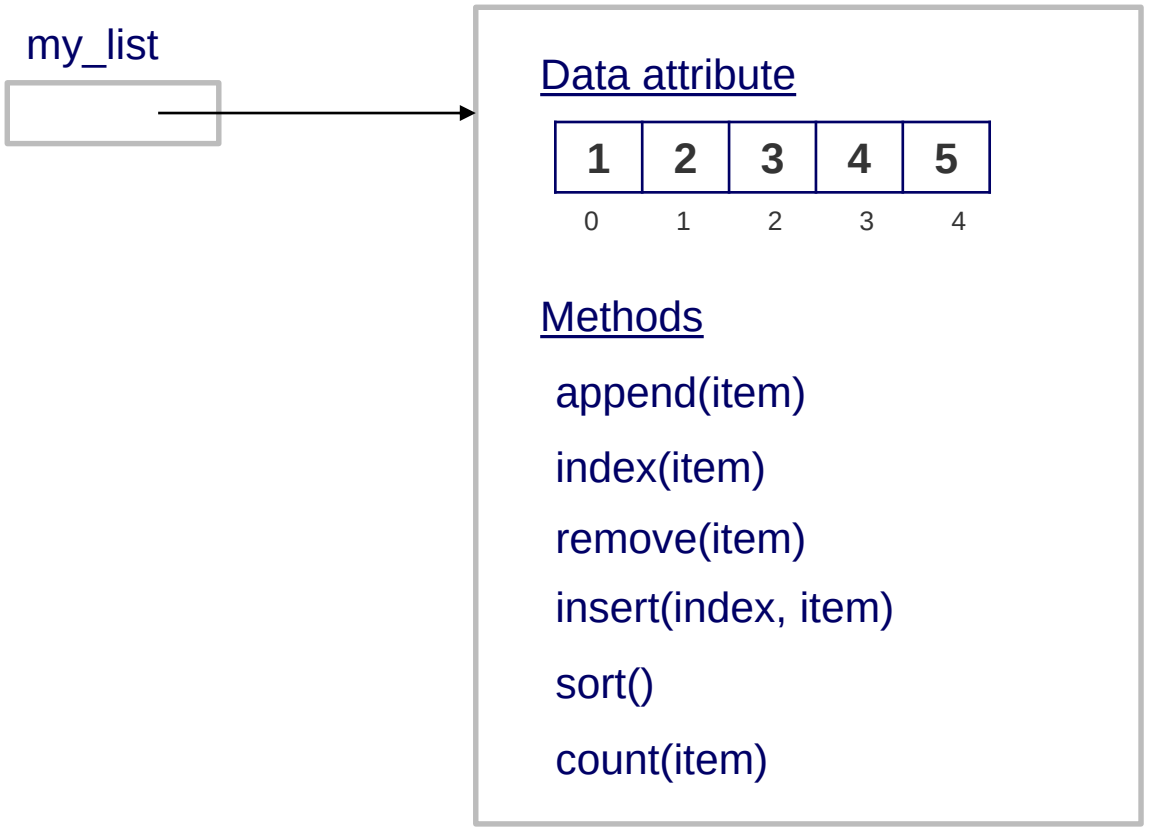
    ```
    name.upper()    =>    "HOMER"
    ```

UniSA

Objects

- For example:
  - The following variable definition results in the variable `my_list` referencing a List object.

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



** See Python docs for all List methods.*

Objects

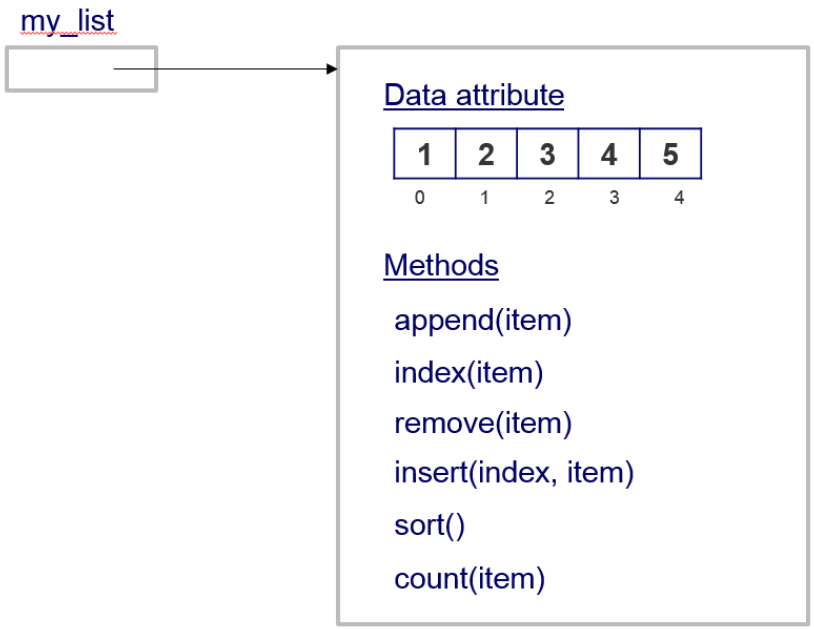
- For example:
  - The following variable definition results in the variable `my_list` referencing a List object.

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

  - To call a method that belongs to a List object, use dot notation, i.e.:
    If `my_name` references a List object, then call method count() like so:

```
my_list.count(2)
```

…call the count() method that belongs to the `my_list` (List) object. The method count() returns the number of times the item appears in the list.



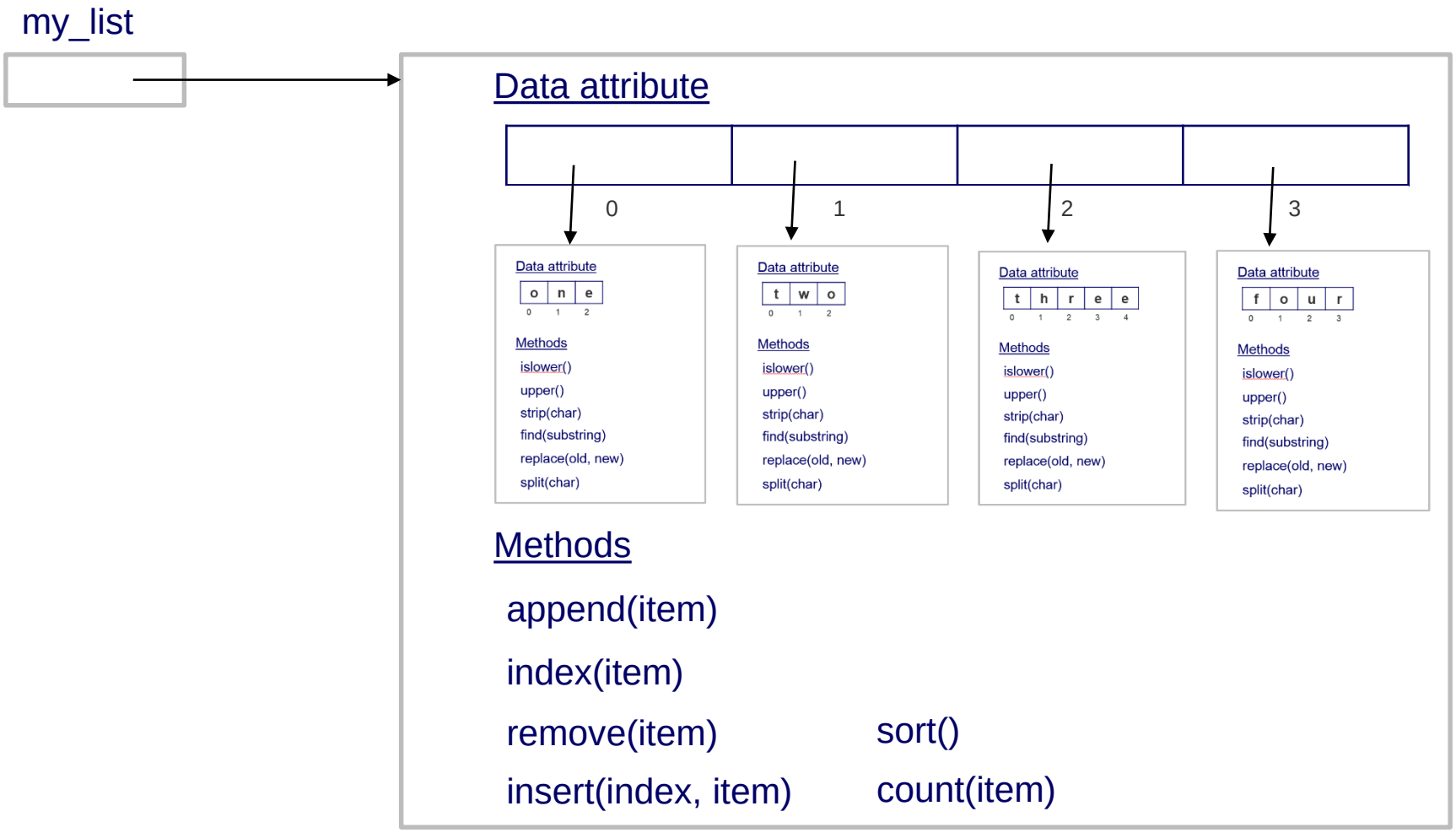
  - Another example:

```
my_list.remove(3)    =>    [1,2,4,5]
```

List of Objects

- Another example:
  - The following variable definition results in the variable `my_list` referencing a List of String objects. i.e.:

```
my_list = ["one", "two", "three", "four"]
```



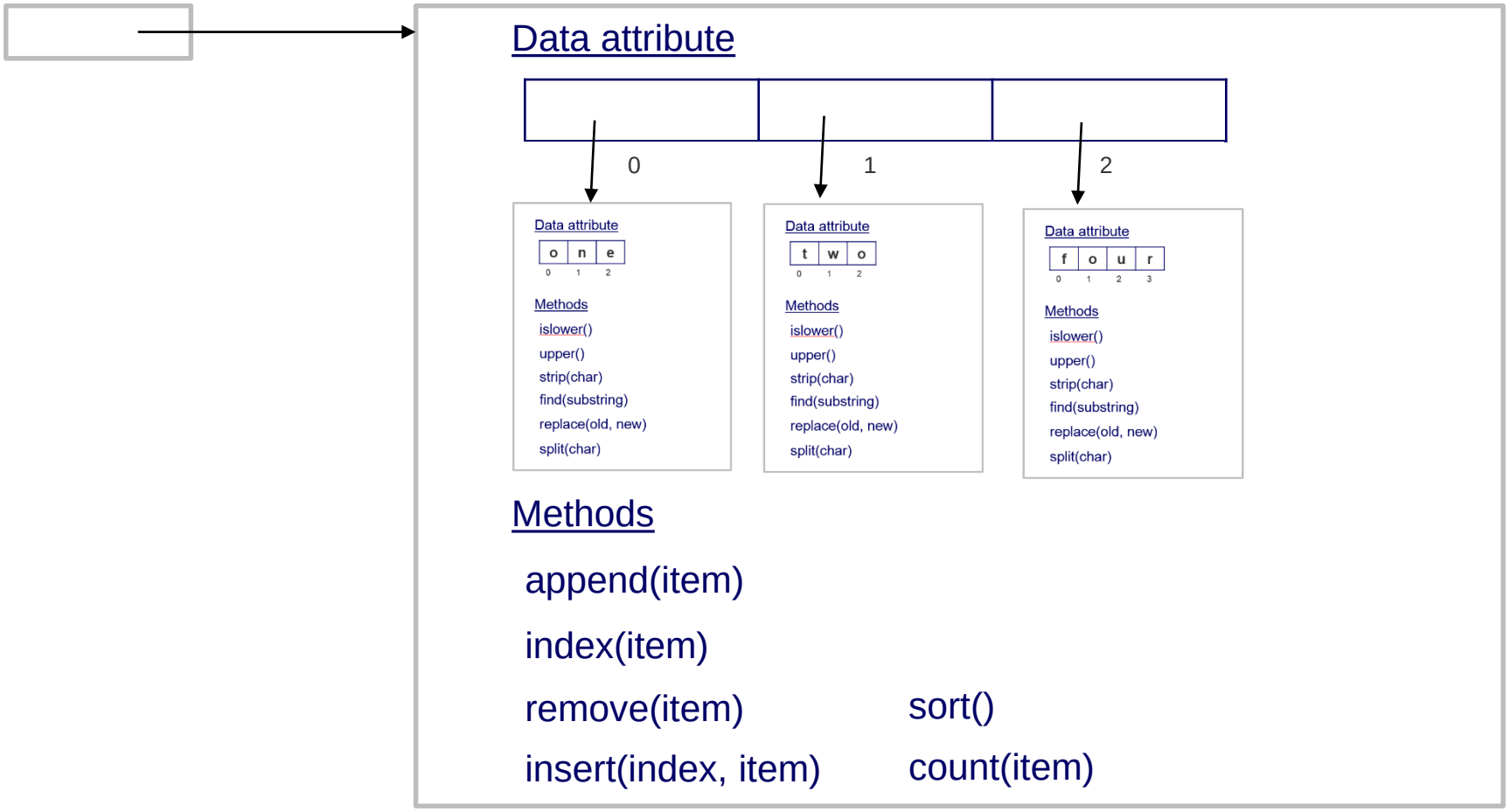
UniSA

List of Objects

- Another example:

```
my_list.remove("three")
```

my_list



UniSA

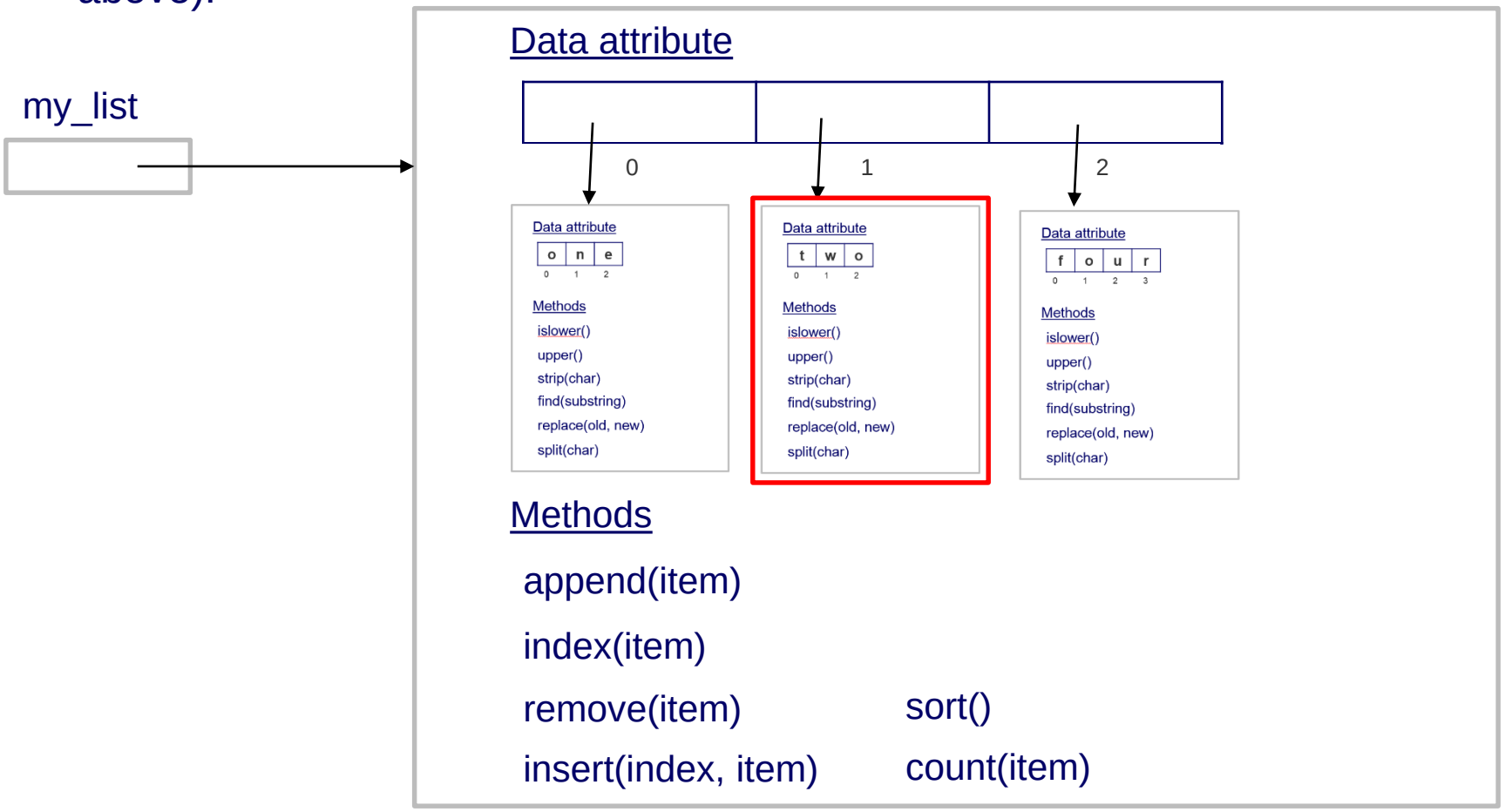
List of Objects

- Accessing an individual string object (i.e. my_list[index]):

```
my_list[1].replace('w', 'o')
```

returns the updated string: `"too"`

Call the replace() method of the second String object in the list (as seen above).



Creating our own Objects
(Optional Reading)

UniSA

Objects

- A class is code that specifies the data attributes and methods for a particular type of object.
- The programmer determines the data attributes and method that are necessary and then creates a class.
- Think of a class as a "blueprint" that objects may be created from.
- The object that is created form a class is called an instance of the class.

Class Definitions

- To create a class, you write a class definition.
- A class definition is a set of statements that define the methods and data attributes of the class.
- Let's write a class named `Coin` that can perform the behaviours of a coin…

Class Definitions

```
class Coin:

    # The __init__ method initialises the sideup data
    # attribute with 'Heads'
    def __init__(self):
        self.sideup = 'Heads'

    # The flip method generates a random number (range 0-1).
    # If the number is 0, then sideup is set to 'Heads',
    # 'Tails' otherwise.
    def flip(self):
        if random.randint(0,1) == 0:
            self.sideup = 'Heads'
        else:
            self.sideup = 'Tails'

    # The get_sideup method returns the value
    # referenced by sideup
    def get_sideup(self):
        return self.sideup
```

Class Definitions - self parameter

```
class Coin:

    # The __init__ method initialises the sideup data
    # attribute with 'Heads'
    def __init__(self):
        self.sideup = 'Heads'

    # The flip method generates a random number (range 0-1).
    # If the number is 0, then sideup is set to 'Heads',
    # 'Tails' otherwise.
    def flip(self):
        if random.randint(0,1) == 0:
            self.sideup = 'Heads'
        else:
            self.sideup = 'Tails'

    # The get_sideup method returns the value
    # referenced by sideup
    def get_sideup(self):
        return self.sideup
```

- The `self` parameter is required in every method of a class.
- A method operates on a specific object's data attributes.
- When a method executes, it must have a way of knowing which object's data attributes it is supposed to operate on.
- When a method is called, Python makes the self parameter reference the specific object that the method is supposed to operate on.

UniSA

Class Definitions - __init__ method

```
class Coin:

    # The __init__ method initialises the sideup data
    # attribute with 'Heads'
    def __init__(self):
        self.sideup = 'Heads'

    # The flip method generates a random number (range 0-1).
    # If the number is 0, then sideup is set to 'Heads',
    # 'Tails' otherwise.
    def flip(self):
        if random.randint(0,1) == 0:
            self.sideup = 'Heads'
        else:
            self.sideup = 'Tails'

    # The get_sideup method returns the value
    # referenced by sideup
    def get_sideup(self):
        return self.sideup
```

- The `__init__` is automatically executed when an instance of the class is created in memory. Special method that initialises the object's data.
- Commonly known as the initialiser method because it initialises the object's data attributes.

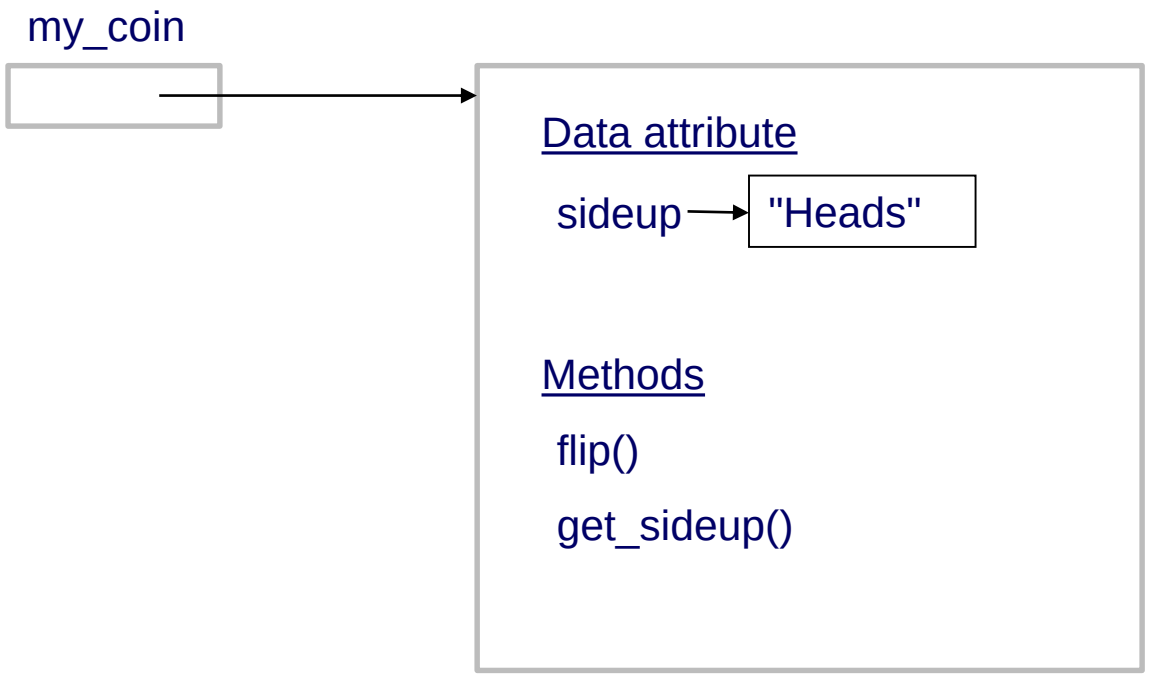
UniSA

Creating an Object (instance of a class)

- To create an instance of the Coin class:

```
my_coin = Coin()
```

- An object is created in memory from the `Coin` class.
- The `Coin` class' `__init__` method is executed, and the `self` parameter is automatically set to the object that was just created. The object's `sideup` data attribute is assigned the string 'Heads'.
- The `my_coin` variable is assigned (references) the newly created `Coin` object.



Creating an Object (instance of a class)

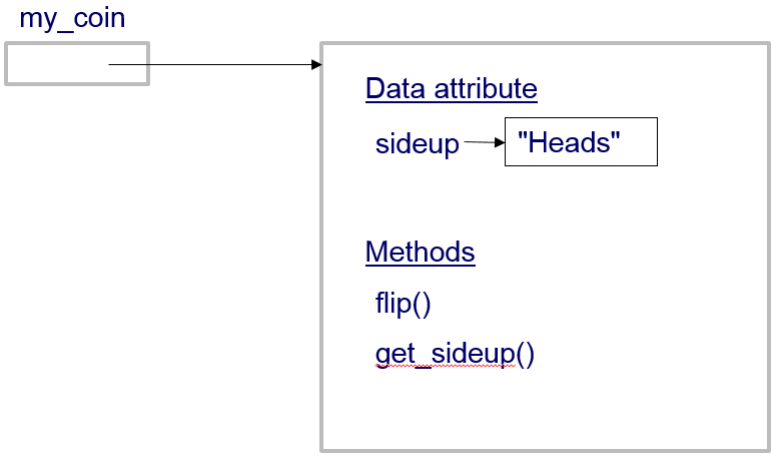
- For example:
  - The following variable definition results in the variable `my_coin` referencing a Coin object.

```
my_coin = Coin()
```

  - To call a method that belongs to a Coin object, use dot notation, i.e.:
    If `my_coin` references a Coin object, then call method flip() like so:

```
my_coin.flip()
```

…call the flip() method that belongs to the `my_coin` (Coin) object. The method flip() simulates the tossing of a coin.



  - Another example:

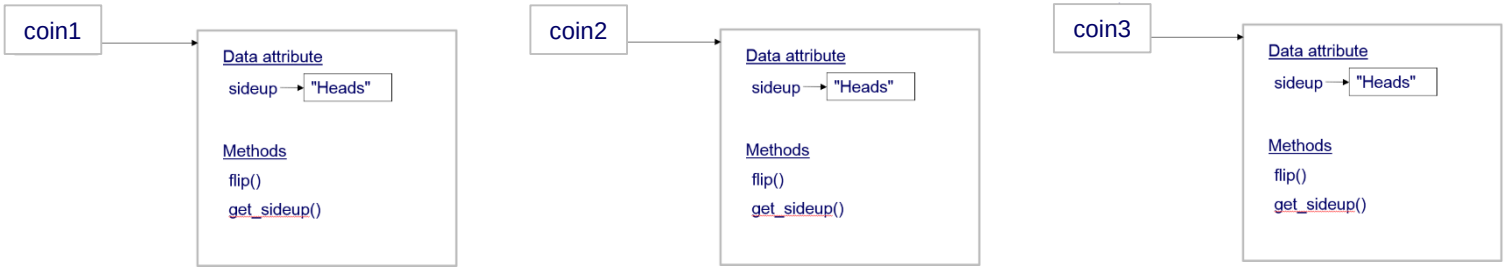
```
my_coin.get_sideup()    =>    "Heads"
```

Creating an Object (instance of a class)

- Creating **many** instances of the same class. For example:
  - The following code creates three objects, each an instance of the `Coin` class.

```
coin1 = Coin()
coin2 = Coin()
coin3 = Coin()
```

  - `coin1`, `coin2` and `coin3` variables reference the three objects. Each object has its own `__sideup` data attribute.



  - Call each object's flip() method:

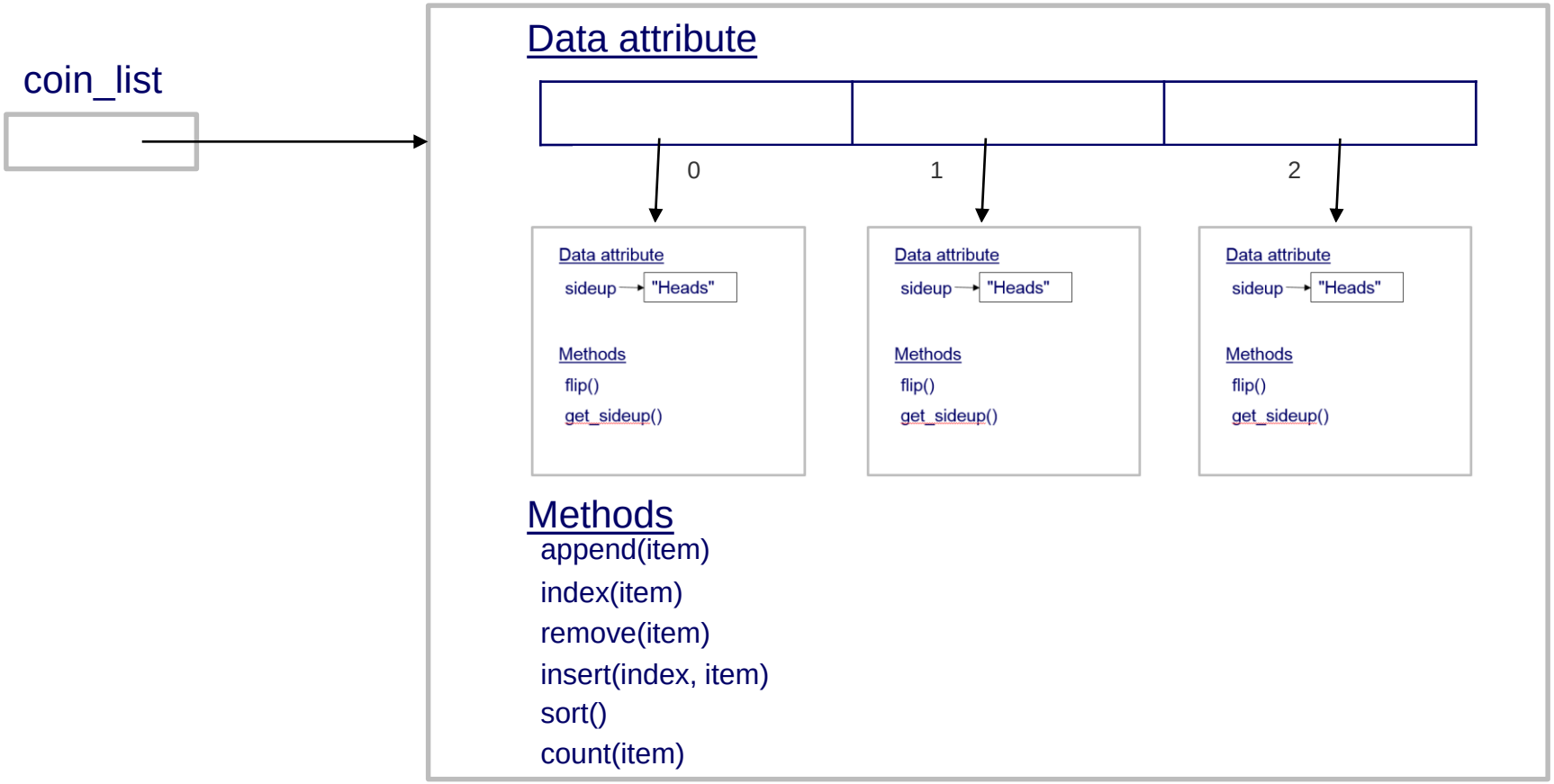
```
coin1.flip()
coin2.flip()
coin3.flip()
```

UniSA

List of Coin Objects

- The following statements result in the variable `coin_list` referencing a List of Coin objects. i.e.:

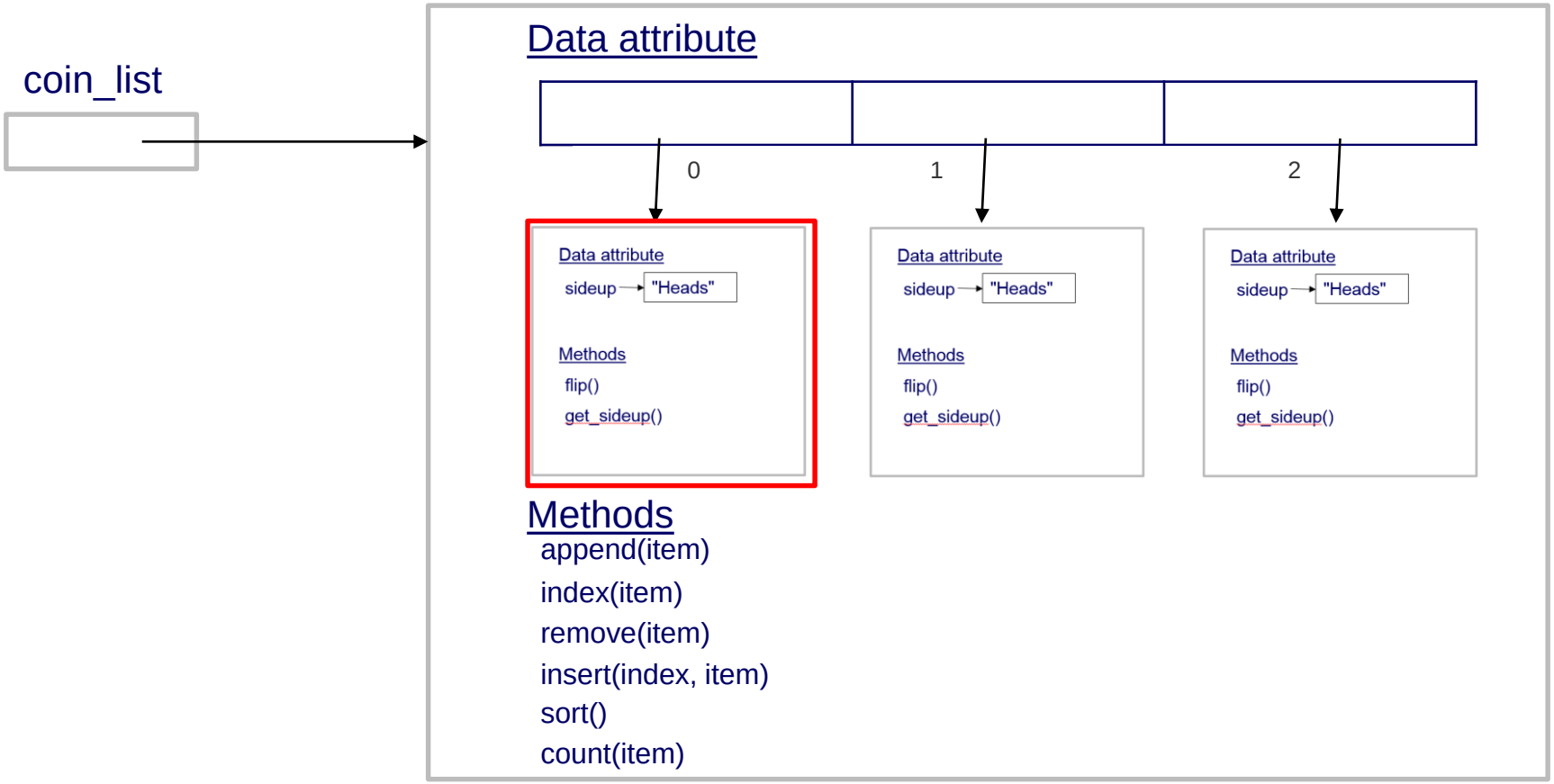
```
coin1 = Coin()
coin2 = Coin()
coin3 = Coin()
coin_list = [coin1, coin2, coin3]
```



Looping over a List of Coin Objects

- Looping over (iterating over) a List of Coin objects. i.e.:

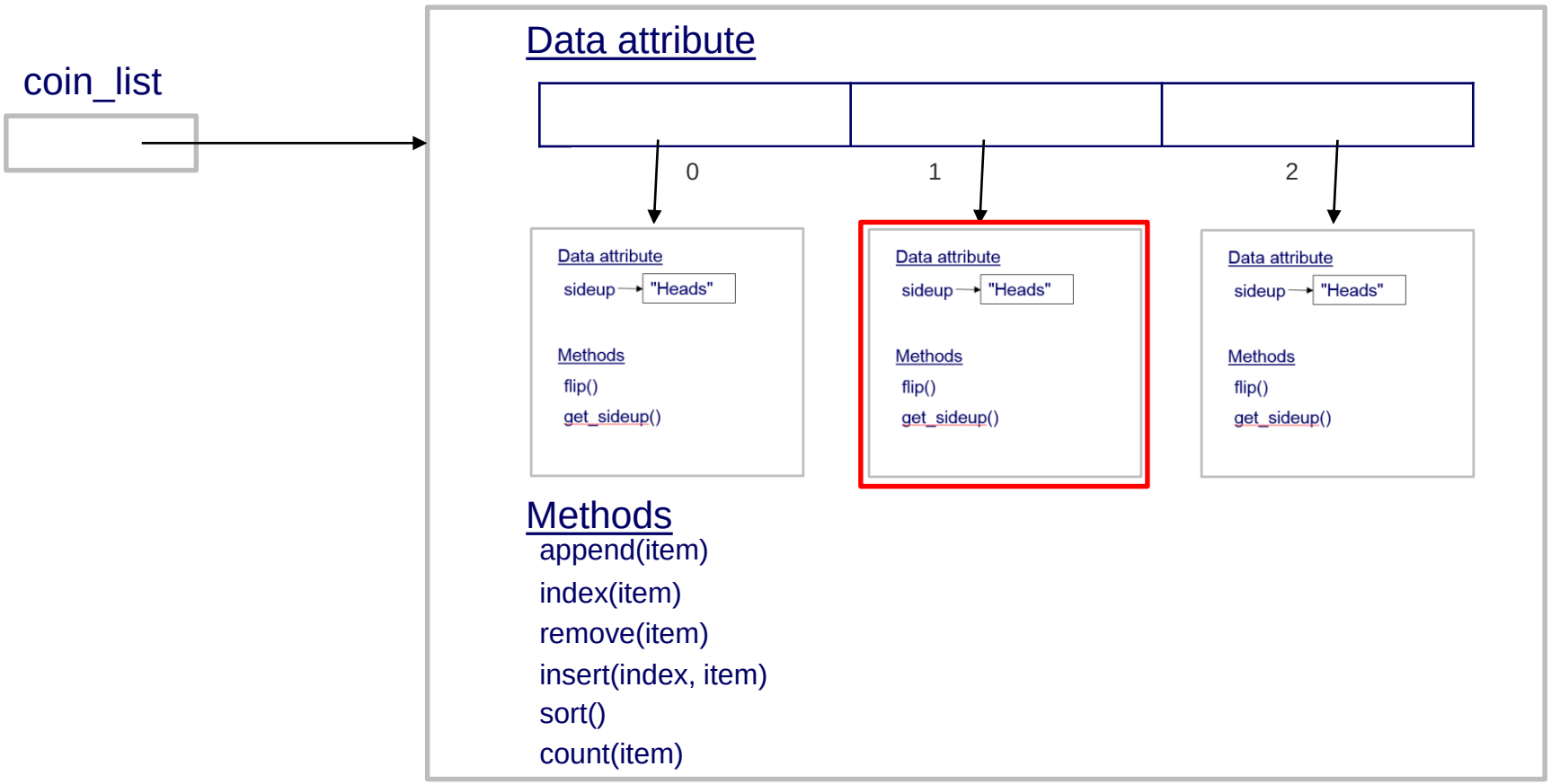
```
coin_list = [coin1, coin2, coin3]
for coin in coin_list:
    print(coin.get_sideup())
```



Looping over a List of Coin Objects

- Looping over (iterating over) a List of Coin objects. i.e.:

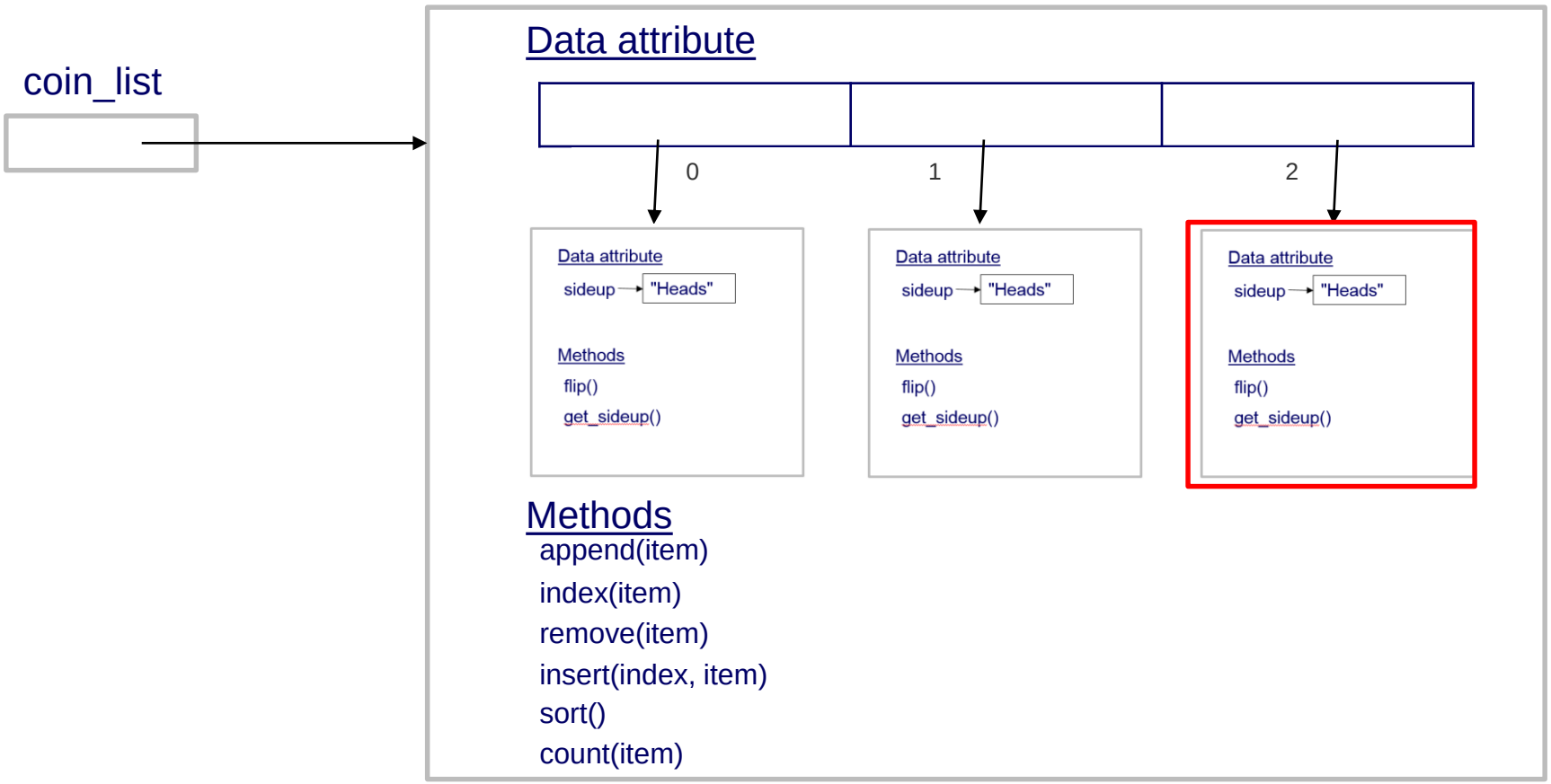
```
coin_list = [coin1, coin2, coin3]
for coin in coin_list:
    print(coin.get_sideup())
```



Looping over a List of Coin Objects

- Looping over (iterating over) a List of Coin objects. i.e.:

```
coin_list = [coin1, coin2, coin3]
for coin in coin_list:
    print(coin.get_sideup())
```

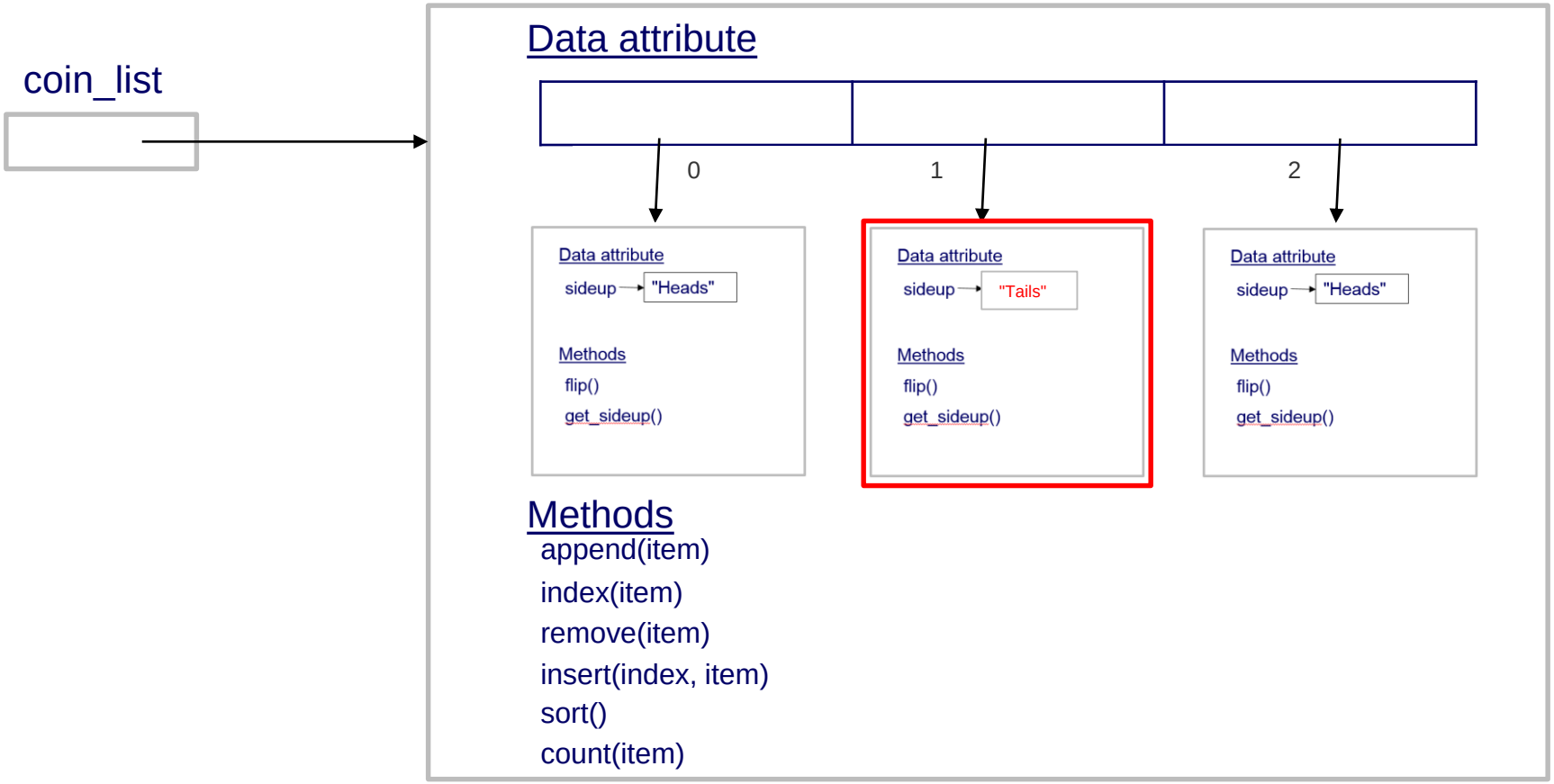


Looping over a List of Coin Objects

- Accessing an individual Coin object (i.e. coin_list[index]).

```
coin_list[1].flip()
```

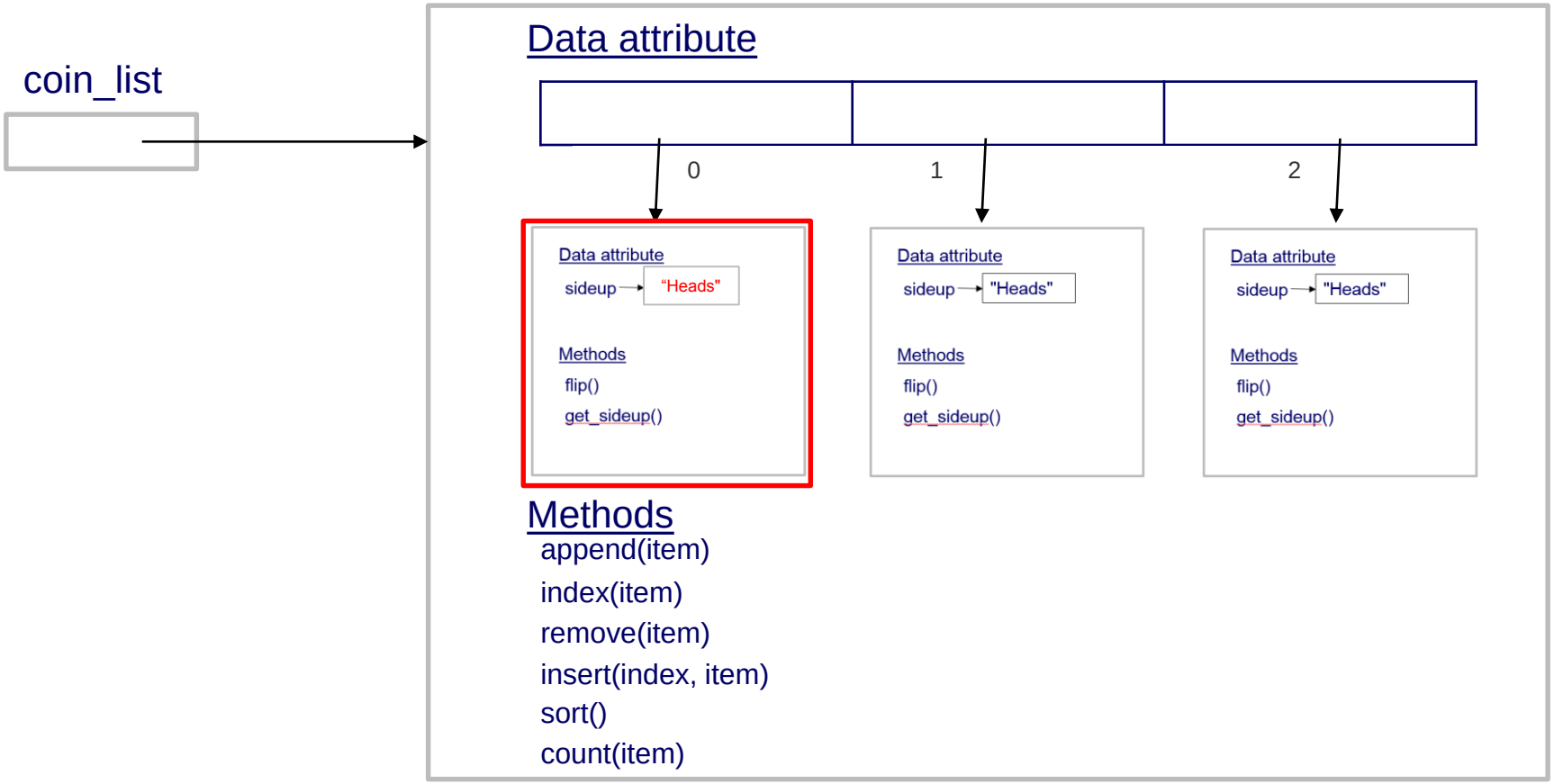
Call the flip() method of the second Coin object in the list (as seen above). Updates the Coin object (data attribute `sideup`) referenced by the second item in the `coin_list`. i.e. the coin object referenced by position/index 1.



Looping over a List of Coin Objects

- Looping over (iterating over) a List of Coin objects in order to "flip" (call the flip() method belonging to) a Coin object:

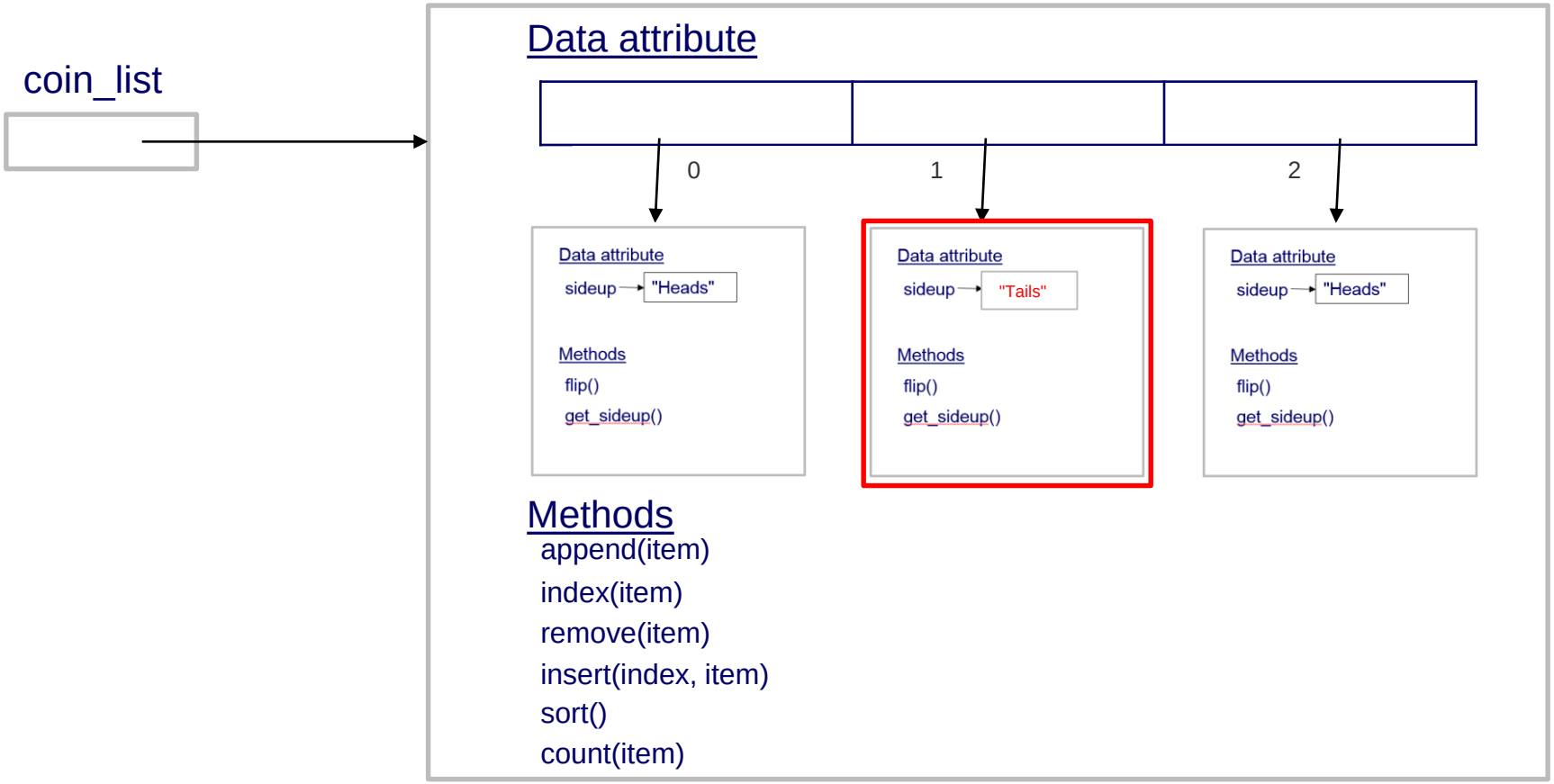
```
coin_list = [coin1, coin2, coin3]
for coin in coin_list:
    coin.flip()
```



Looping over a List of Coin Objects

- Looping over (iterating over) a List of Coin objects in order to "flip" (call the flip() method belonging to) a Coin object:

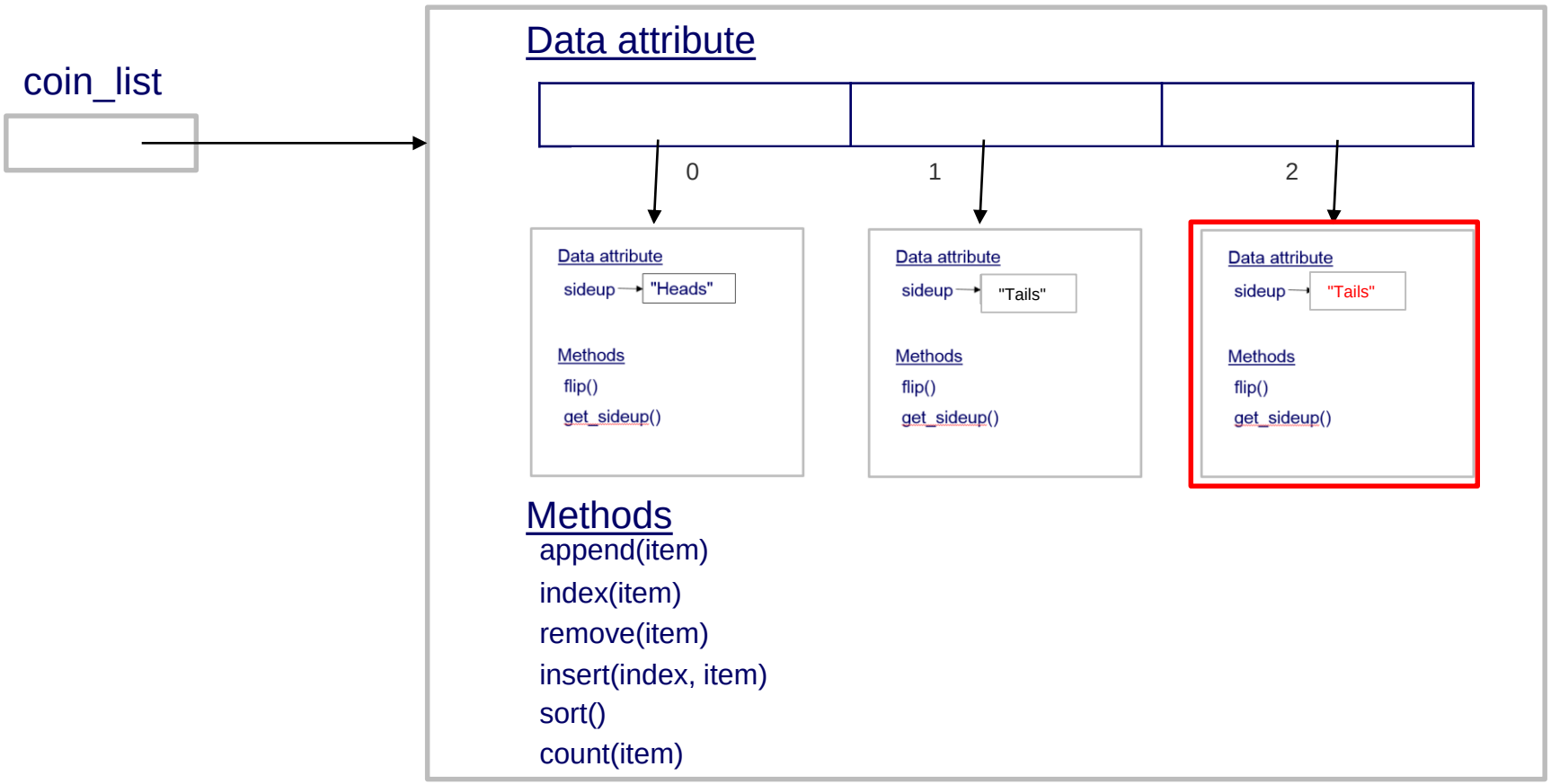
```
coin_list = [coin1, coin2, coin3]
for coin in coin_list:
    coin.flip()
```



Looping over a List of Coin Objects

- Looping over (iterating over) a List of Coin objects in order to "flip" (call the flip() method belonging to) a Coin object:

```
coin_list = [coin1, coin2, coin3]
for coin in coin_list:
    coin.flip()
```



End of Week 13 Objects