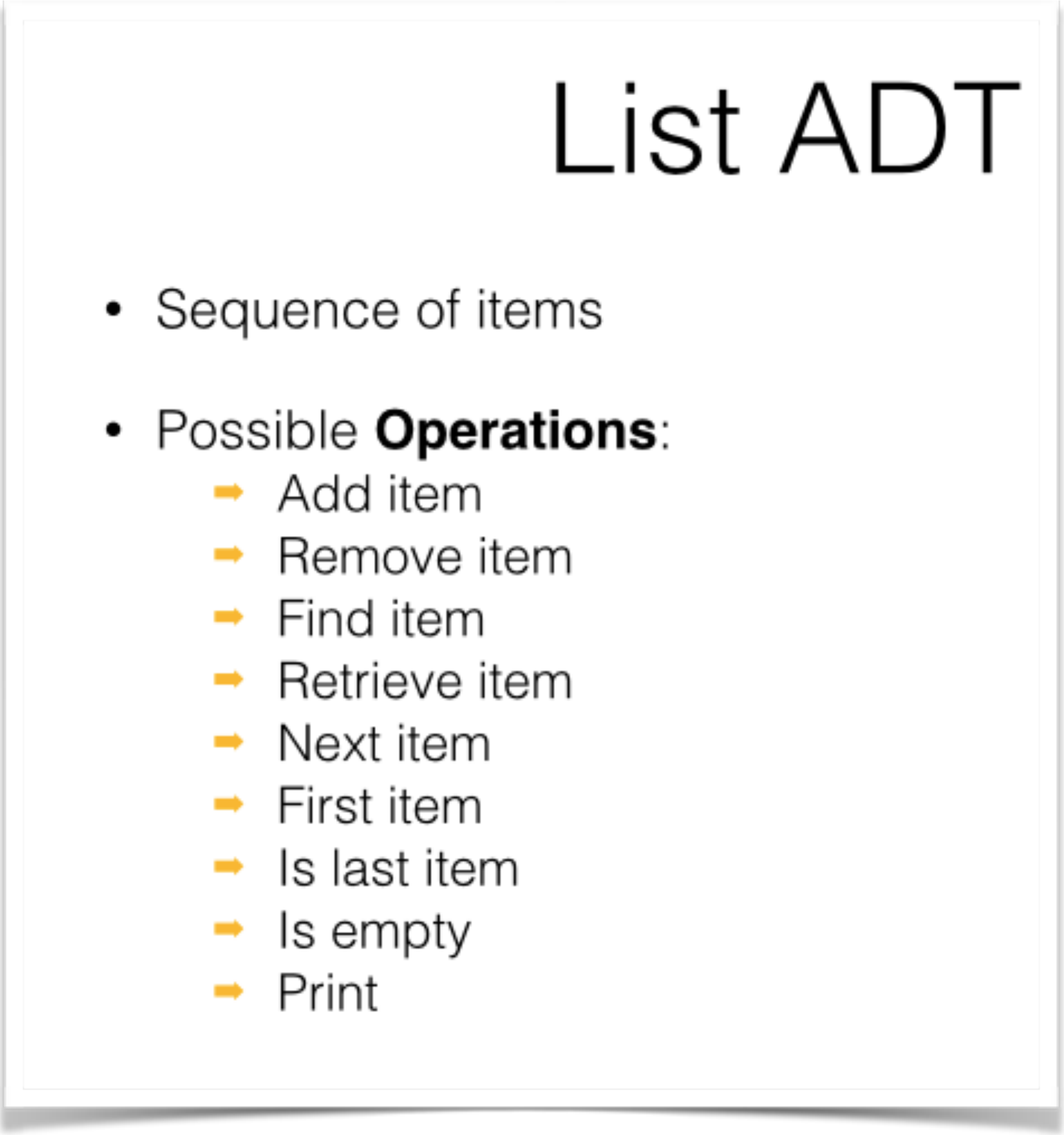
Prepared by Julian García, based on material by
David Albrecht and María García de la Banda

Lecture 16
Lists and Sorted List
(Array Implementation)

FIT 1008
Introduction to Computer Science

MONASH University
Information Technology

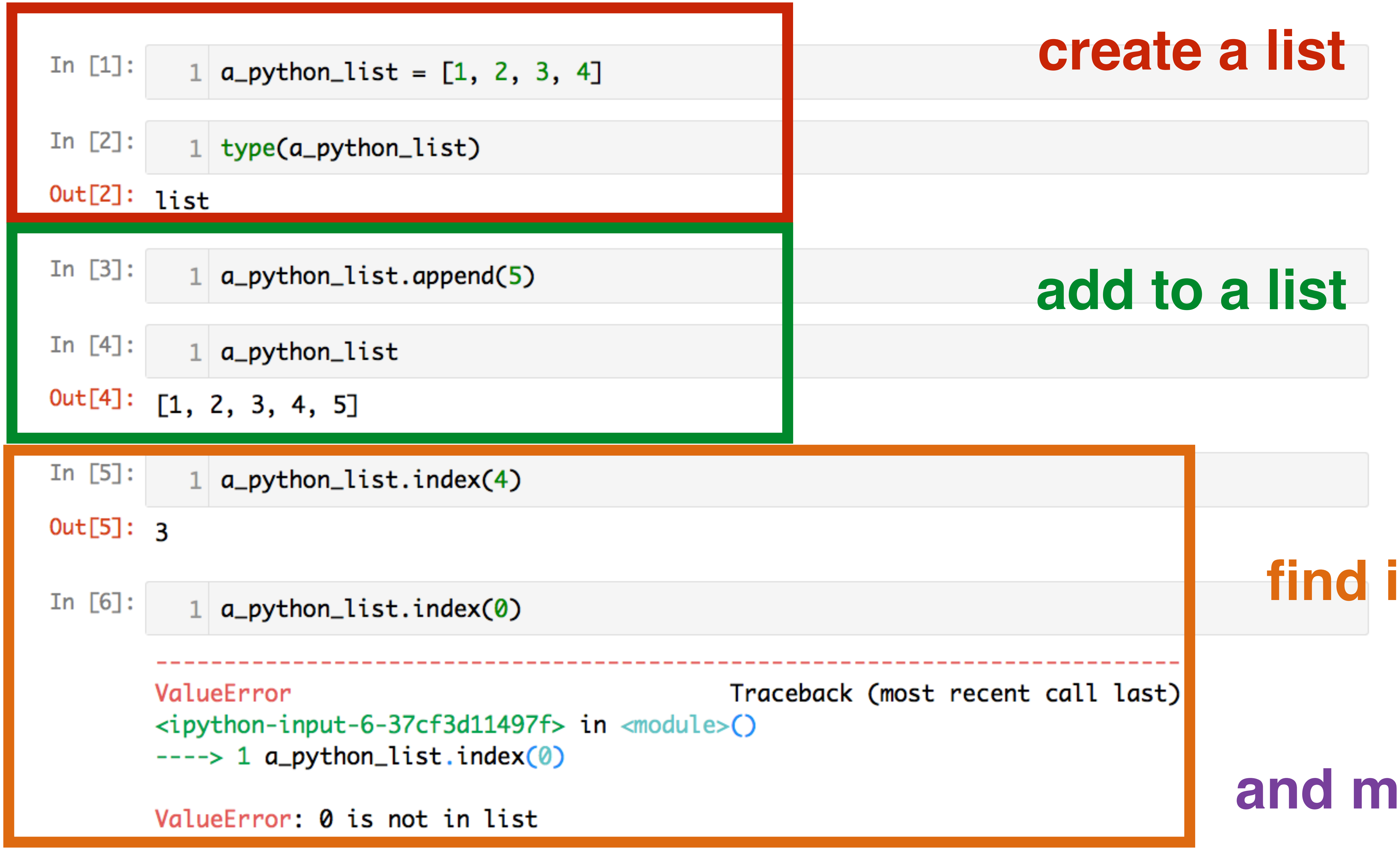
Put everything together…
and implement some
Abstract Data Types



List ADT

- Sequence of items
- Possible **Operations**:
  - ➡ Add item
  - ➡ Remove item
  - ➡ Find item
  - ➡ Retrieve item
  - ➡ Next item
  - ➡ First item
  - ➡ Is last item
  - ➡ Is empty
  - ➡ Print

We already use one implementation of a List ADT



```
In [1]: a_python_list = [1, 2, 3, 4]

In [2]: type(a_python_list)
Out[2]: list

In [3]: a_python_list.append(5)

In [4]: a_python_list
Out[4]: [1, 2, 3, 4, 5]

In [5]: a_python_list.index(4)
Out[5]: 3

In [6]: a_python_list.index(0)

        ---------------------------------------------------------------------------
        ValueError                                Traceback (most recent call last)
        <ipython-input-6-37cf3d11497f> in <module>()
        ----> 1 a_python_list.index(0)

        ValueError: 0 is not in list
```

create a list

add to a list

find item

and more…

Implementing your own List ADT

- How do we start? Easy:
  - Create a **new file** (called *my_list.py*)
  - Import the build_array function so that you can create arrays.
  - Add any **operations/methods** users my need to use.

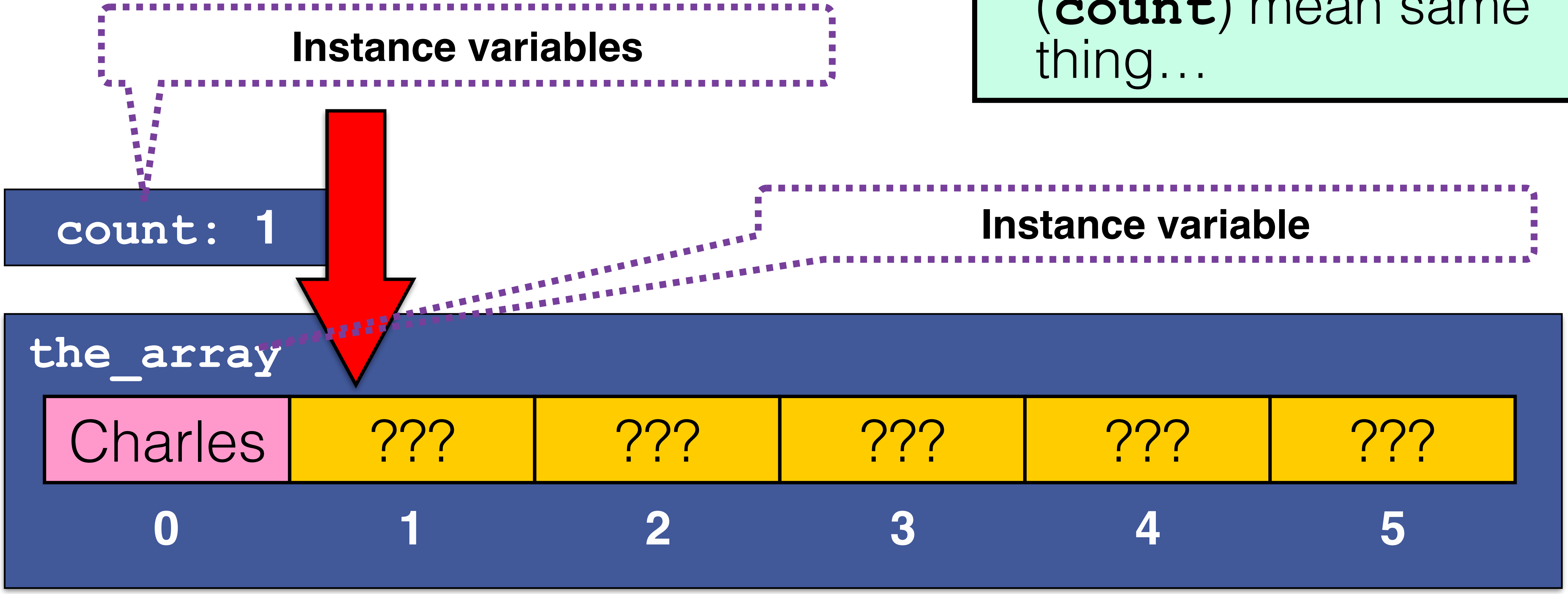
- What operations?
  - **Create a list**, **access** an element, compute the **length**
  - Determine whether **is empty**
  - Determine whether it **has a given item**
  - Find the **position of an item** (if in)
  - **Add/delete** an item
  - **Delete/insert** the item in position *i*

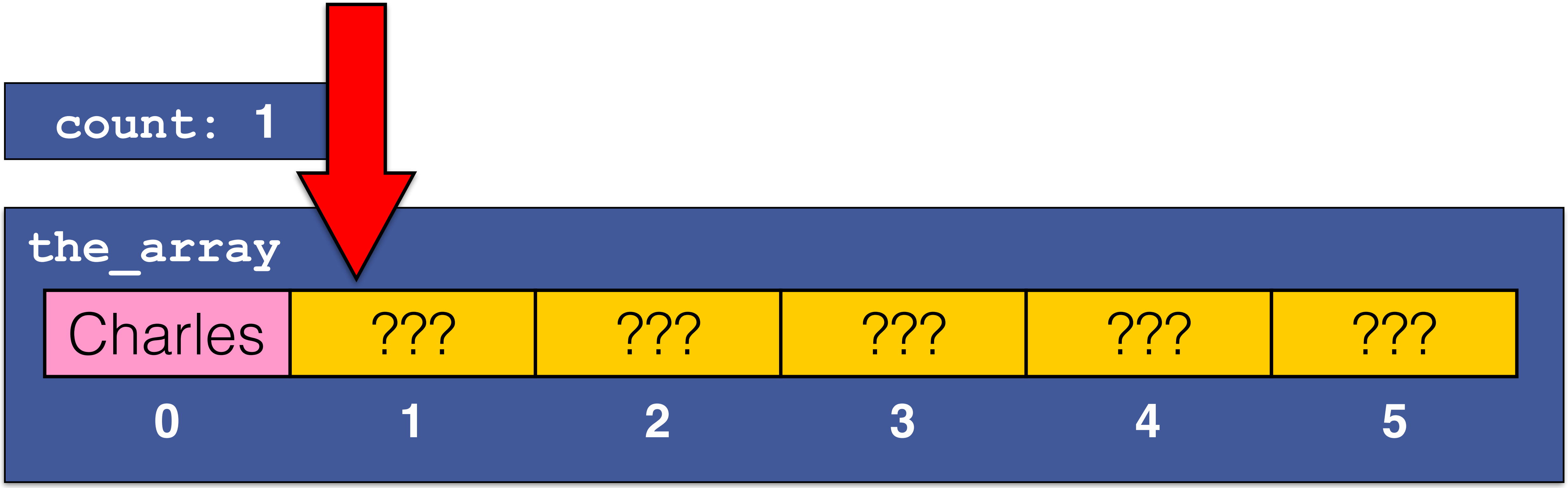
Visualising lists implemented with arrays

- Consider a list defined:
  - Over an array of size 6
  - Currently with one element (Charles)

- We will visualise it like this:

Visual Clarity:
Arrow, **Colour**, Counter (`count`) mean same thing…

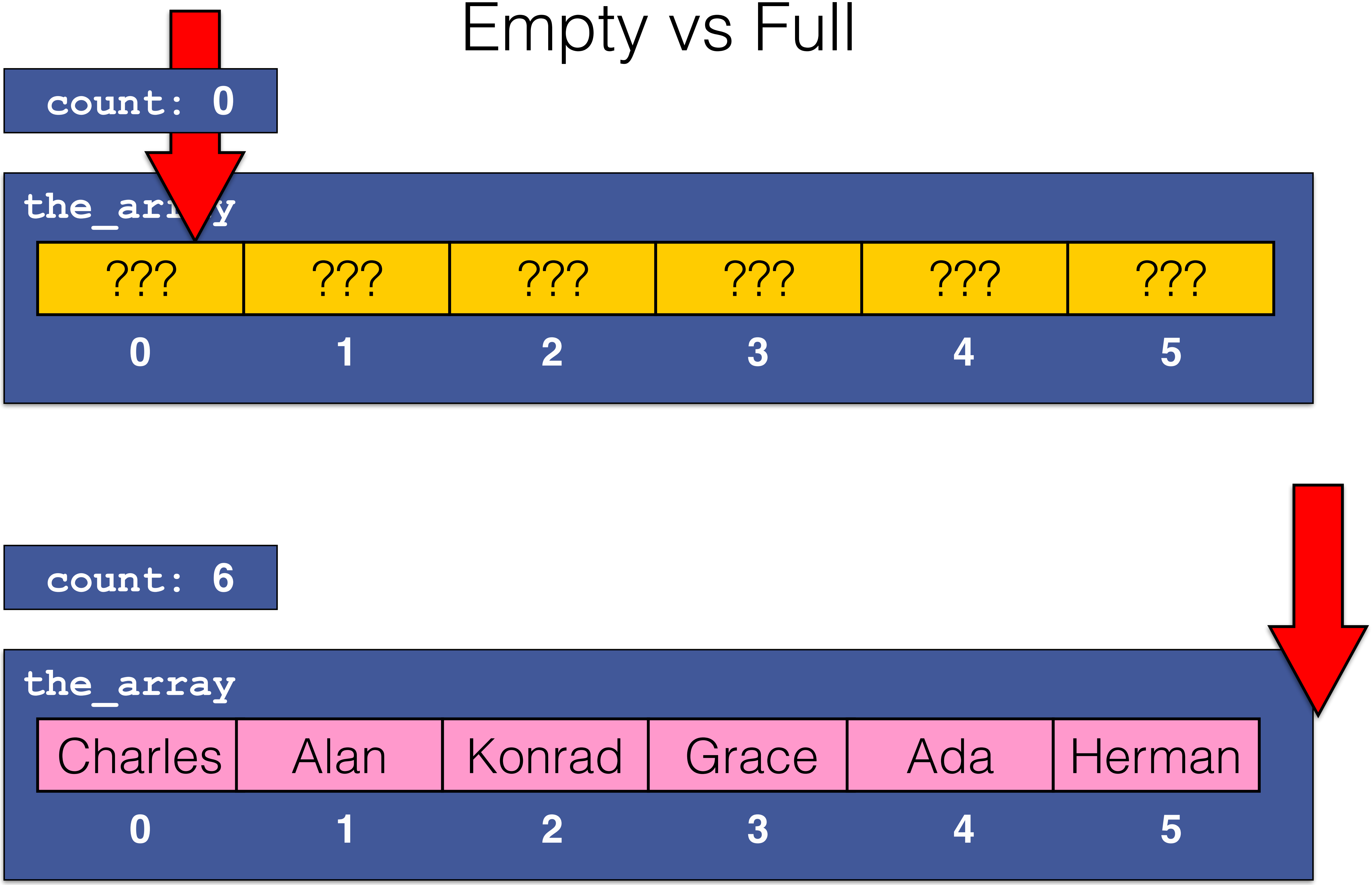




Invariant: count points to the first free position in the array

In other words: valid data appear in the 0..count-1 positions

Empty vs Full



count: 0

the_array

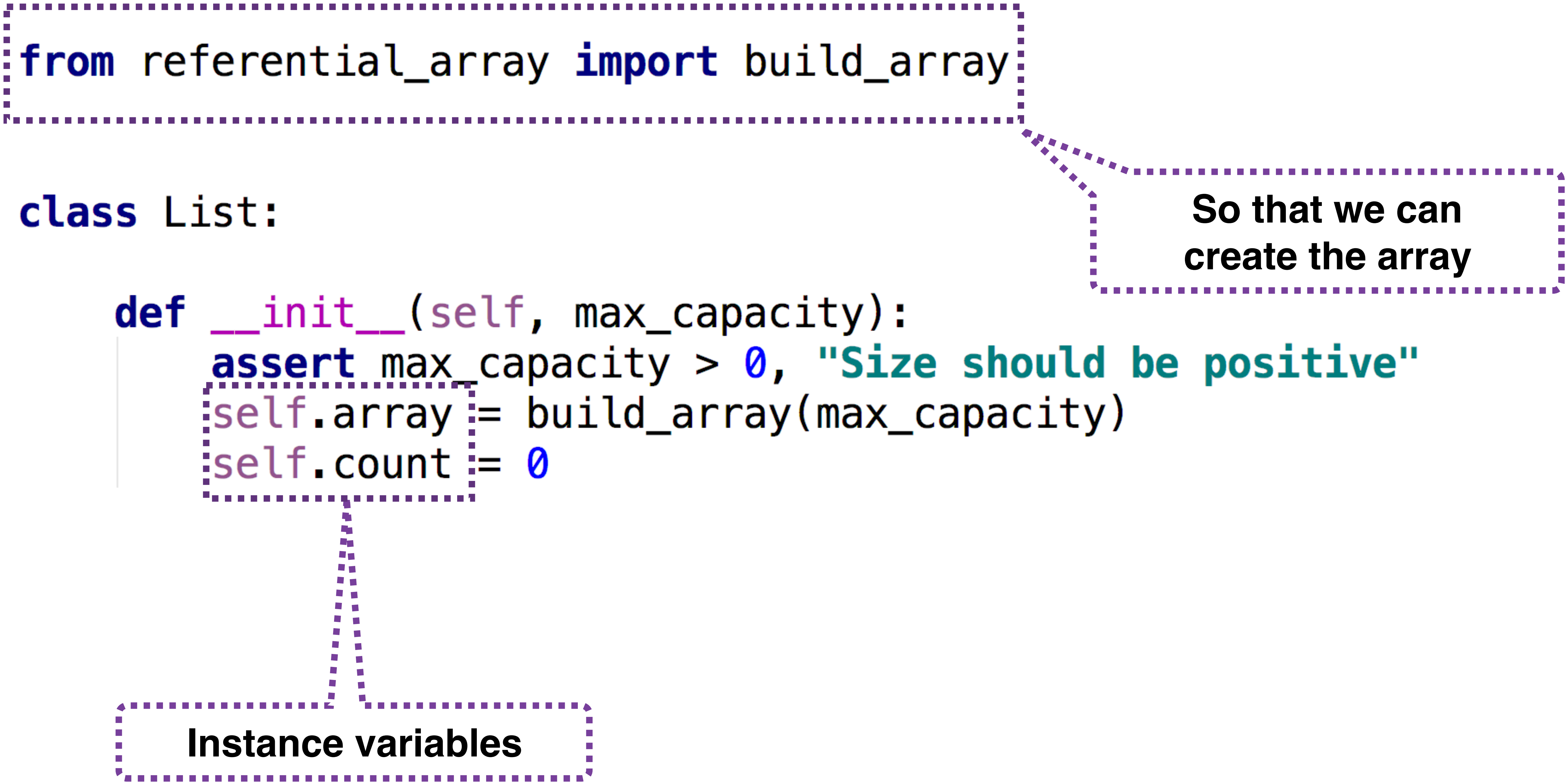
???	???	???	???	???	???
0	1	2	3	4	5

count: 6

the_array

Charles	Alan	Konrad	Grace	Ada	Herman
0	1	2	3	4	5

Creating a list

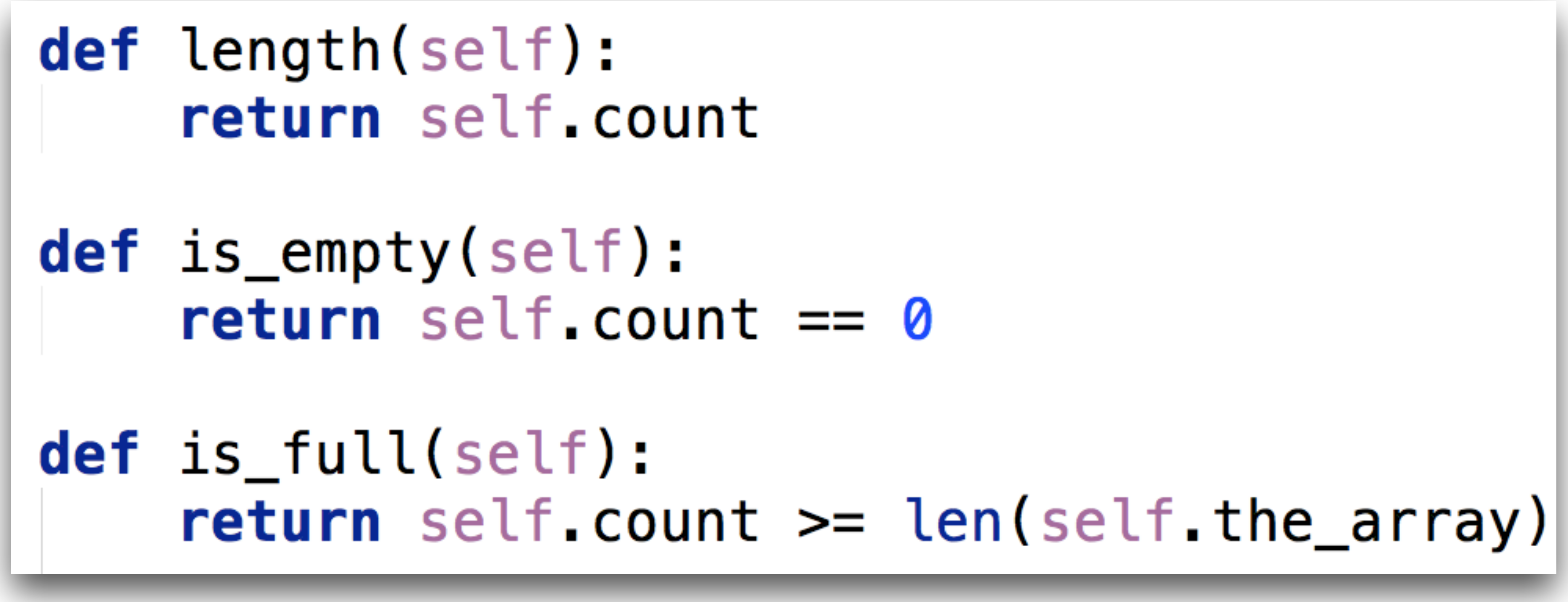


```
from referential_array import build_array

class List:

    def __init__(self, max_capacity):
        assert max_capacity > 0, "Size should be positive"
        self.array = build_array(max_capacity)
        self.count = 0
```

So that we can create the array

Instance variables

Simple methods



```python
def length(self):
    return self.count

def is_empty(self):
    return self.count == 0

def is_full(self):
    return self.count >= len(self.the_array)
```

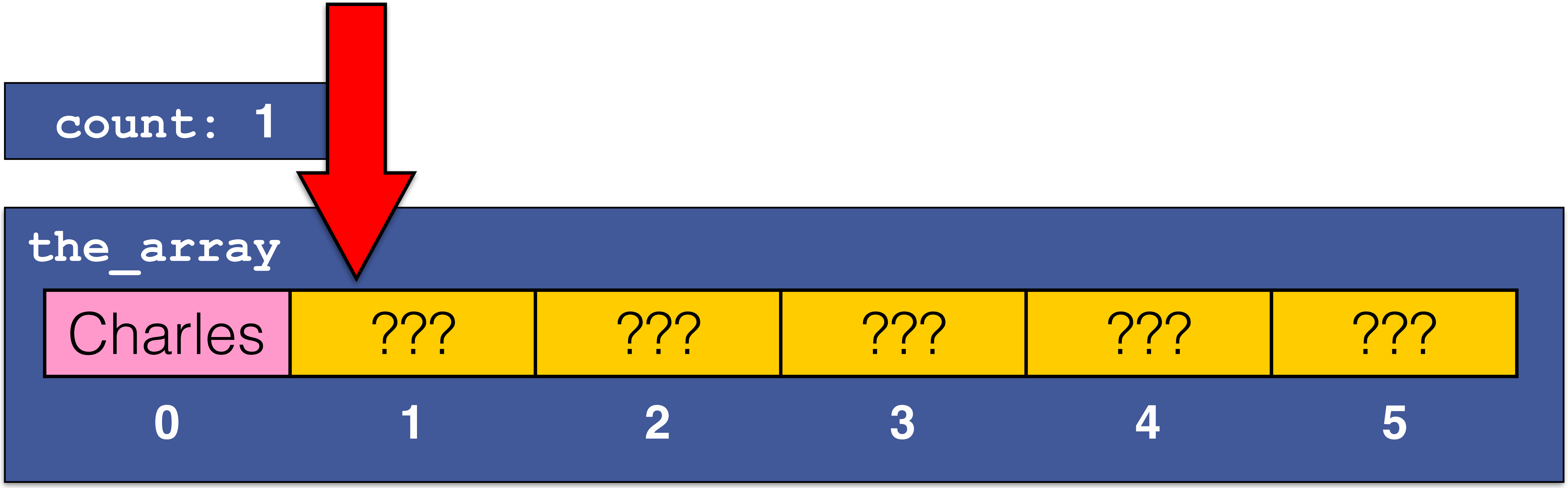
Adding an element to a list

- **Input:**
  - List (in our case: array + count)
  - **Element to be added**
- **Output:**
  - List
  - Contains all original elements in the same order AND the input one (this is the *post-condition*)

Adding an element

Recall: count indicates the first empty position (if any)

`Example: add "Ada"`



`count: 1`

`the_array`

Charles	???	???	???	???	???
0	1	2	3	4	5

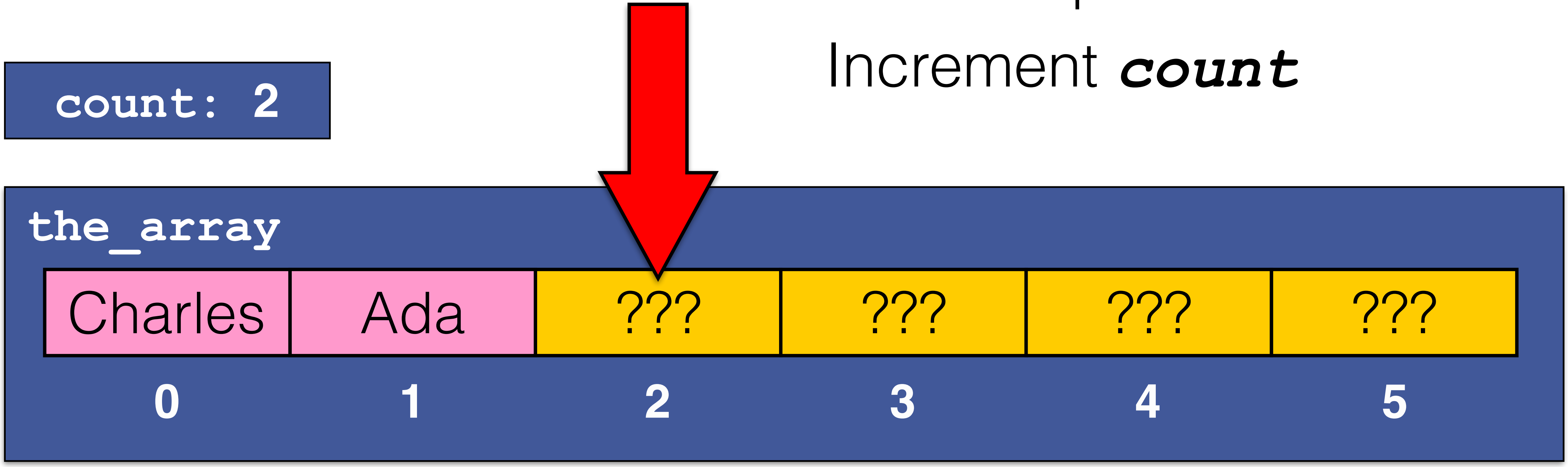
Adding an element

Recall: count indicates the first empty position (if any)

`Example: add "Ada"`

Add the item at position ***`count`***

Increment ***`count`***



`count: 2`

`the_array`

Charles	Ada	???	???	???	???
0	**1**	**2**	**3**	**4**	**5**

Adding an element

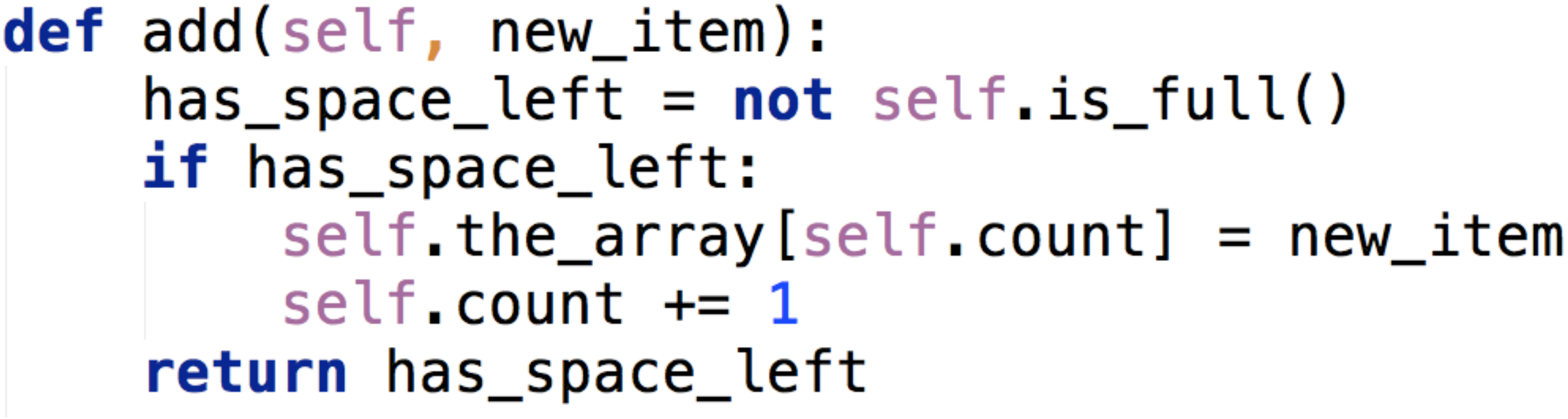
- Algorithm: add item to the_array, then increment count

- Does it always work?

- We are assuming we can always add…

- What if it is full? What to do then?
  - One possibility: return **True** if we can, **False** otherwise
  - This changes the output AND the postcondition
  - Create a new larger array copy things over?
  - What does Python do with its own lists? lists are never full…

Function add



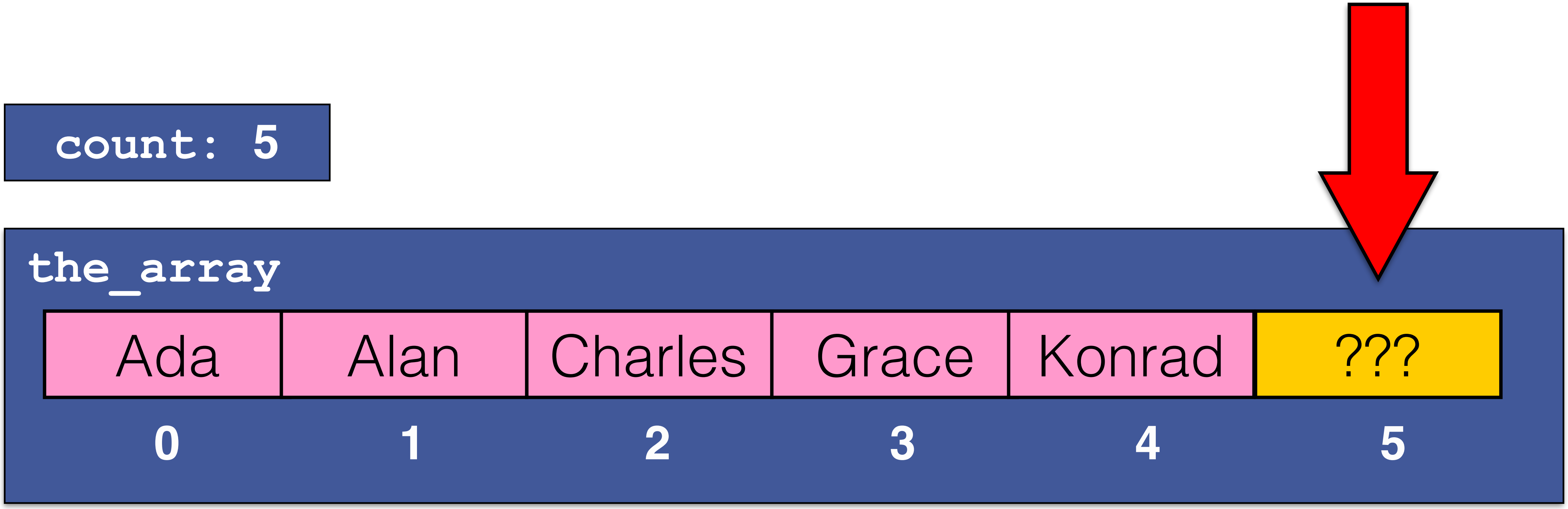
```python
def add(self, new_item):
    has_space_left = not self.is_full()
    if has_space_left:
        self.the_array[self.count] = new_item
        self.count += 1
    return has_space_left
```

What if this raises an Exception instead of returning a boolean?

Deleting an element from a list

- **Input:**
  - List (in our case: array + count)
  - **<u>Position of the element to be deleted</u>**
- **Output:**
  - List
  - Contains <u>all original elements</u> <u>EXCEPT the deleted element</u>
  - **Assume:** Remaining elements retain initial ordering.

`Example: delete item in position 2`

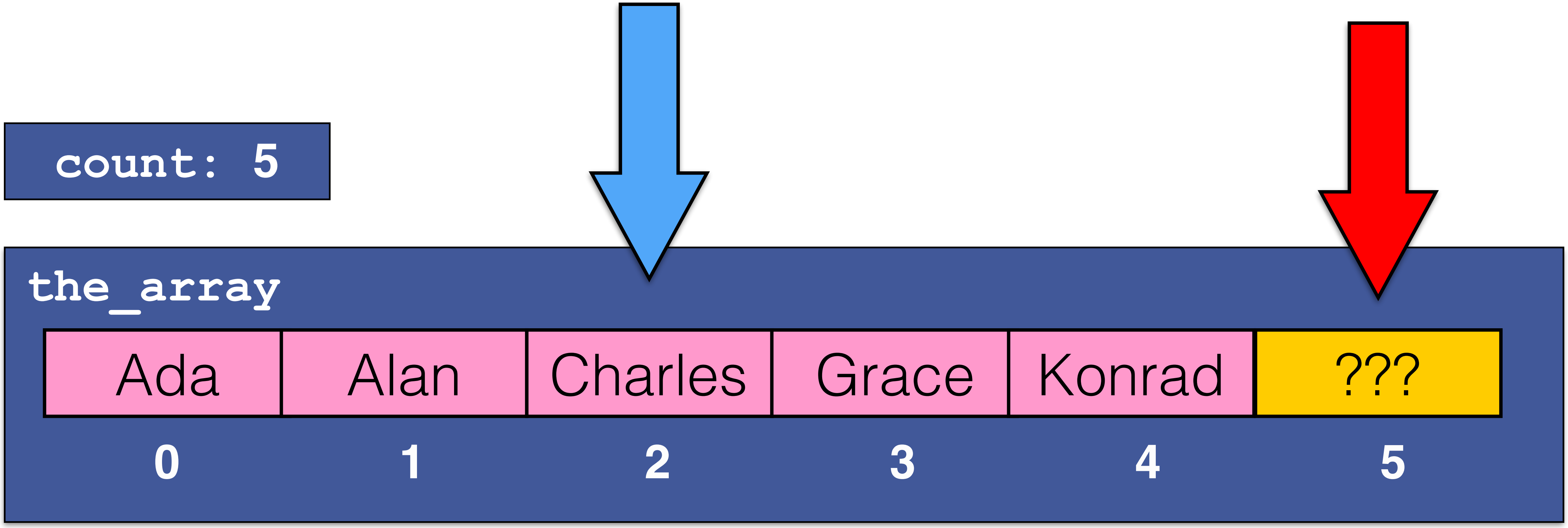


`count: 5`

`the_array`

0	1	2	3	4	5
Ada	Alan	Charles	Grace	Konrad	???

Example: delete item in position 2

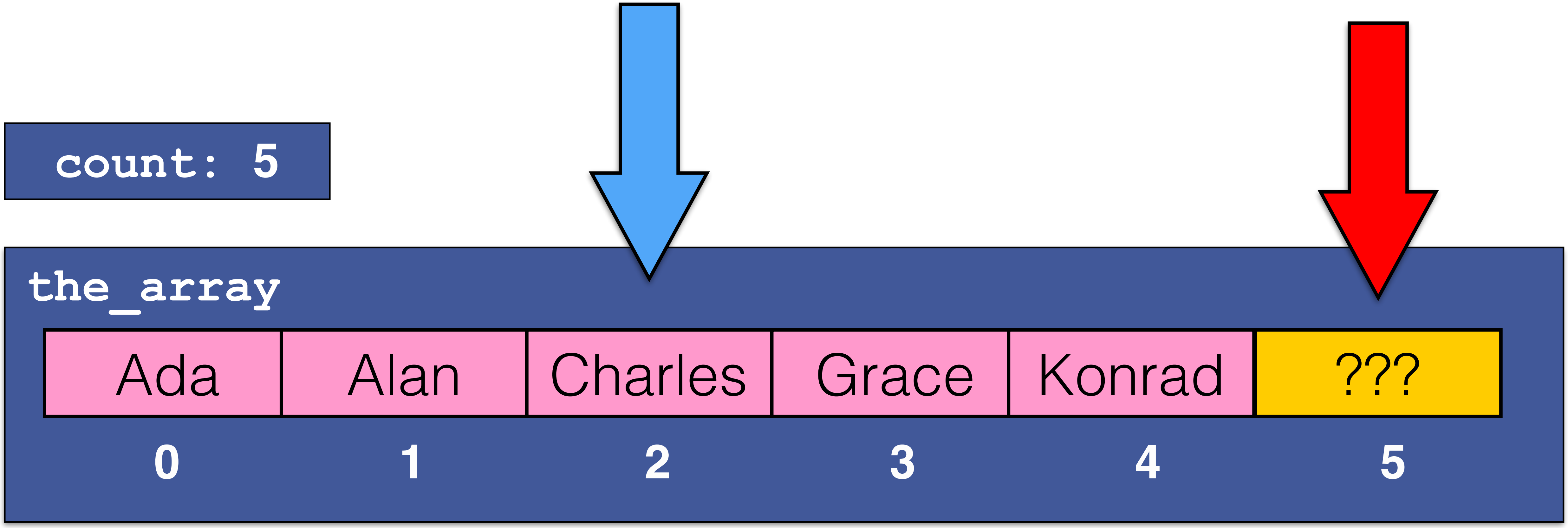


count: 5

the_array

Ada	Alan	Charles	Grace	Konrad	???
0	1	2	3	4	5

Example: delete item in position 2



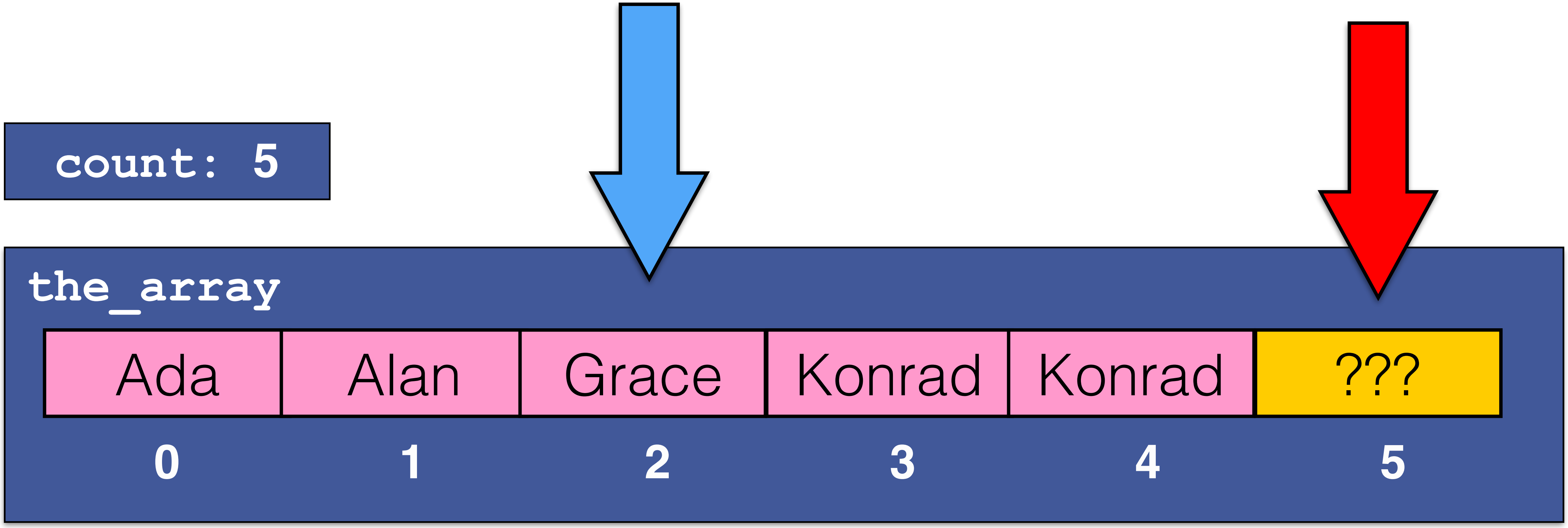
count: 5

the_array

Ada	Alan	Charles	Grace	Konrad	???
0	1	2	3	4	5

Move items appearing after the deleted item

Example: delete item in position 2



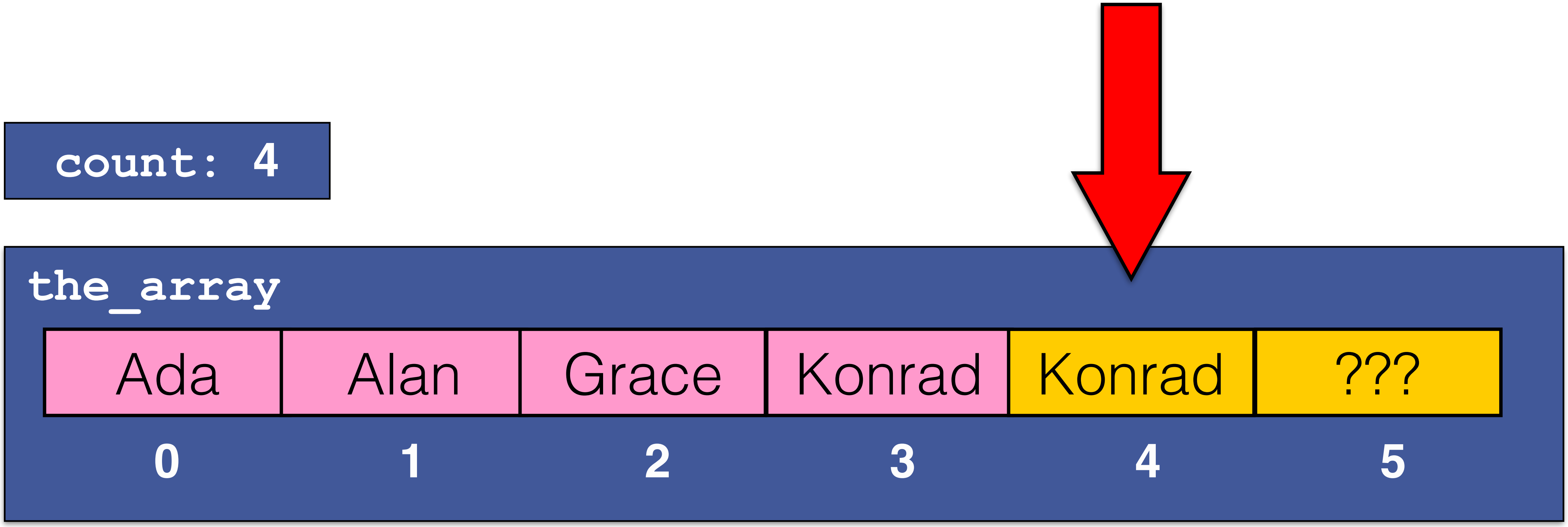
count: 5

the_array

Ada	Alan	Grace	Konrad	Konrad	???
0	1	2	3	4	5

Move items appearing after the deleted item

`Example: delete item in position 2`



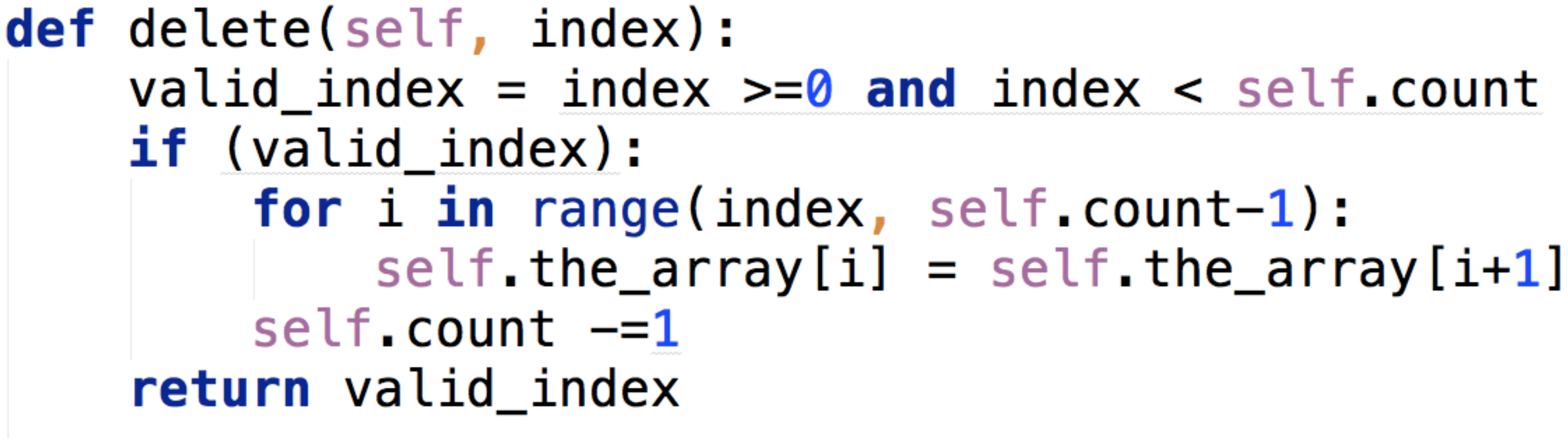
`count: 4`

`the_array`

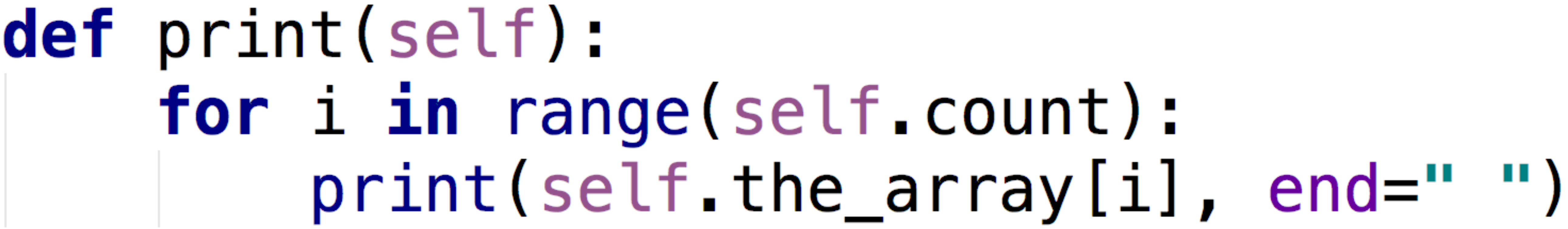
Ada	Alan	Grace	Konrad	Konrad	???
0	1	2	3	4	5

Move items appearing after the deleted item

Decrement count



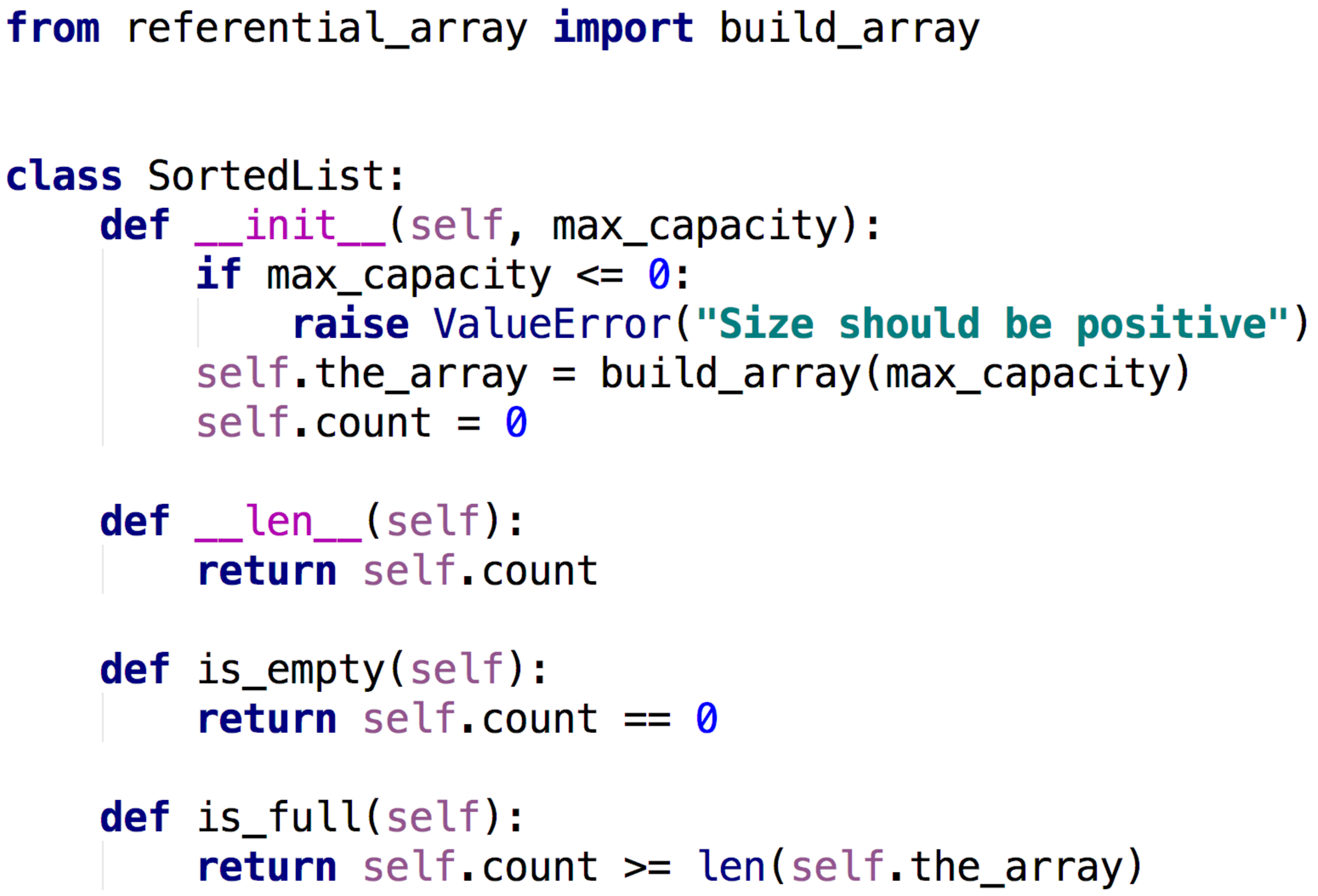
```python
def delete(self, index):
    valid_index = index >=0 and index < self.count
    if (valid_index):
        for i in range(index, self.count-1):
            self.the_array[i] = self.the_array[i+1]
        self.count -=1
    return valid_index
```



```python
def print(self):
    for i in range(self.count):
        print(self.the_array[i], end=" ")
```

SortedList ADT

- Sequence of items in increasing order
- Possible Operations:
    - **Create** a list
    - **Add item** to the list
    - **Delete** an **item** at a given position from the list
    - Check whether the list **is empty**
    - Check whether the list **is full**
    - Get the **length** of the list.



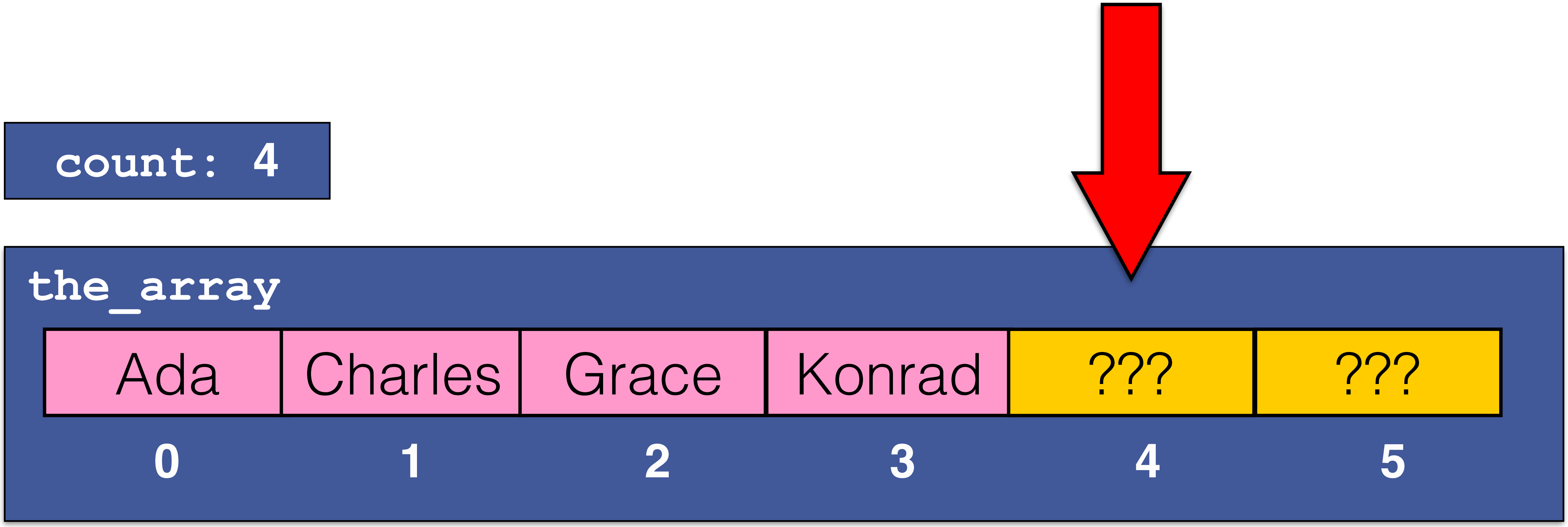
```
from referential_array import build_array


class SortedList:
    def __init__(self, max_capacity):
        if max_capacity <= 0:
            raise ValueError("Size should be positive")
        self.the_array = build_array(max_capacity)
        self.count = 0

    def __len__(self):
        return self.count

    def is_empty(self):
        return self.count == 0

    def is_full(self):
        return self.count >= len(self.the_array)
```

Adding an element to a sorted list

- **Sorted list**: Element at position i is <= than that at postion i+1

- **Input:**
  - Sorted list
  - new_item to be added
- **Output:**
  - Sorted list
  - **False** if the list was full; **True**, then the list contains all original elements in the same order together with the new_item (postcondition)
- **Note**:
  - the "Sorted" is also a pre/postcondition

Example: add “Alan”
to the sorted list.

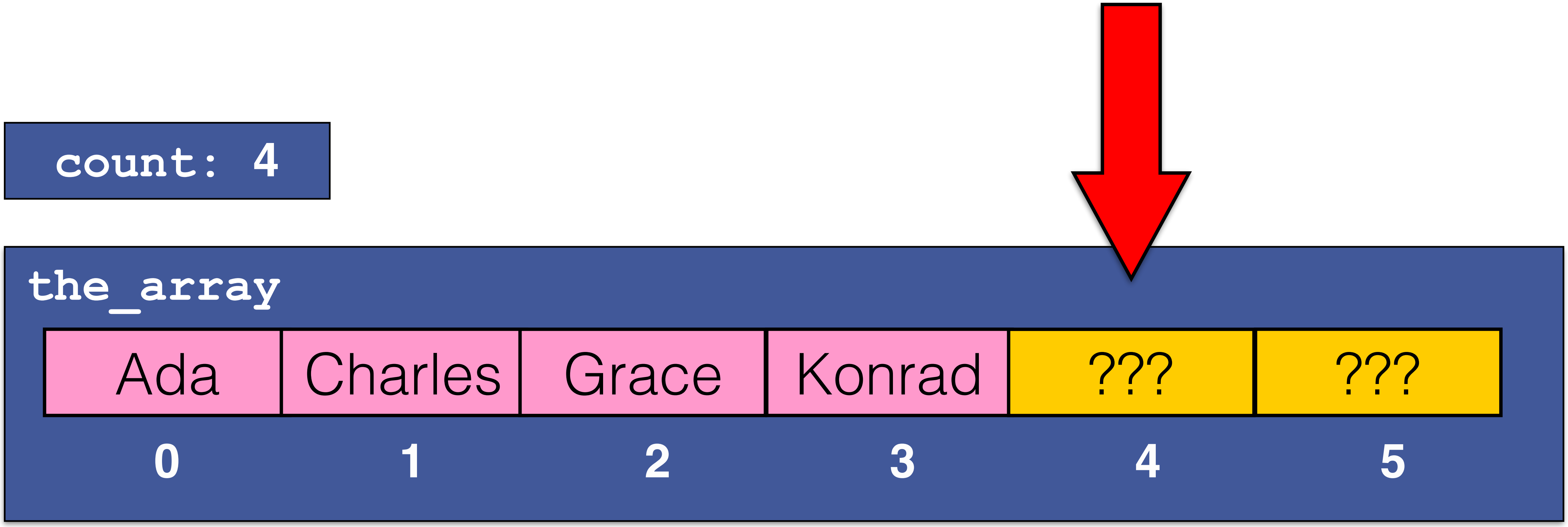


count: 4

the_array

0	1	2	3	4	5
Ada	Charles	Grace	Konrad	???	???

Example: add “Alan” to the sorted list.



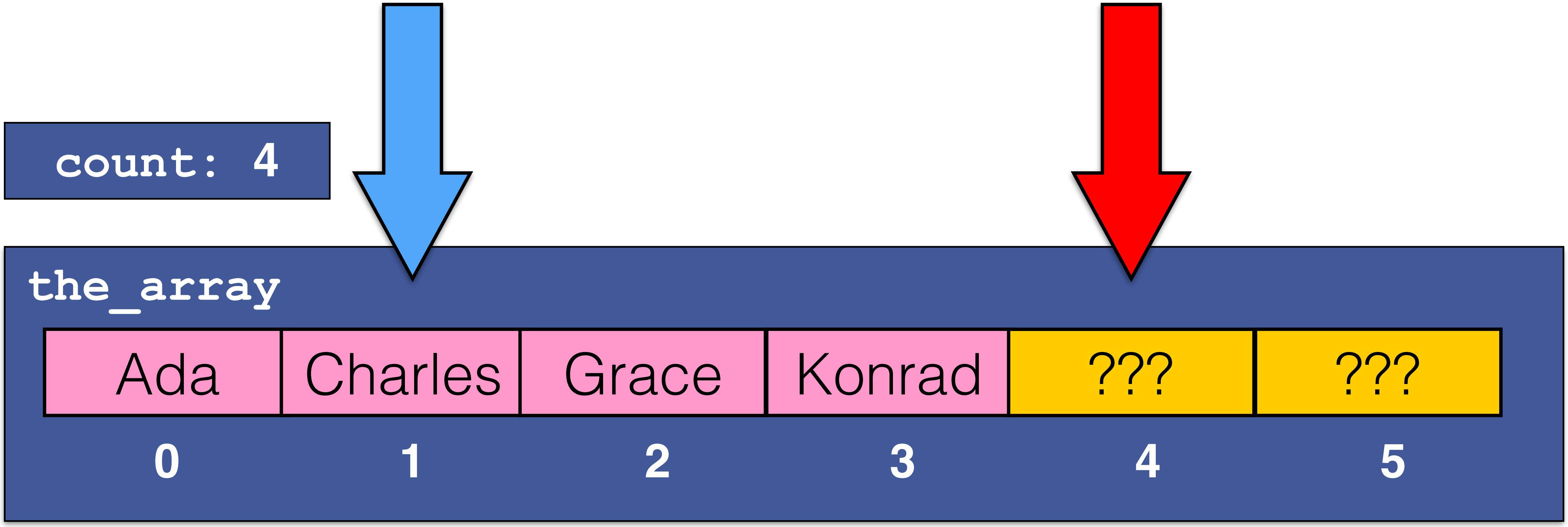
count: 4

the_array

0	1	2	3	4	5
Ada	Charles	Grace	Konrad	???	???

If there is space, find the correct position

Example: add “Alan”
to the sorted list.



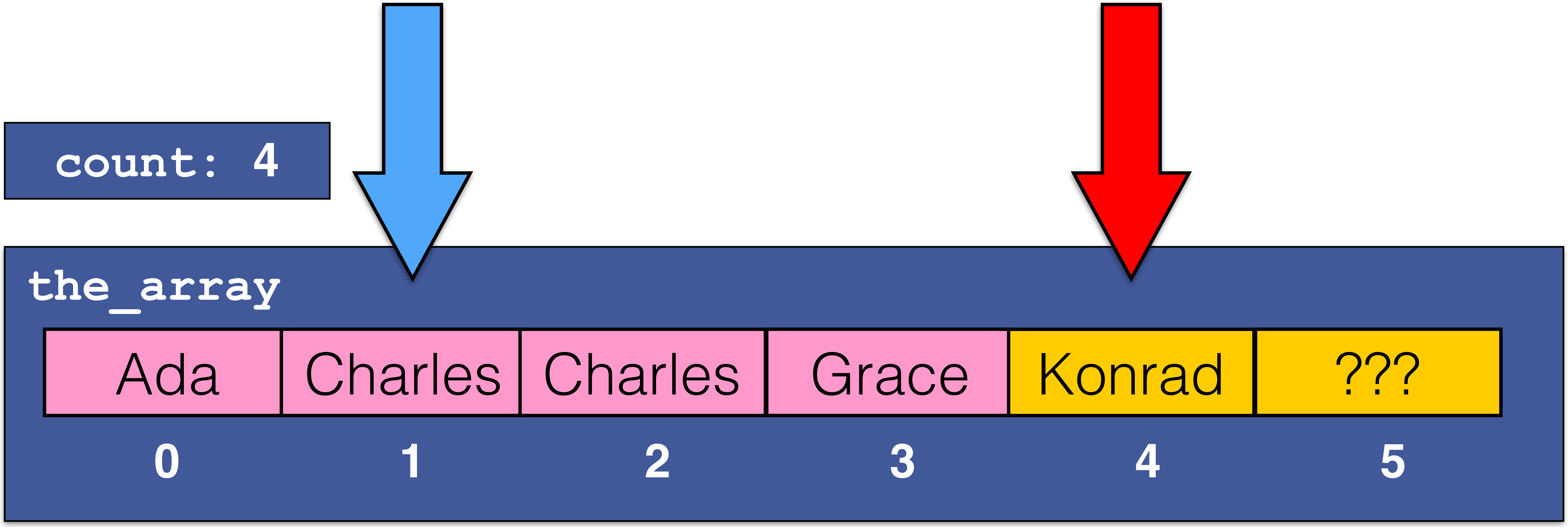
count: 4

the_array

0	1	2	3	4	5
Ada	Charles	Grace	Konrad	???	???

If there is space, find the correct position

Example: add "Alan" to the sorted list.



count: 4

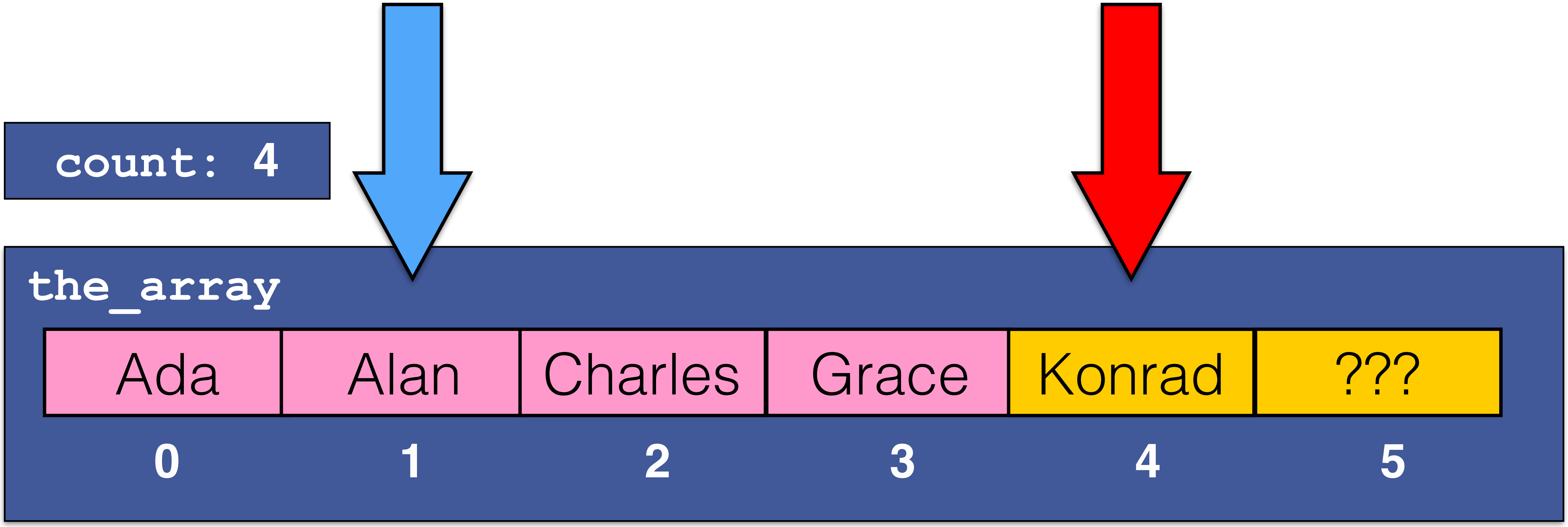
the_array

Ada	Charles	Charles	Grace	Konrad	???
0	1	2	3	4	5

If there is space, find the correct position

Make room by moving all to the right.

Example: add “Alan” to the sorted list.



count: 4

the_array

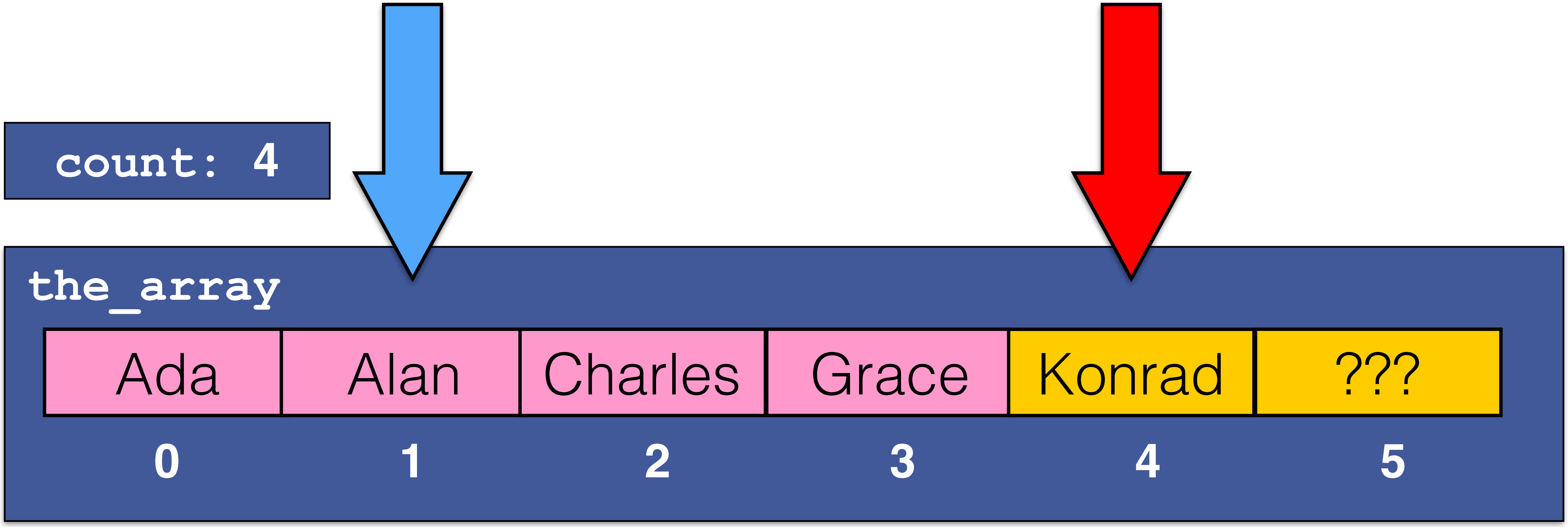
0	1	2	3	4	5
Ada	Alan	Charles	Grace	Konrad	???

If there is space, find the correct position

Make room by moving all to the right.

Put item in position.

Example: add "Alan" to the sorted list.



count: 4

the_array

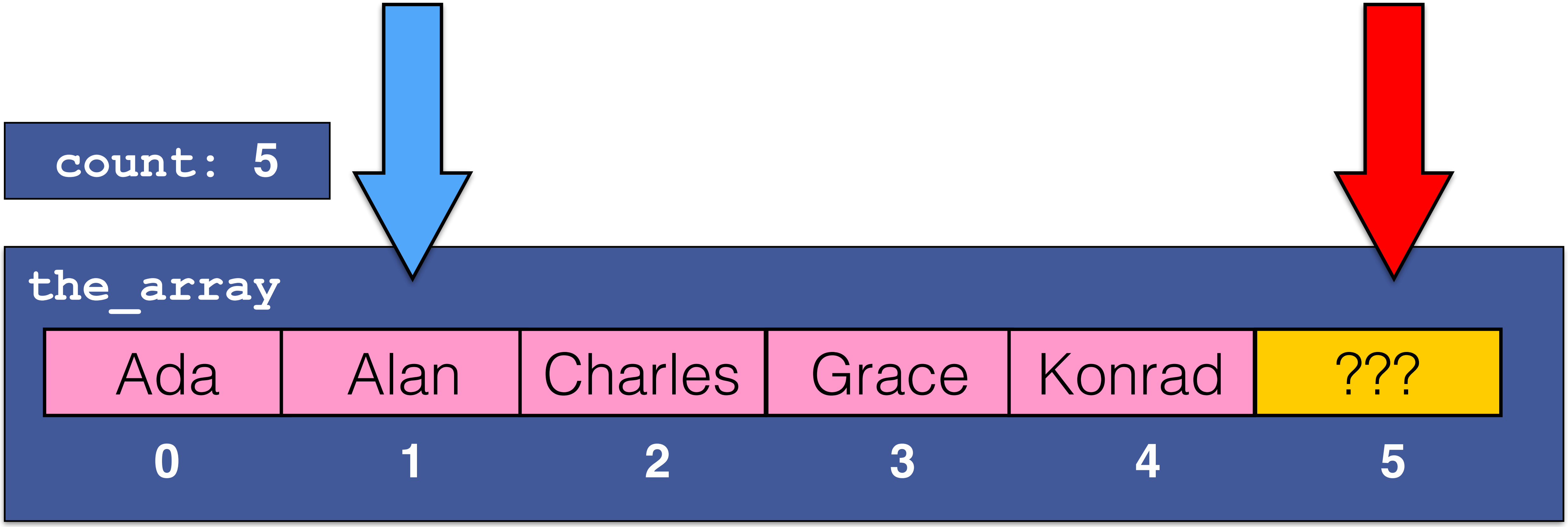
Ada	Alan	Charles	Grace	Konrad	???
0	1	2	3	4	5

If there is space, find the correct position

Make room by moving all to the right.

Put item in position.

Example: add "Alan" to the sorted list.



count: 5

the_array

0	1	2	3	4	5
Ada	Alan	Charles	Grace	Konrad	???

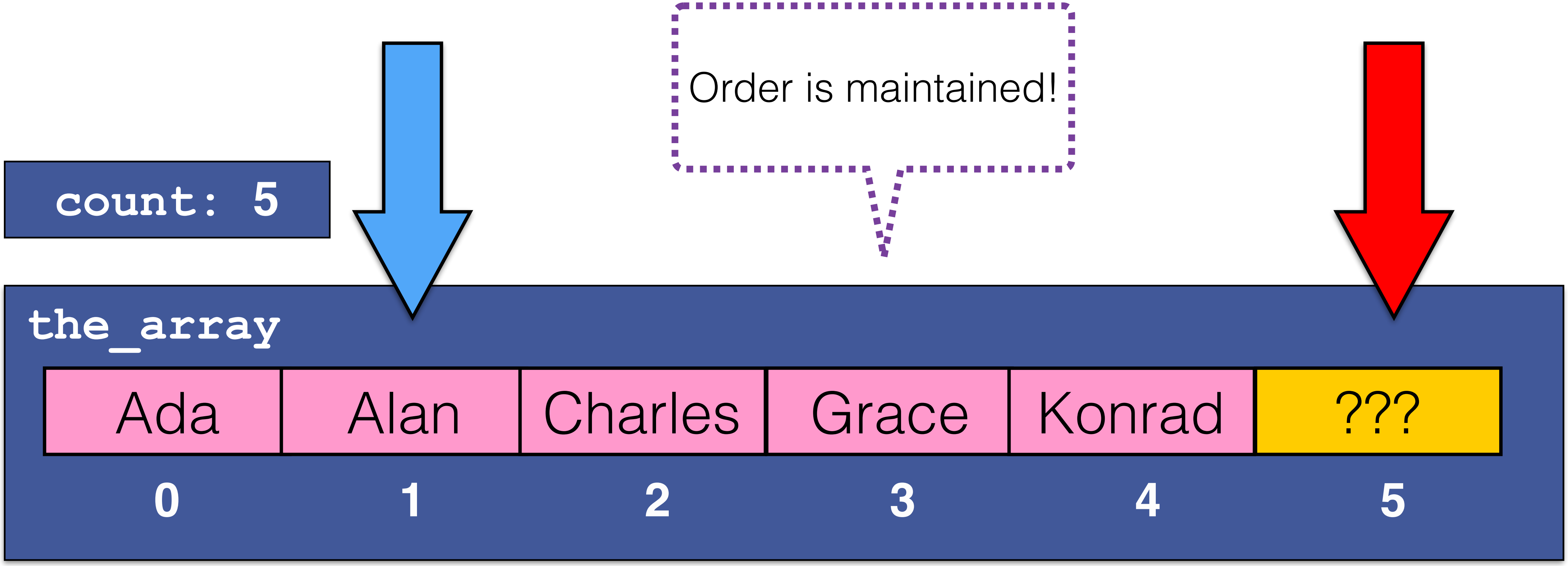
If there is space, find the correct position

Make room by moving all to the right.

Put item in position.

Update count

Example: add "Alan" to the sorted list.



Order is maintained!

count: 5

the_array

Ada	Alan	Charles	Grace	Konrad	???
0	1	2	3	4	5

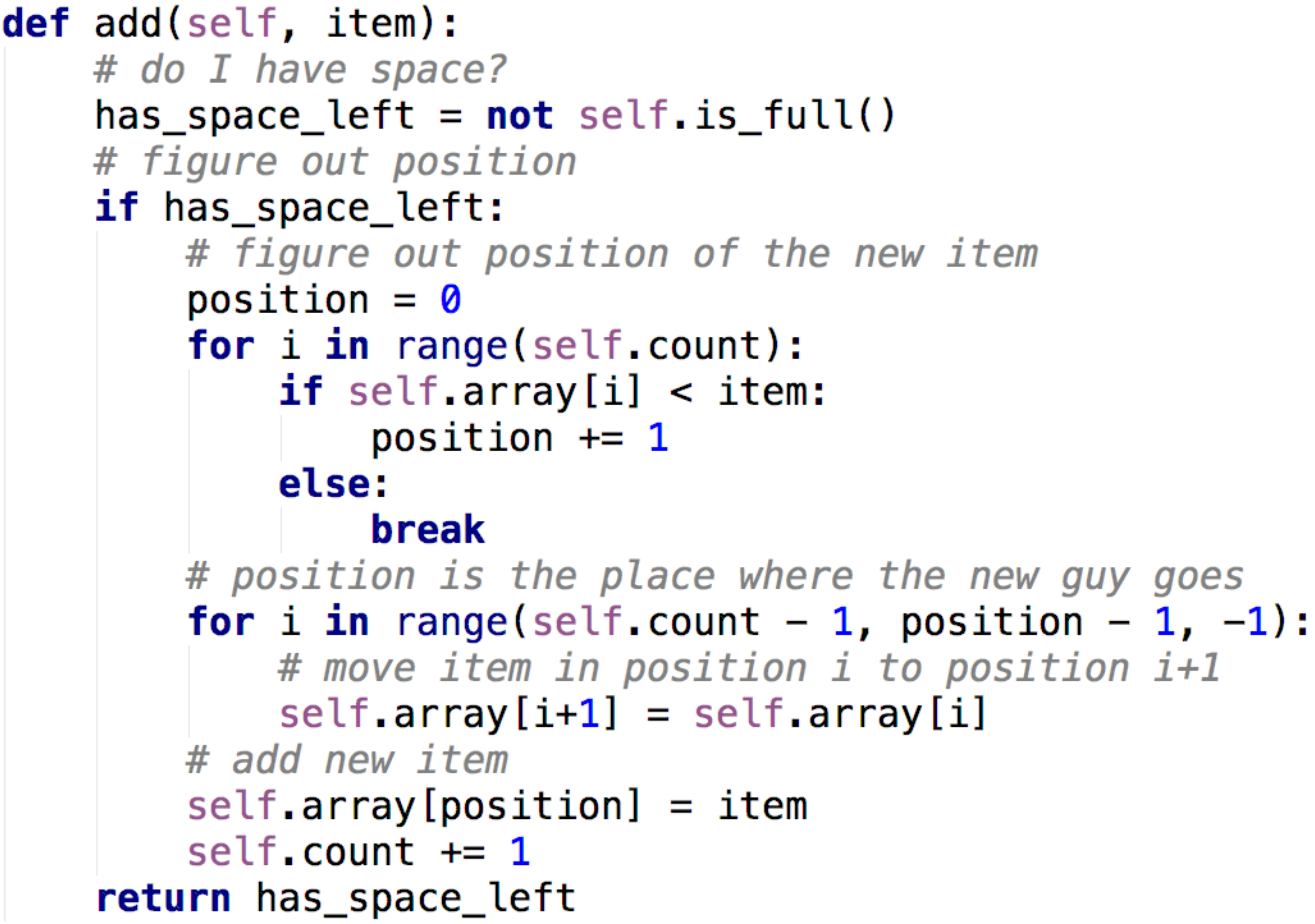
If there is space, find the correct position

Make room by moving all to the right.

Put item in position.

Update count then **return True**

```
If the array has some space left:
    find correct index at which to add item.
    make room: move all items from index to count-1 to the right
    put item in position index.
    increment count
    return True.
else:
    return False.
```



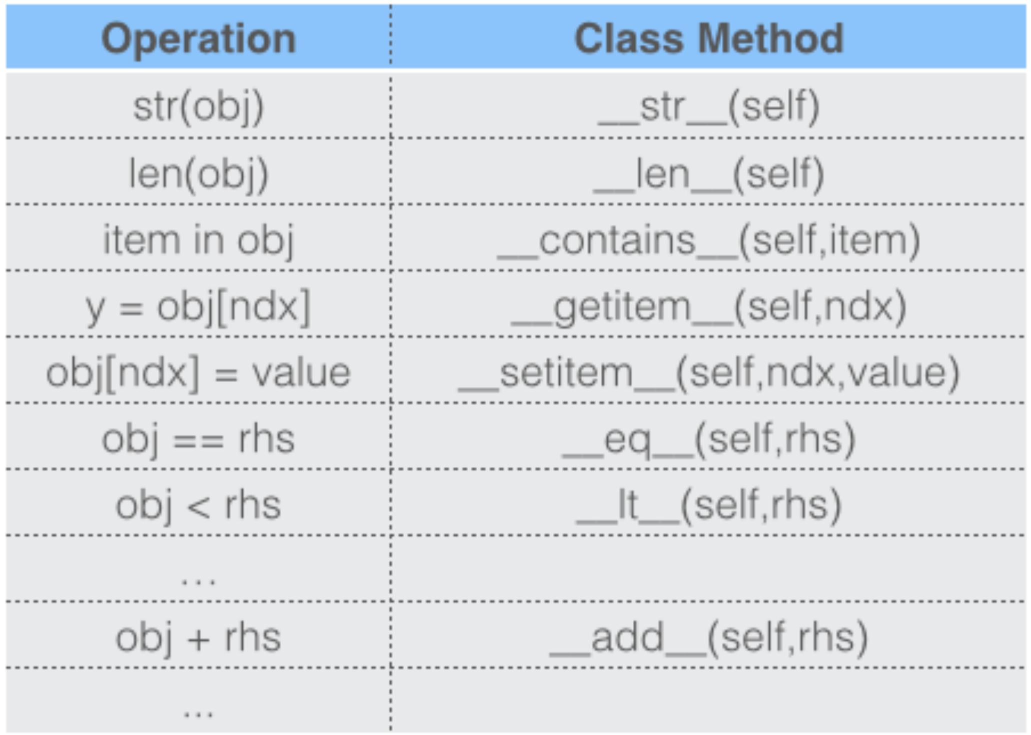
```python
def add(self, item):
    # do I have space?
    has_space_left = not self.is_full()
    # figure out position
    if has_space_left:
        # figure out position of the new item
        position = 0
        for i in range(self.count):
            if self.array[i] < item:
                position += 1
            else:
                break
        # position is the place where the new guy goes
        for i in range(self.count - 1, position - 1, -1):
            # move item in position i to position i+1
            self.array[i+1] = self.array[i]
        # add new item
        self.array[position] = item
        self.count += 1
    return has_space_left
```

Overloading operators

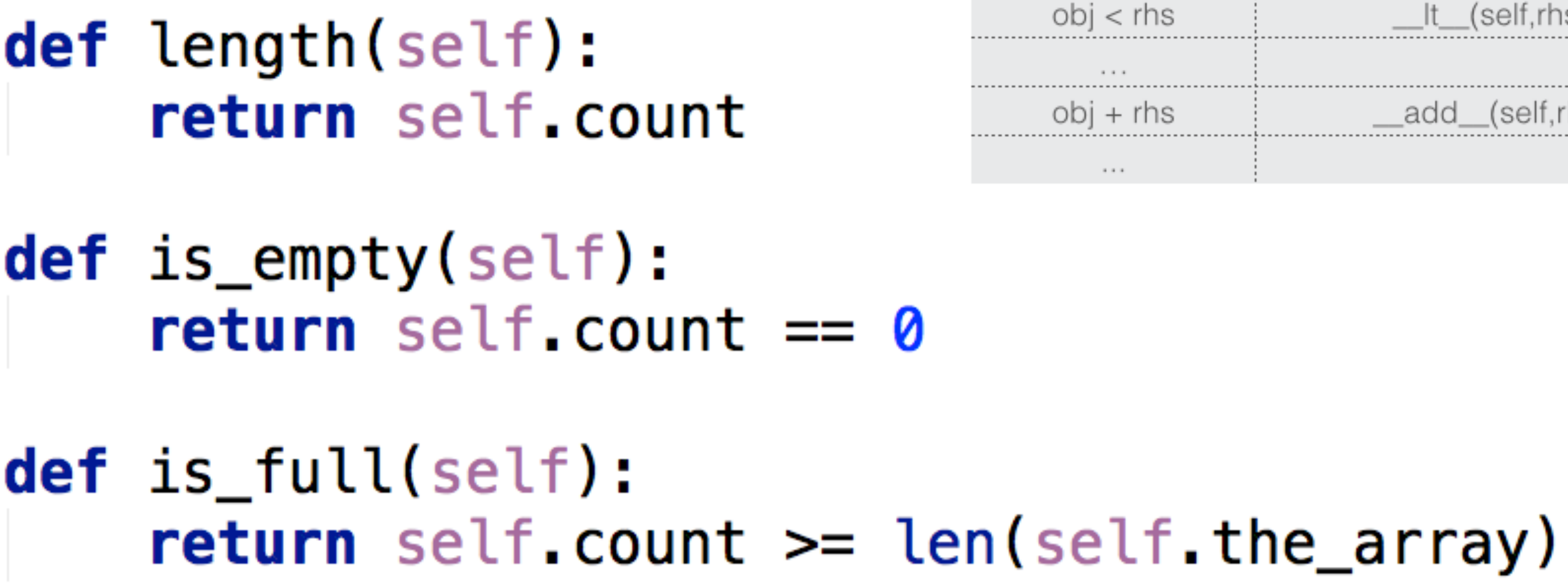
- Any class can **redefine** certain special operations:
- By simply defining the **associated method** inside the class

Operation	Class Method
str(obj)	__str__(self)
len(obj)	__len__(self)
item in obj	__contains__(self,item)
y = obj[ndx]	__getitem__(self,ndx)
obj[ndx] = value	__setitem__(self,ndx,value)
obj == rhs	
obj < rhs	
...	
obj + rhs	
...	

Python checks whether the appropriate method is available to the object. If **not defined**, the **built-in operation** (if any) is used.

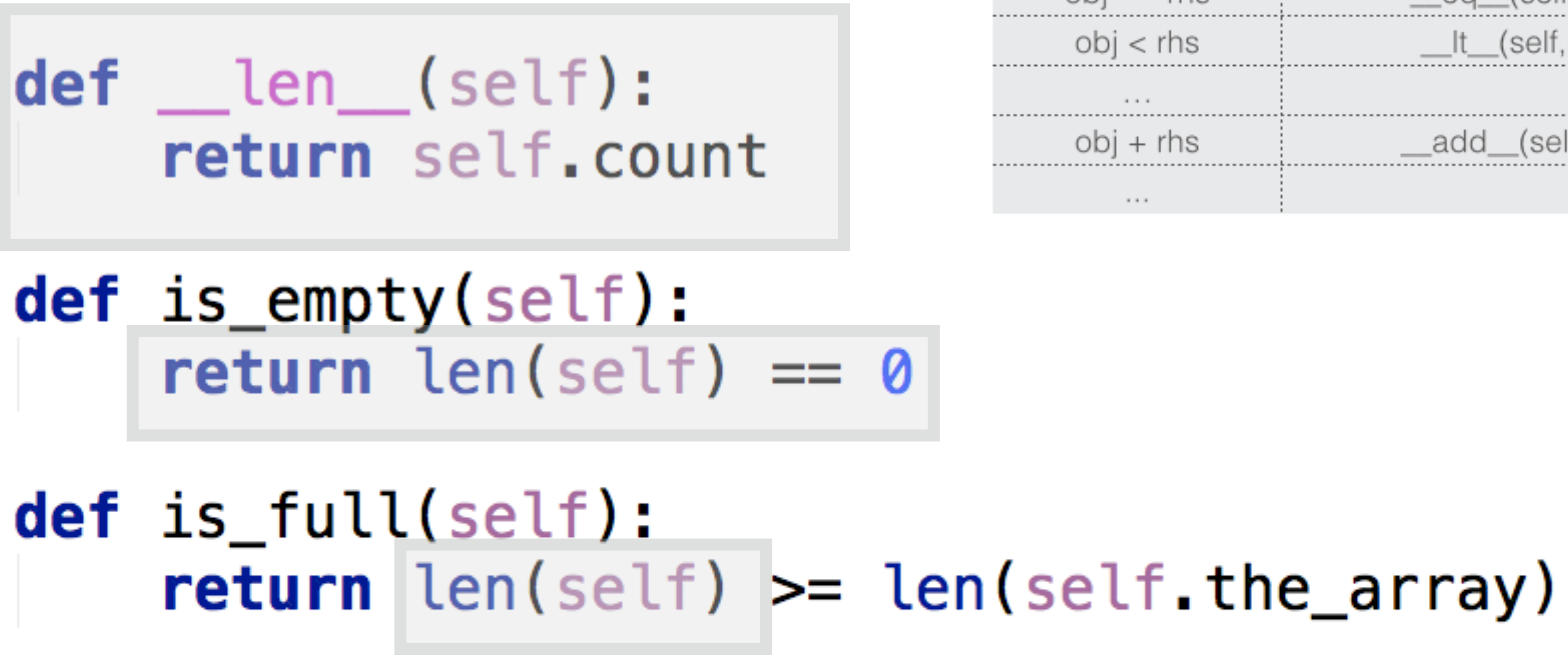


Operation	Class Method
str(obj)	__str__(self)
len(obj)	__len__(self)
item in obj	__contains__(self,item)
y = obj[ndx]	__getitem__(self,ndx)
obj[ndx] = value	__setitem__(self,ndx,value)
obj == rhs	__eq__(self,rhs)
obj < rhs	__lt__(self,rhs)
...	
obj + rhs	__add__(self,rhs)
...	



```
def length(self):
    return self.count

def is_empty(self):
    return self.count == 0

def is_full(self):
    return self.count >= len(self.the_array)
```

Operation	Class Method
str(obj)	__str__(self)
len(obj)	__len__(self)
item in obj	__contains__(self,item)
y = obj[ndx]	__getitem__(self,ndx)
obj[ndx] = value	__setitem__(self,ndx,value)
obj == rhs	__eq__(self,rhs)
obj < rhs	__lt__(self,rhs)
…	
obj + rhs	__add__(self,rhs)
…	



```python
def __len__(self):
    return self.count

def is_empty(self):
    return len(self) == 0

def is_full(self):
    return len(self) >= len(self.the_array)
```

Operation	Class Method	
str(obj)	__str__(self)	
len(obj)	__len__(self)	☑
item in obj	__contains__(self,item)	
y = obj[ndx]	__getitem__(self,ndx)	
obj[ndx] = value	__setitem__(self,ndx,value)	
obj == rhs	__eq__(self,rhs)	
obj < rhs	__lt__(self,rhs)	
…		
obj + rhs	__add__(self,rhs)	
...		

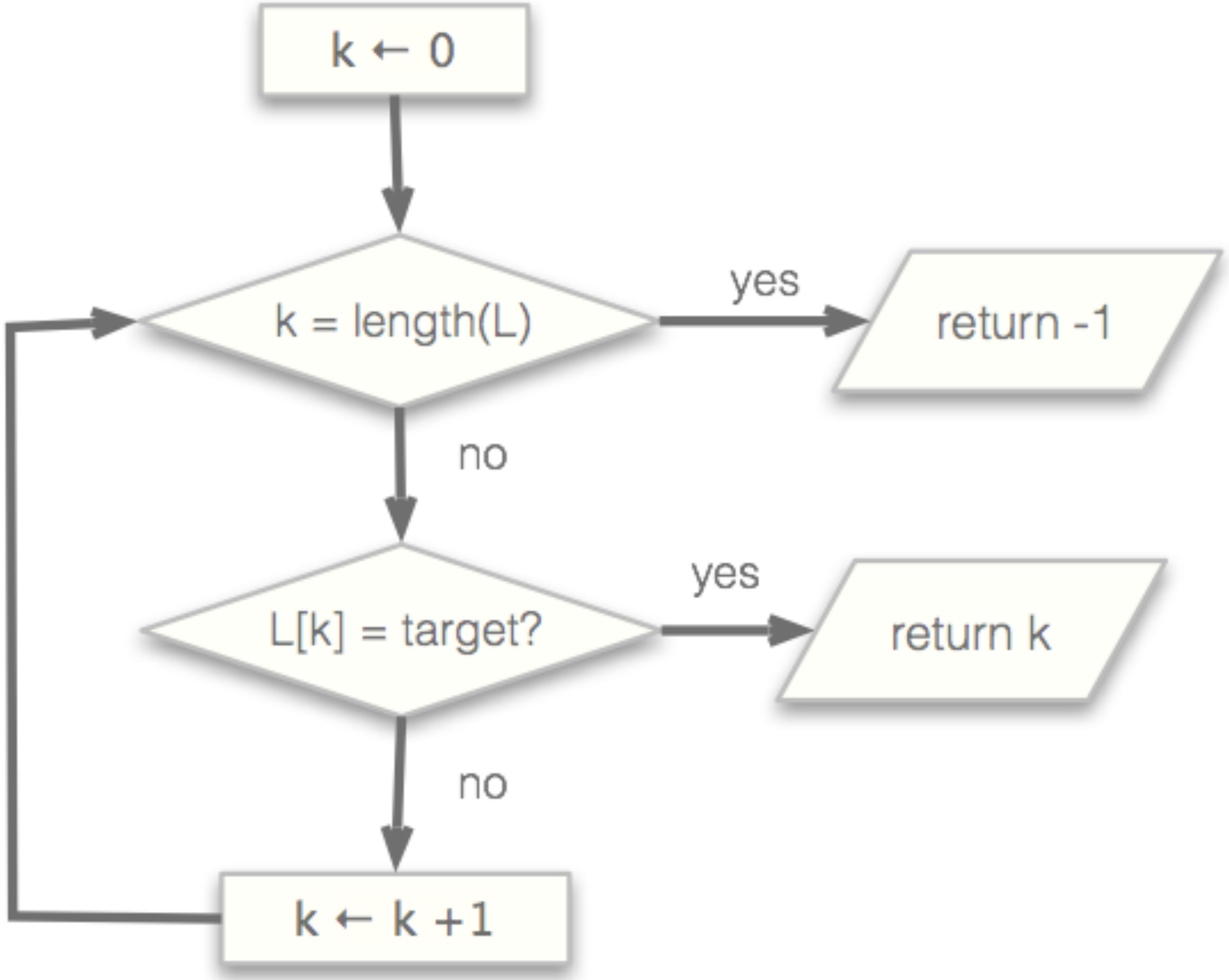
Item in List

```
>>> the_list = [1, 2, 3, 4, 5]
>>> x = 3
>>> x in the_list
True
>>> y = 8
>>> y in the_list
False
```

Item in List

```
def __contains__(self, item):
    for k in range(len(self)):
        if item == self.the_array[k]:
            return True
    return False
```

Linear Search



Can you do Binary Search
since the list is always sorted?

Summary

- Implementing lists using arrays:
  - Class structure for a list
    - Add an element to an unsorted list
    - Delete an element
  - A list that is always sorted
    - Add / Delete/ Search