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

Lecture 20
Stacks
(Array Implementation)

FIT 1008
Introduction to Computer Science

Where we were at?

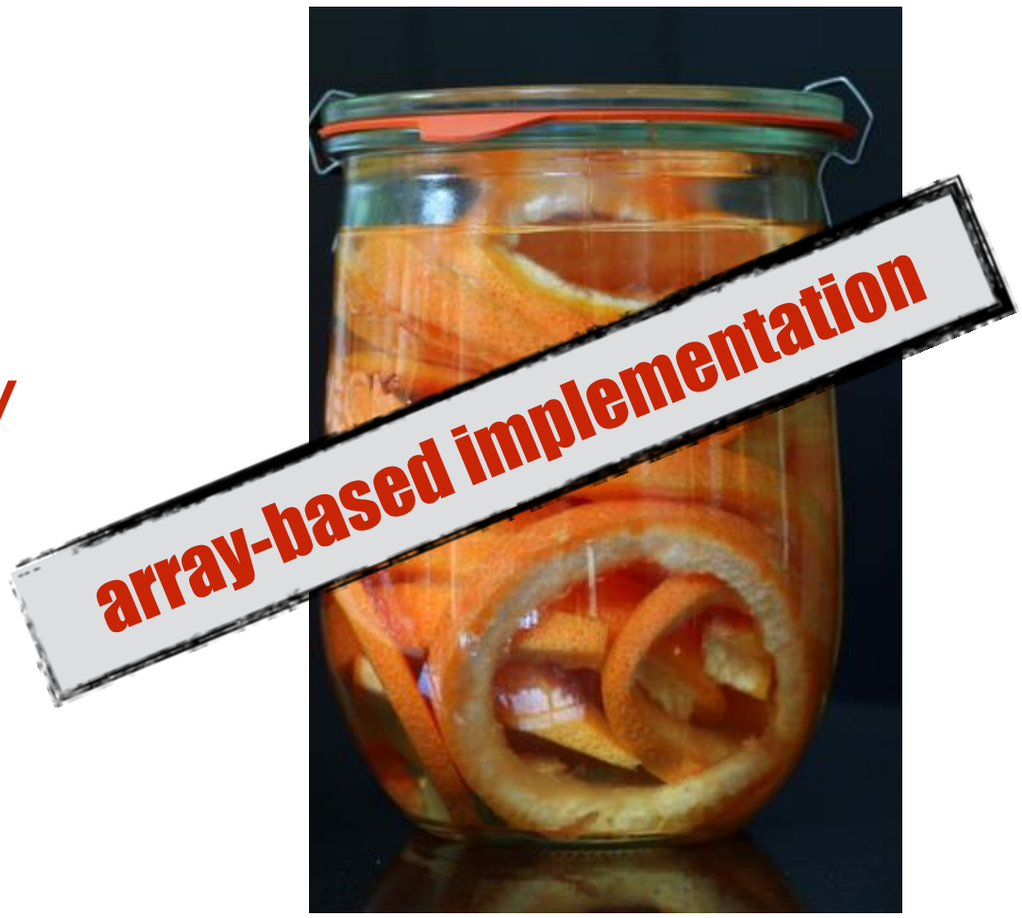
- We are now familiar with:
  - Developing simple algorithms in Python
  - Computing Big O for them
  - The concept of Abstract Data Type
  - Variable and object representation in Python
  - Mutable/immutable types
  - Basic exception handling
  - Implementing a List ADT (and Sorted List ADT) using arrays

Container ADTs

- Stores and removes items independent of contents.
- Examples include:
  - List ADT ☑
  - Stack ADT
  - Queue ADT.

← Today

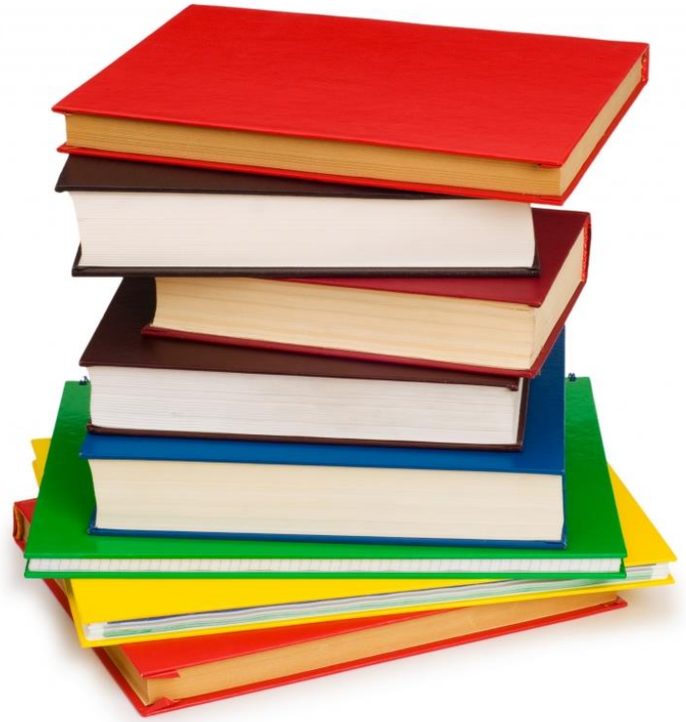
- Core operations:
  - add item
  - remove item



http://cdn.mommypotamus.com/wp-content/uploads/2014/05/IMG_3944-001.jpg

Objectives for this lecture

- To be able to implement a Stack
- To be able to use the Stack.
- To be able to reason about the complexity of the methods.



Stack

Like a list…

but…

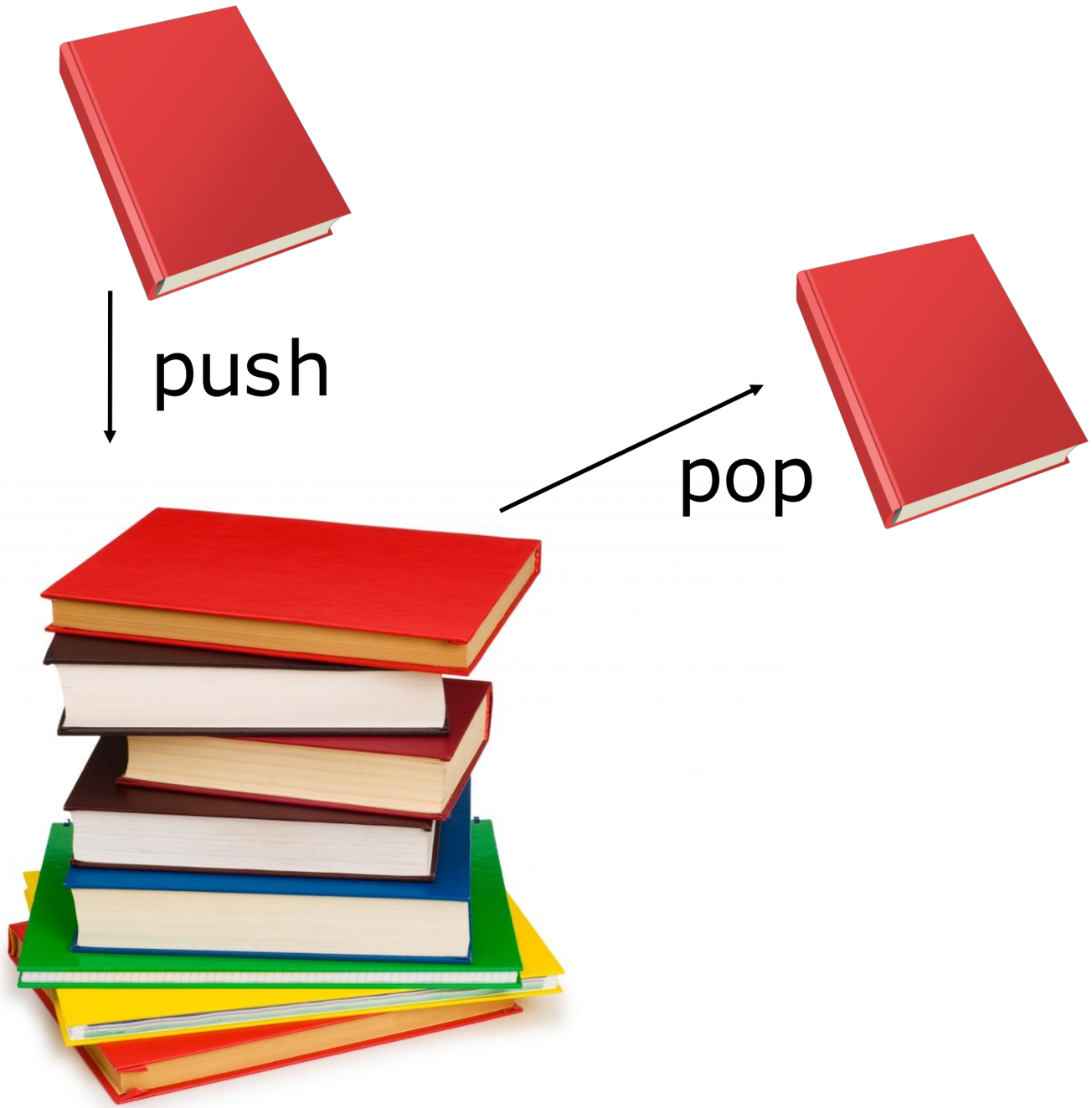
the order in which items arrive is important

LIFO

- LIFO (Last In First Out): The last element to arrive, is the first to be processed.

- The last element to be added, is the first to be deleted

- Access to any other element is unnecessary (and thus not allowed).


push
pop

Stack Data Type

- Follows a LIFO model
- Its operations (interface) are :
  - Create a stack (Stack)
  - Add an item to the top (push)
  - Take an item off the top (pop)
  - Look at the item on top, don't alter the stack (top/peek)
  - Is the stack empty?
  - Is the stack full?
  - Empty the stack (reset)

Remember: it only provides access to the element at the top of the stack (last element added)

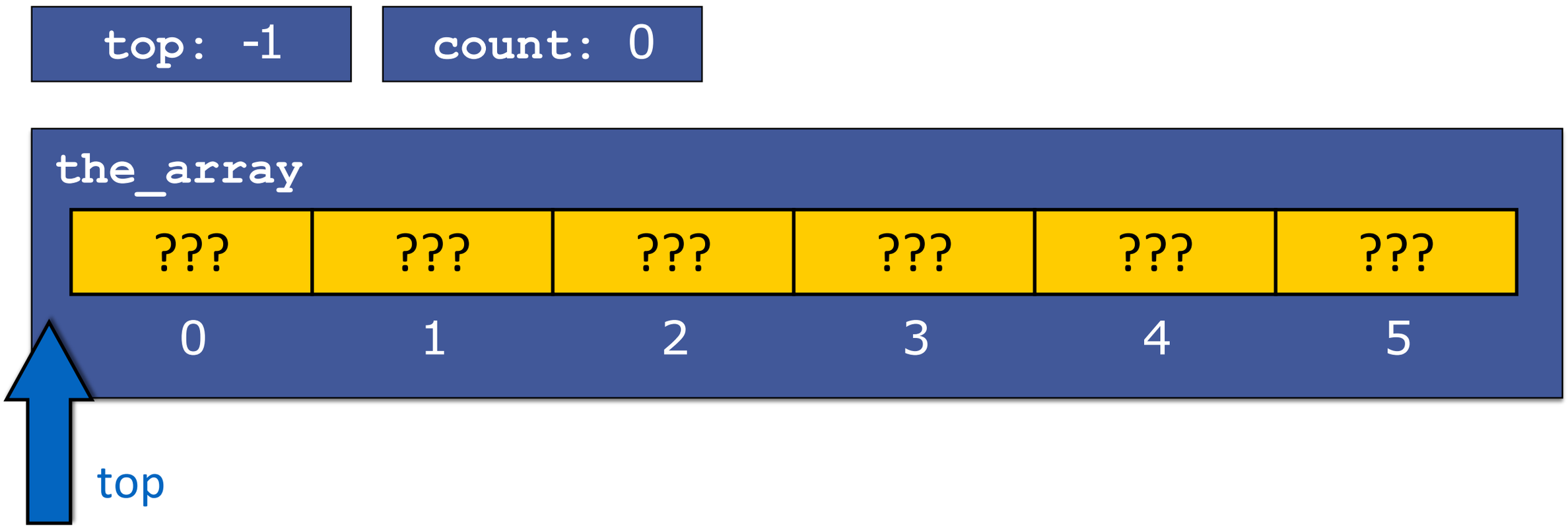
Stack implementation

- Stacks will have the following elements:
  - An array to store the items in the order in which they arrive.
  - An integer indicating how many items are in the stack.
  - An integer indicating which is the top item in the stack.

- Invariant: valid data in the `0..count-1` positions

- Pretty similar to lists, so what is the difference? The operations provided!
  - **`Stack, is_empty, is_full, size`**
  - **`push, pop, peek`**

```python
class Stack:
    def __init__(self, size):
        assert size > 0, "size should be positive"
        self.the_array = size*[None]
        self.count = 0
        self.top = -1
```



Simple methods

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

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

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

Simple methods

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

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

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

Reusing methods

Simple methods

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

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

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

def reset(self):
    self.count = 0
    self.top = -1
```

Simple methods

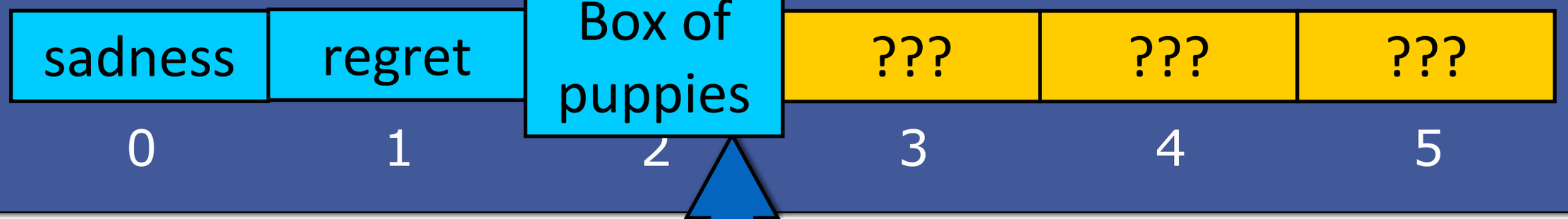
```
def size(self):
    return self.count

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

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

def reset(self):
    self.count = 0
    self.top = -1
```

Complexity is O(1)
for all of these methods.

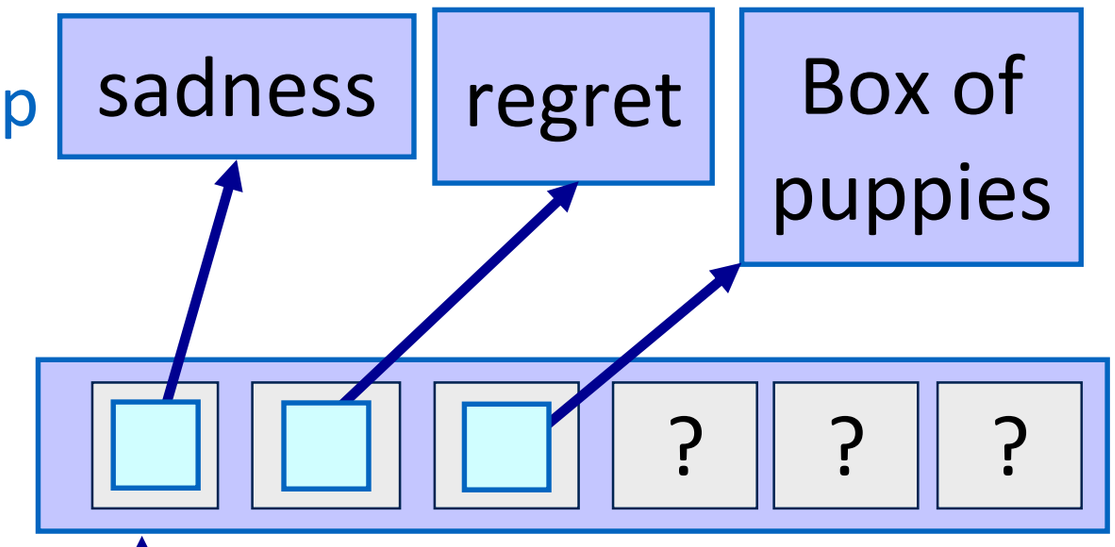
`top: 2`

`count: 3`

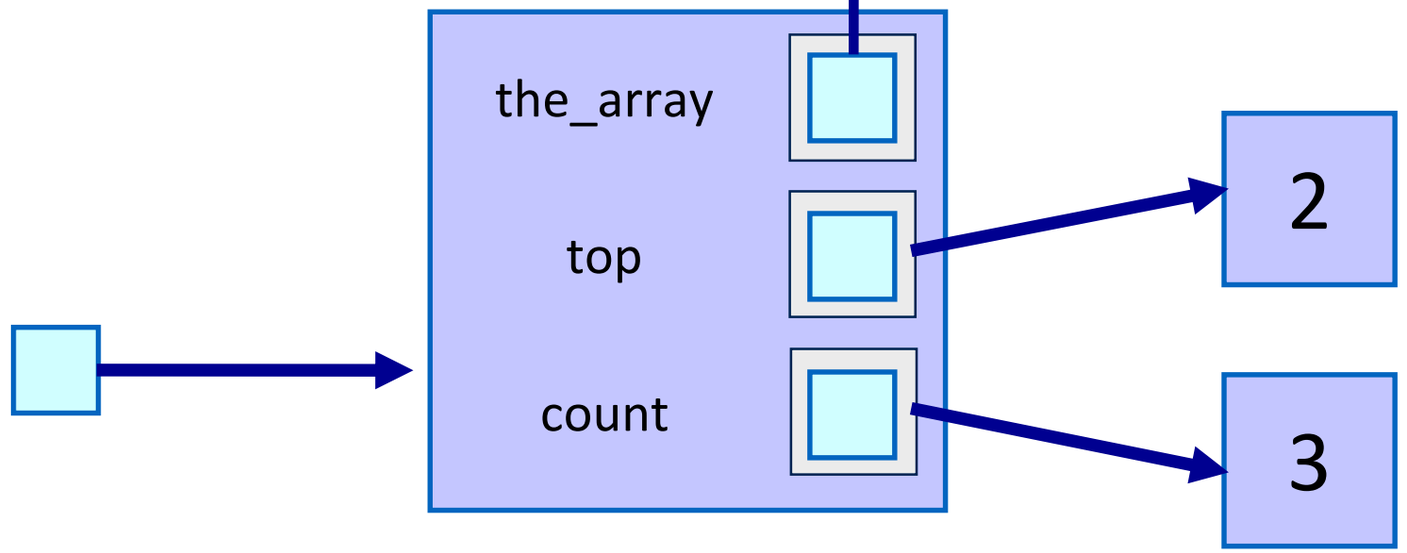
`the_array`

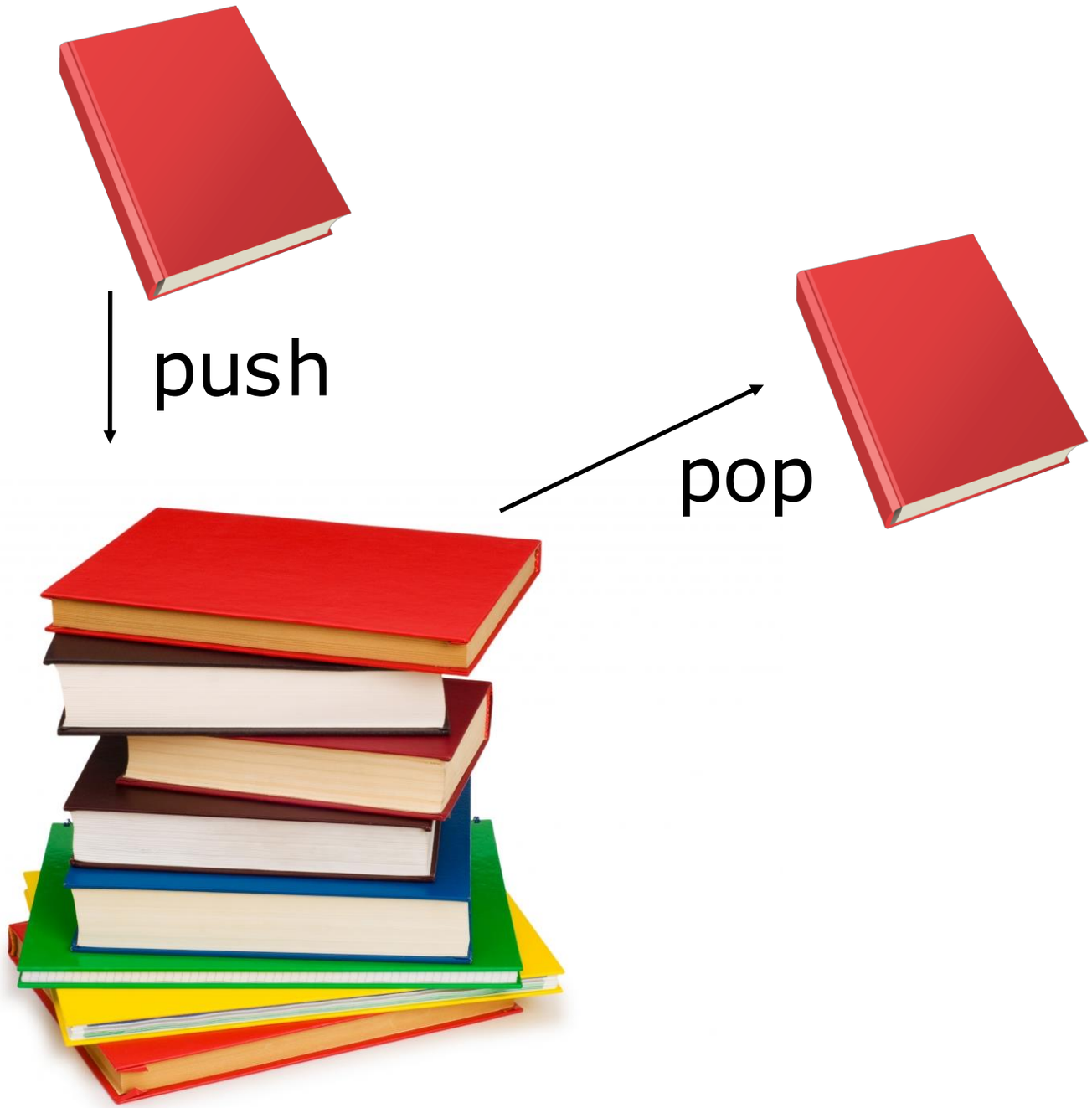


top



the_stack




push
pop

Push Implementation

- What do we do if the stack is full?

Let's check the 3 options

- Alternatives:

1. Return False (leave stack unchanged)
   - Problems may not be detected
2. Raise an AssertionError Exception
   - Assertions can be switched off
3. Raise our own Exception
   - Your code may do unnecessary checking

Push Implementation (option 1)

Reusing methods

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

Push Implementation (option 2)

Assert

Reusing methods

```
def push(self, new_item):
    assert not self.is_full(), "The stack is full."
    self.top+=1
    self.the_array[self.top] = new_item
    self.count +=1
```

Push Implementation (exception)

```
def push(self, new_item):
    if self.is_full():
        raise Exception("The stack is full")
    self.top+=1
    self.the_array[self.top] = new_item
    self.count +=1
```

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

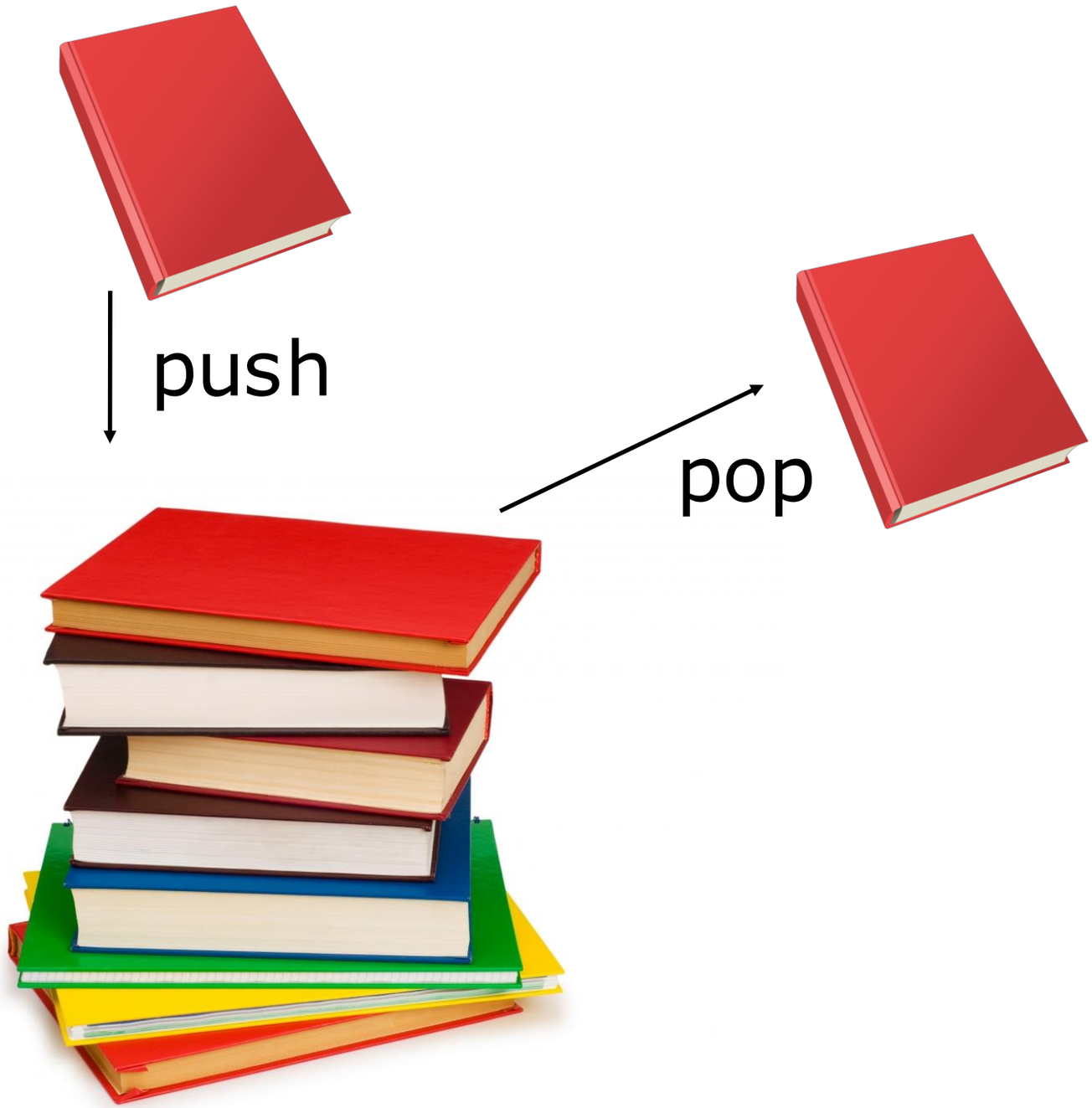
return False

```python
def push(self, new_item):
    assert not self.is_full(), "The stack is full."
    self.top+=1
    self.the_array[self.top] = new_item
    self.count +=1
```

assert

```python
def push(self, new_item):
    if self.is_full():
        raise Exception("The stack is full")
    self.top+=1
    self.the_array[self.top] = new_item
    self.count +=1
```

Exception
handling


push
pop

Reusing methods

```
def pop(self):
    if self.is_empty():
        raise Exception("The stack is empty")
    item = self.the_array[self.top]
    self.top-=1
    self.count-=1
    return item
```

Complexity is O(1)

Peek Implementation

```
def peek(self):
    assert not self.is_empty(), "The stack is empty"
item = self.the_array[self.top]
return item
```

Complexity is O(1)

Using a Stack.

Algorithm Reversing elements

Read in a list of items and print them out in reverse order

Input: A list of items

Output: A list of items in reverse order

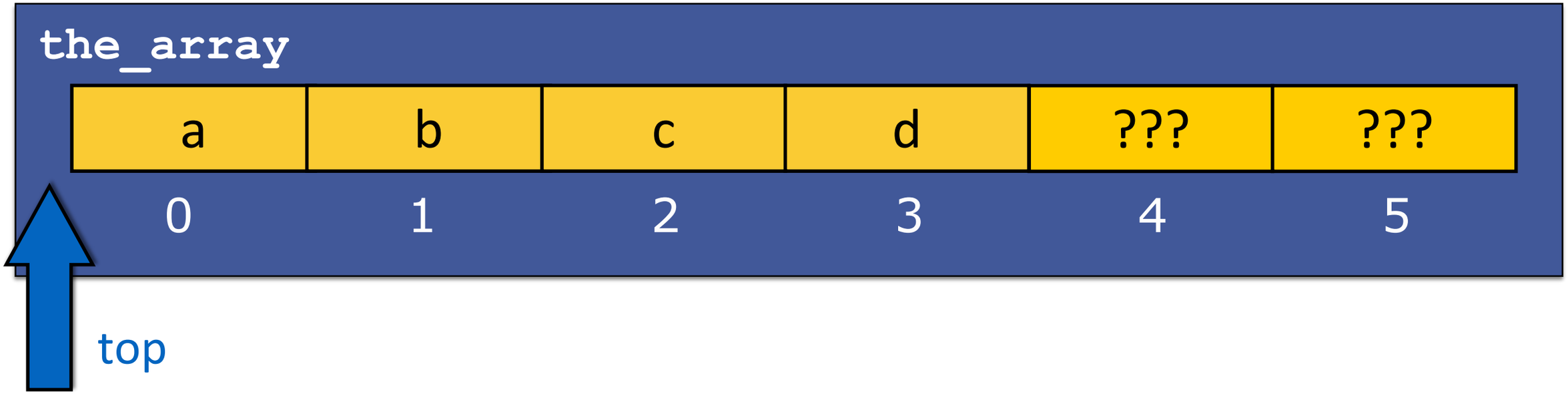
```
Clear the stack

While (there is some input)
        Read the next item.
        Push the next item onto the stack.

While (the stack is not empty)
        Pop the top item from the stack
        Print this item
```

- Stack: Last-In-First-Out

Input: String “abcd”

```
the_stack.push(a)
the_stack.push(b)
the_stack.push(c)
the_stack.push(d)
the_stack.pop()
the_stack.pop()
the_stack.pop()
the_stack.pop()
```

`top: -1` `count: 0`



Output: `d c b a`

Example: reversing a sequence of chars

- Create a stack of the appropriate size
  - Use len(string) to compute the length of the input string
- Traverse the input string pushing each char onto the stack
- Initialise the *output* string to empty " "
- Pop each element from the stack and concatenate it to the output string
- You are a user of the stack ADT
  - You have no idea how it is implemented
  - You use methods, NOT the knowledge about how it is implemented with arrays

```
from lecture_17 import Stack

def reverse_string(my_string):
```

```python
from lecture_17 import Stack


def reverse_string(my_string):
    # create a stack of appropriate size
    string_size = len(my_string)
    my_stack = Stack(string_size)
    # push each character into the stack
    for i in range(0, string_size):
        my_stack.push(my_string[i])

    # create empty output string
    ans = ""
    # pop from the stack
    while not my_stack.is_empty():
        ans = ans + my_stack.pop()
    # ans contains the reversed string
    return ans

if __name__ == "__main__":
    my_string = input("Please enter a string: ")
    result = reverse_string(my_string)
    print(result)
```

```python
from lecture_17 import Stack


def reverse_string(my_string):
    # create a stack of appropriate size
    string_size = len(my_string)
    my_stack = Stack(string_size)
    # push each character into the stack
    for i in range(0, string_size):
        my_stack.push(my_string[i])

    # create empty output string
    ans = ""
    # pop from the stack
    while not my_stack.is_empty():
        ans = ans + my_stack.pop()
    # ans contains the reversed string
    return ans

if __name__ == "__main__":
    my_string = input("Please enter a string: ")
    result = reverse_string(my_string)
    print(result)
```

```python
from lecture_17 import Stack


def reverse_string(my_string):
    # create a stack of appropriate size
    string_size = len(my_string)
    my_stack = Stack(string_size)
```

```
Lectures — bash — 84×15
dyn-49-127-69-0:Lectures juliangarcia$ python string_reverse.py
Please enter a string: anna
anna
dyn-49-127-69-0:Lectures juliangarcia$ python string_reverse.py
Please enter a string: Hello world
dlrow olleH
dyn-49-127-69-0:Lectures juliangarcia$
```

```python
        ans = ans + my_stack.pop()
    # ans contains the reversed string
    return ans


if __name__ == "__main__":
    my_string = input("Please enter a string: ")
    result = reverse_string(my_string)
    print(result)
```

Some Stacks Applications

- Undo editing
- Parsing
  - Reverse polish notation
  - Delimiter matching
- Run-time memory management
  - Stack oriented programming languages
  - Virtual machines
  - Function calling
- Implement recursion

Summary

- Stacks
- Array implementation
- Basic operations
- Their complexity