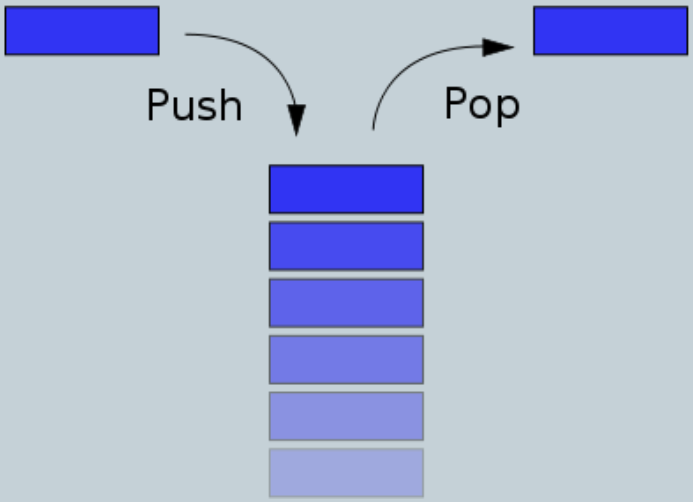
Stacks and Queues

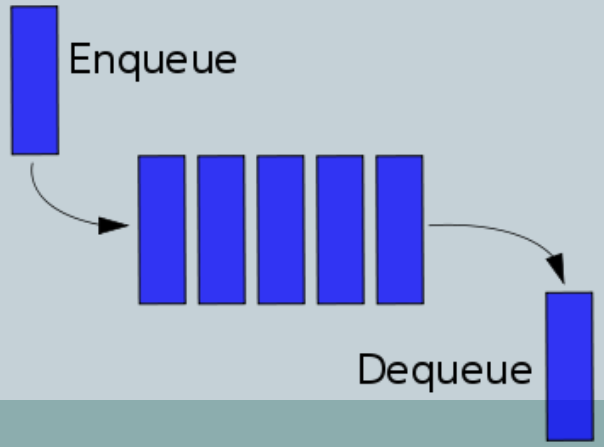
http://www.flickr.com/photos/mac-ash/4534203626/

http://www.flickr.com/photos/thowi/182298390/

Outline

- Terminology
  - Abstract Data Types (ADT)
  - Data structures
- Stack ADT
  - Last-in first-out (LIFO)
- Queue ADT
  - First-in first-out (FIFO)





ADT vs. Data Structure

- Abstract Data Type (ADT)
  - A collection of data and a set of operations on that data
  - Why is it "abstract"?
    - Doesn't specify implementation details
    - Just describes what the type can do
    - You can use without knowing internal workings
  - e.g. Stack, Queue, HashTable, List, SortedList
- Data structure
  - How the data type is implemented in software
  - e.g. array, linked list, linked graph

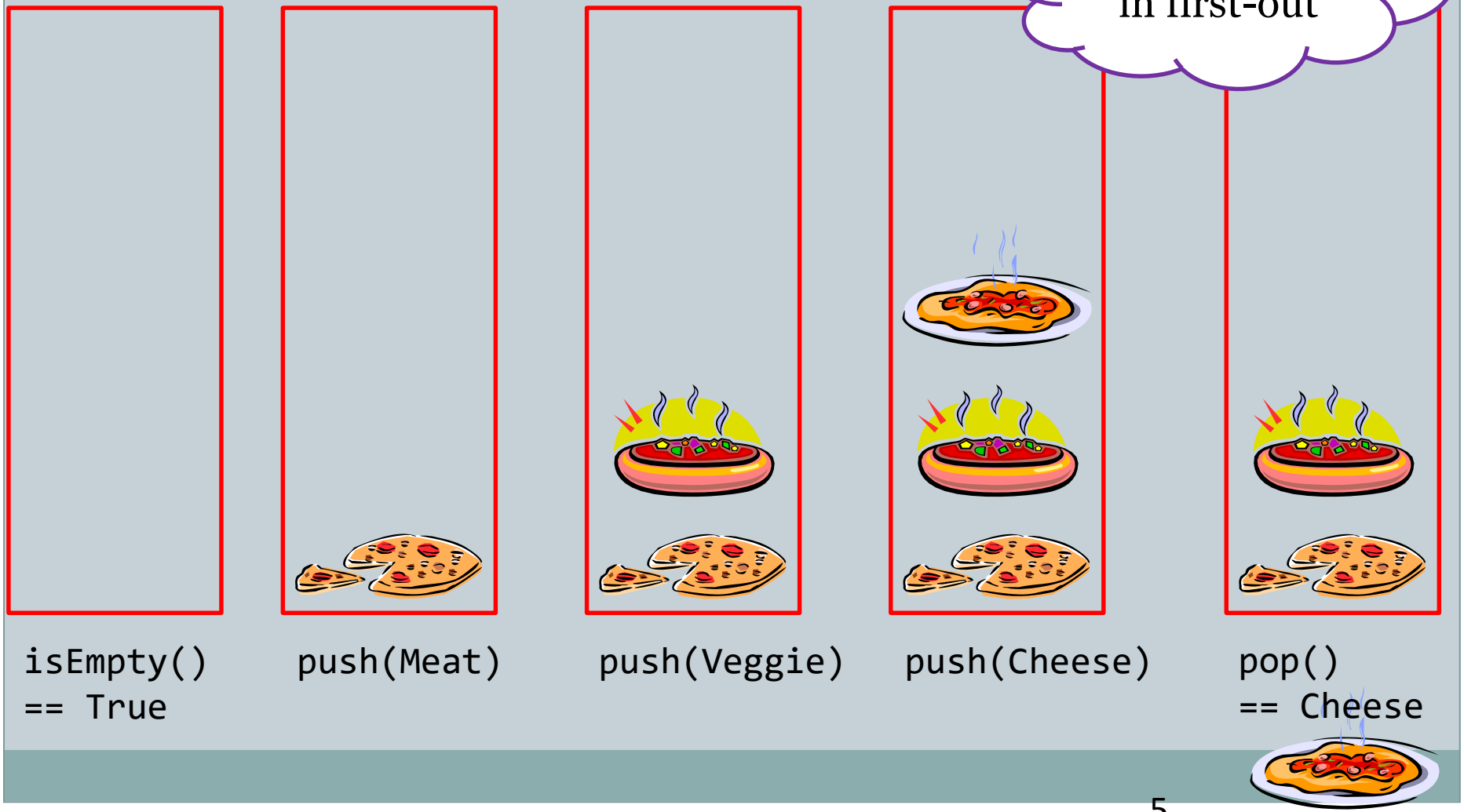
Collections

- Collection: A common data type for storing data
  - Allow users to insert item
  - Allow users to remove item, but *which one*?
  - Allow users to see if the collection is empty
- List
  - Remove at specified position
  - e.g. pile of resumes in order of GPA, Python's list
- Stack
  - Remove the most recently added = LIFO (Last-In First-Out)
  - e.g. trays in the cafeteria
- Queue
  - Remove the least recently added = FIFO (First-In First-Out)
  - e.g. line at the grocery store
- Symbol Table
  - Remove item with a given key
  - e.g. phone book: maps a name to a phone number

LIFO Stack Example

Remove most recently added

LIFO = last-in first-out



`isEmpty()`
`== True`

`push(Meat)`

`push(Veggie)`

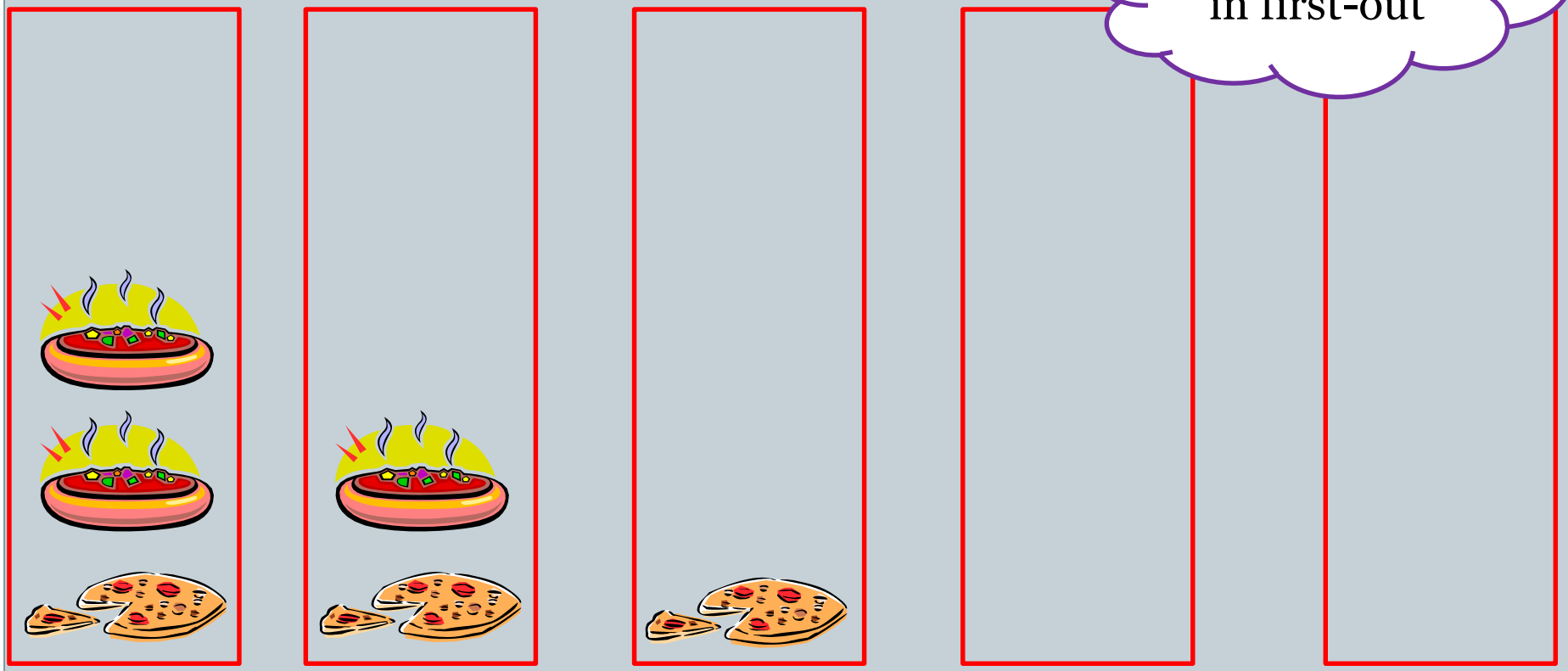
`push(Cheese)`

`pop()`
`== Cheese`

LIFO Stack Example

Remove most recently added

LIFO = last-in first-out



`push(Veggie)`

`pop()`
`== Veggie`

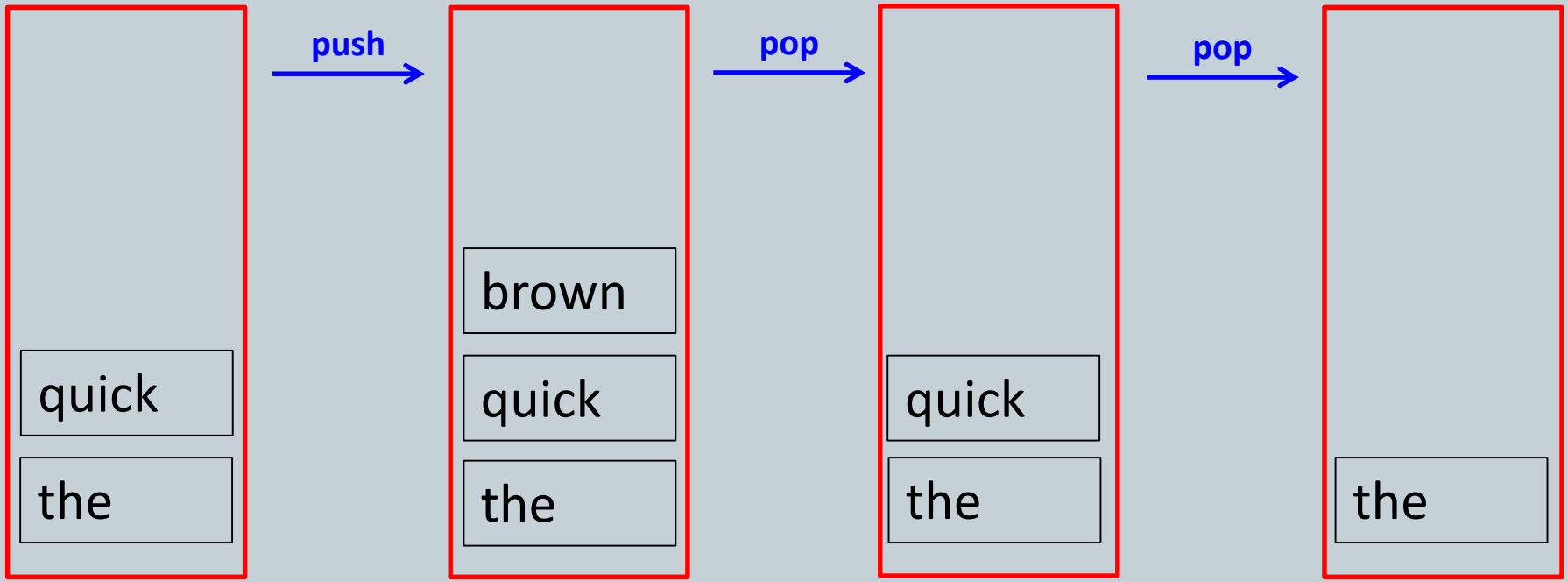
`pop()`
`== Veggie`

`pop()`
`== Meat`

`isEmpty()`
`== True`

LIFO Stack API

```
class StackOfStrings
---------------------------------------------------------------------
            __init__()        # Construct a new stack
            push(String s)    # Add a new string to the stack
     string pop()             # Remove the most recently added string
    boolean isEmpty()         # Check if the stack is empty
```



LIFO Stack Example 1

- Goal: Reverse all the words in a file
  - "glory is fleeting but obscurity is forever" →
  - "forever is obscurity but fleeting is glory"
- Approach:
  - Use a Stack ADT as implemented by `StackOfStrings`
  - While more text available from standard input:
    - Read a word, push on stack
  - While stack is not empty:
    - Pop from stack, output word

Reverse Words from Input

```
from StackOfStrings import StackOfStrings

stack = StackOfStrings()
words = input("Enter the words you would like to reverse:")
words = words.split()
for i in range(0, len(words)):
    print(i)
    stack.push(words[i])
    print("AFTER PUSH: " + stack.toString())
while not stack.isEmpty():
    print(stack.pop() + " ", end="")
print()
```

Create an instance of a stack ADT. Notice we don't specify a size, the class promises to handle any size.

Next word goes on top of the stack that contains all the previously read in words.

Start peeling off words starting with the last one pushed on top of the stack.

LIFO Stack Example 2

- Goal: Check for balanced ()'s and []'s

[((a+b)*d)+(e*f)] → balanced

[([a+b]*d)+(e*f)] → balanced

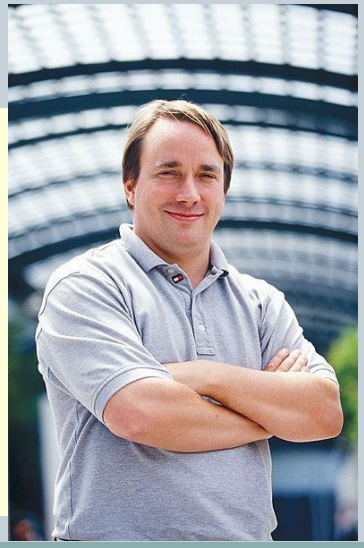
[((a+b)*d)+(e*f) → unbalanced

(a+b)*d)+(e*f) → unbalanced

[((a+b)*d)+(e*f)) → unbalanced

"I will, in fact, claim that the difference between a bad programmer and a good one is whether he considers his code or his data structures more important. **Bad programmers worry about the code. Good programmers worry about data structures** and their relationships."

-Linus Torvalds, creator of Linux



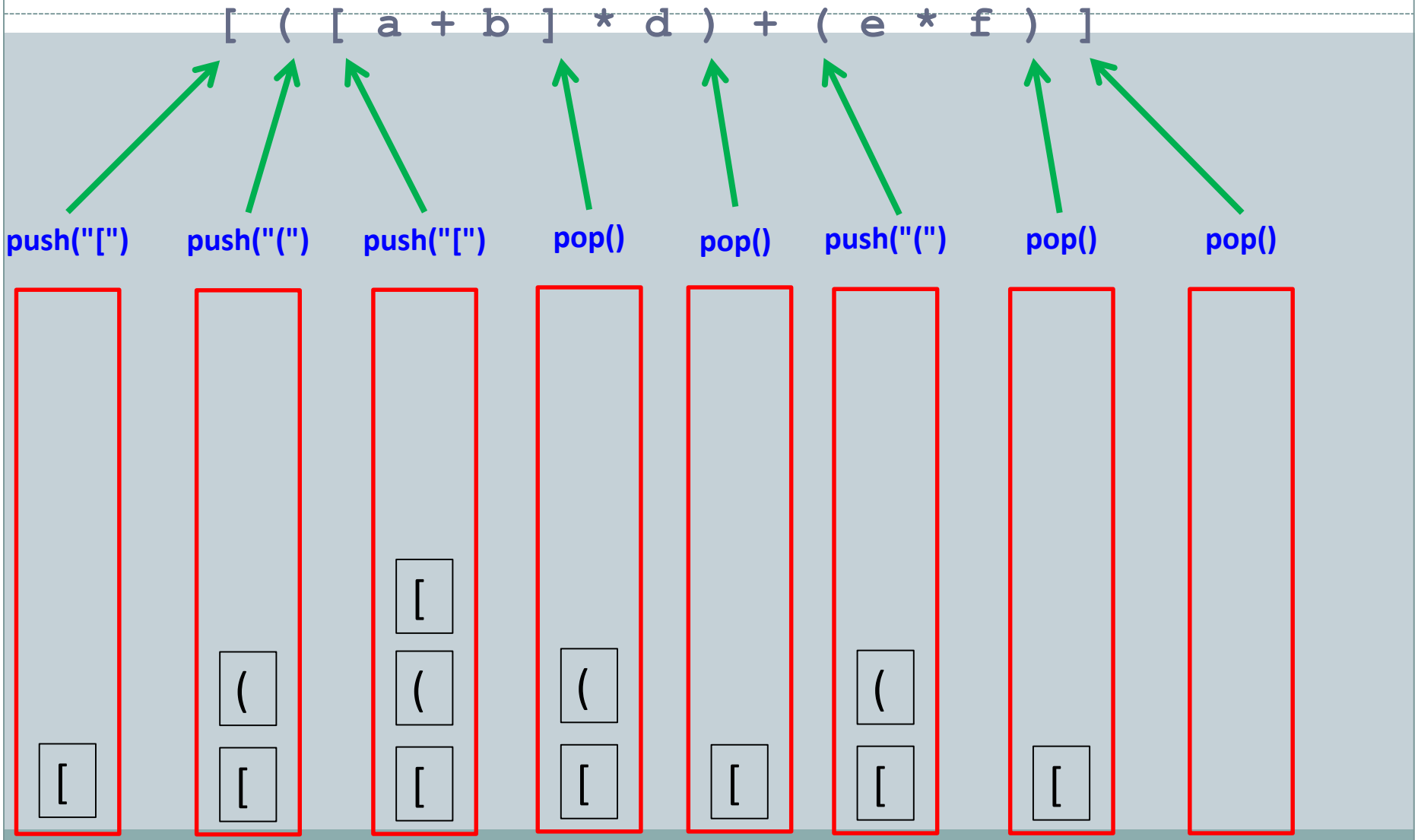
LIFO Stack Example 2

- Goal: Check for balanced ()'s and []'s

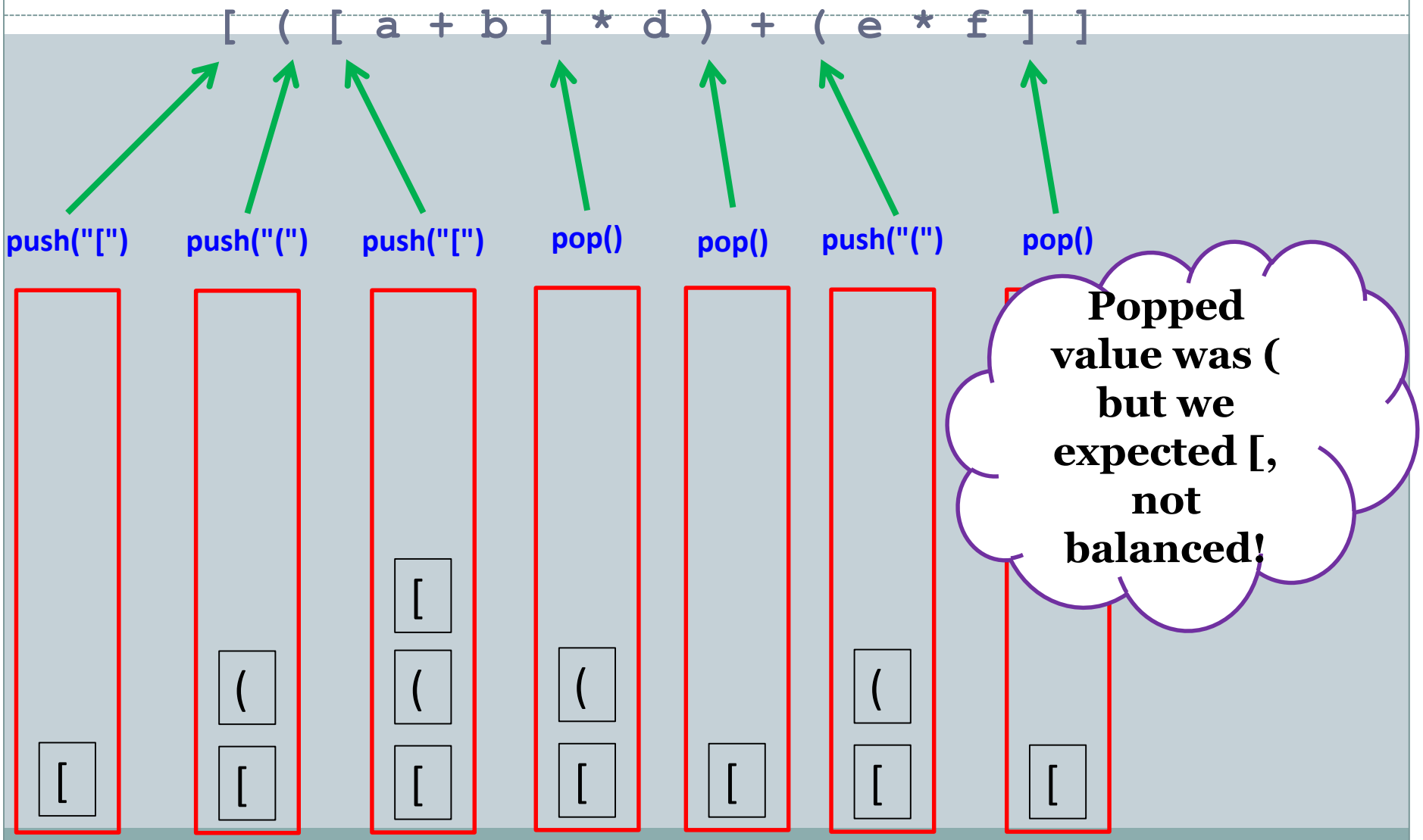
```
[((a+b)*d)+(e*f)]   → balanced
[([a+b]*d)+(e*f)]   → balanced
[((a+b)*d)+(e*f)    → unbalanced
(a+b)*d)+(e*f)      → unbalanced
[((a+b)*d)+(e*f))   → unbalanced
```

- Approach:
  - Use a Stack ADT as implemented by `StackOfStrings`
  - If token is (or [then push onto stack
  - If token is) then pop stack and make sure popped value is (
  - If token is] then pop stack and make sure popped value is [
  - Any other token, ignore

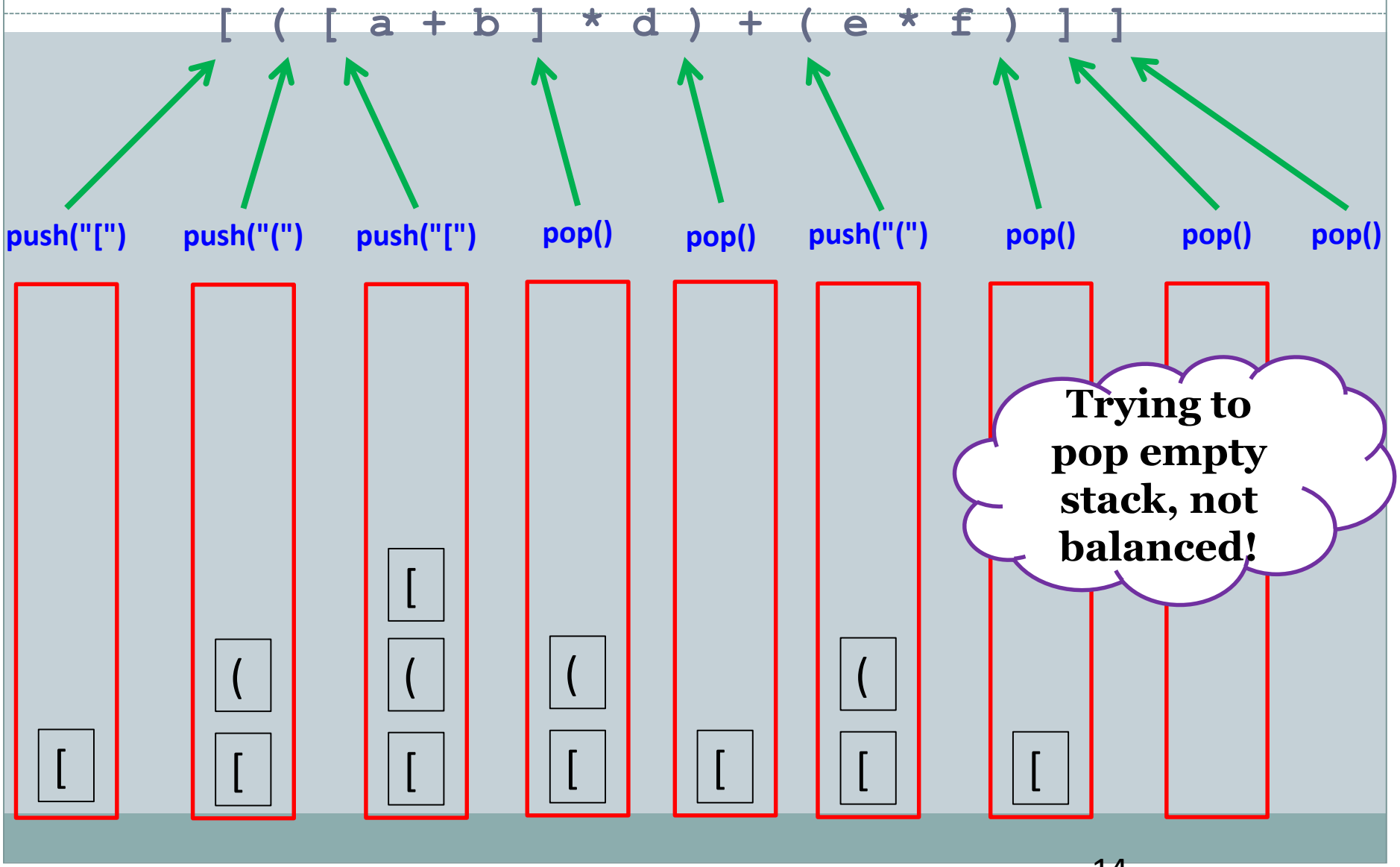
Balanced, Success



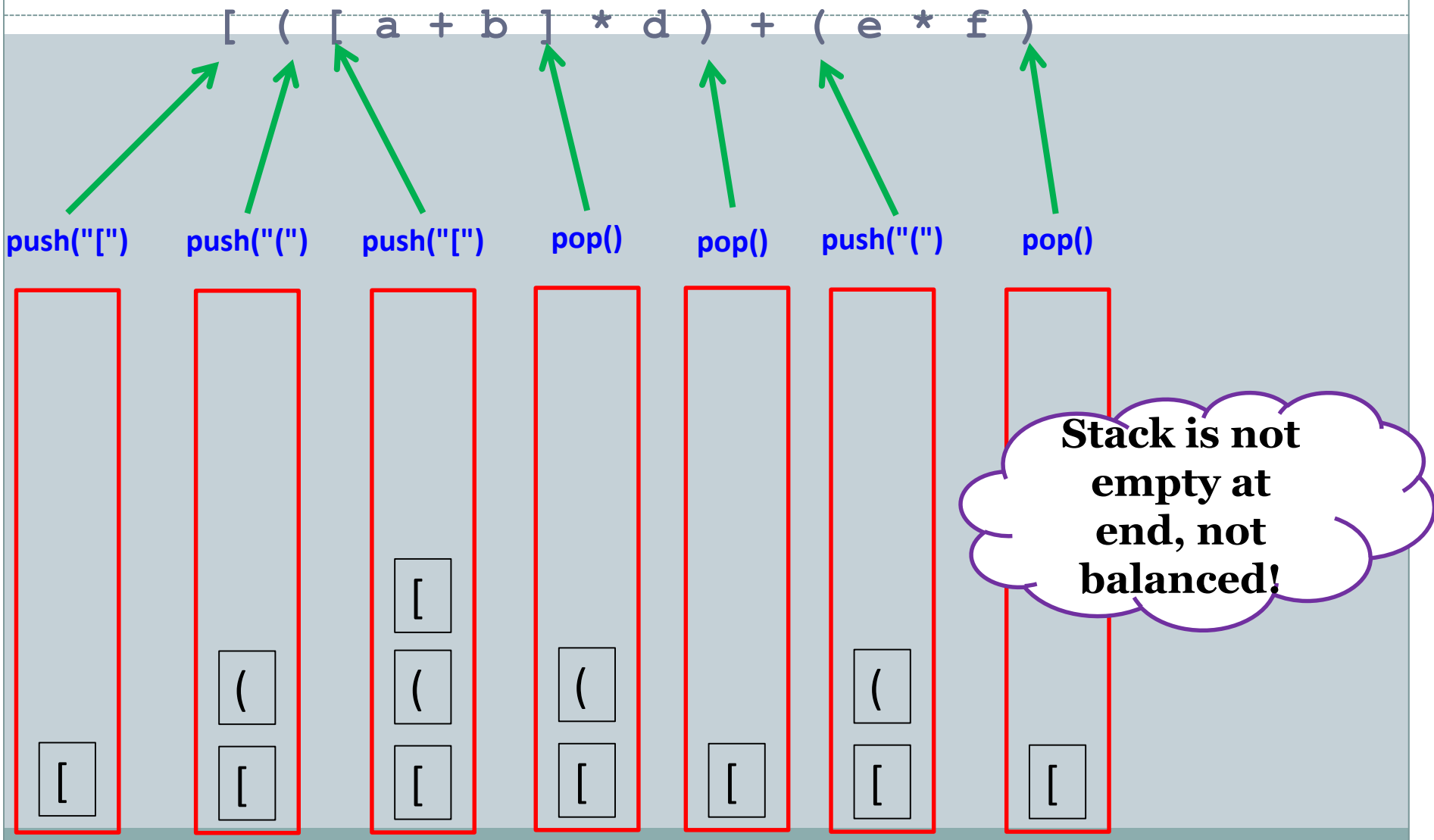
Balanced, Failure 1



Balanced, failure 2



Balanced, failure 3



```
from StackOfStrings import StackOfStrings

stack = StackOfStrings()
inputString = input("Enter the equation:")
for token in range(0,len(inputString)):
    if inputString[token] == '(' or inputString[token] == '[':
        stack.push(inputString[token])
    elif inputString[token] == ')':
        if stack.isEmpty() or stack.pop() != '(':
            exit
    elif inputString[token] == ']':
        if stack.isEmpty() or stack.pop() != '[':
            exit
if stack.isEmpty():
    print("Balanced")
else:
    print("Not balanced")
```

Balanced.py

FIFO Queue Example

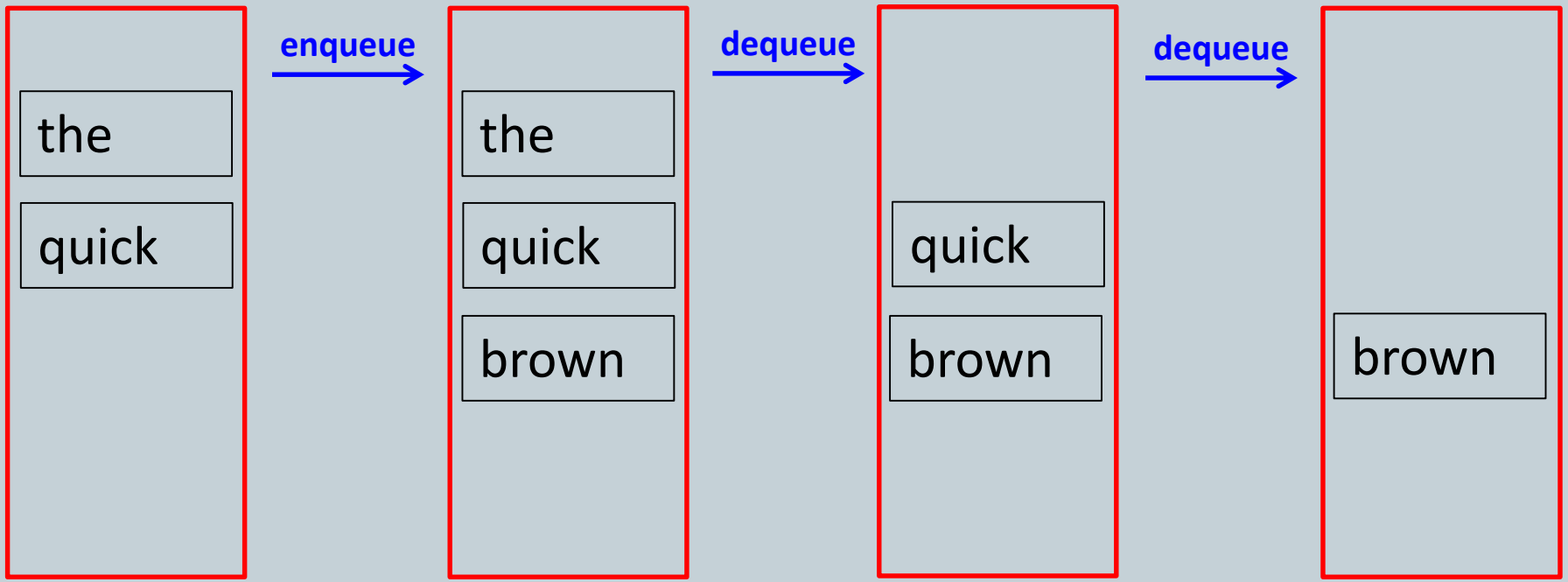
Remove least recently added

FIFO = first-in first-out

```
isEmpty() == True
enqueue(Abe)
enqueue(Bill)
enqueue(Carol)
dequeue() == Abe
enqueue(Diana)
```

FIFO Queue API

```
class QueueOfStrings
-----------------------------------------------------------------------
            __init__()          # Construct a new queue
            enqueue(string s)   # Add a new string to the queue
     string dequeue()           # Remove the least recently added string
    boolean isEmpty()           # Check if the queue is empty
```



FIFO Queue Example

- Goal: Parental spelling obfuscation aid
  - "After the kids go to sleep let's have some..."
  - Parent types "cookies" into computer
  - Computer spells out each letter, "c--o--o--k--i--e--s"
    - Pausing one second between letters
- Approach:
  - Use a Queue ADT as implemented by `QueueOfStrings`
  - Queue each new letter as it is typed
  - Delay 1s before dequeue'ing
    - Display letter
    - Play WAV audio file

```
import StdDraw
import StdAudio
from QueueOfStrings import QueueOfStrings

StdDraw.setFontFamily("Courier")
StdDraw.setFontSize(120)
delay = 0
queue = QueueOfStrings()

while True:
    # Check for a new key from a-z
    if StdDraw.hasNextKeyTyped():
        key = StdDraw.nextKeyTyped()
        if key >= 'a' and key <= 'z':
            queue.enqueue(key)
    StdDraw.show(100)
    delay += 100

    # Only update the display every second
    if delay >= 1000:
        delay = 0
        StdDraw.clear()

        # Check there is something in the queue before
        # attempting to dequeue!
        if not queue.isEmpty():
            letter = queue.dequeue()
            StdDraw.text(.5, .5, letter)
            StdAudio.playFile(letter)
```

Create an instance of a Queue data type. Notice we don't specify a size, class promises to handle any size.

Latest character goes at the back of the line. All other character have to play first.

Play the character that has been waiting the longest in the queue.

Speller.py

Summary

- Abstract Data Types (ADTs)
  - A collection of data and operations on that data
  - LIFO Stack
    - Push and pop items, always pops the last thing pushed
    - Examples: reversing words in a sentence, check for balanced parameters
  - FIFO Queue
    - Enqueue and dequeue items
    - Always dequeue the thing that has been waiting the longest
    - Examples: tracking and eventually servicing asynchronous events (keys typed by parent)
- Data structures
  - Implementation of an ADT (there may be many ways!)
  - e.g. using a normal array, using an `linked list` …

Your Turn

- Complete the StackofStringsArray.py program posted on the website. Everywhere there is a #TBD in the code, replace it with the correct code. You should use a Python list instead of the linked list implementation we looked at in class. If done correctly, the output from the test main method should look like:

```
before adding:
Pushing words: it was the best of times
times of best the was it
after popping:
```

- Submit your solution to the Moodle Activity 2 dropbox for today. You get 1 point for turning something in, another for turning in something correct.