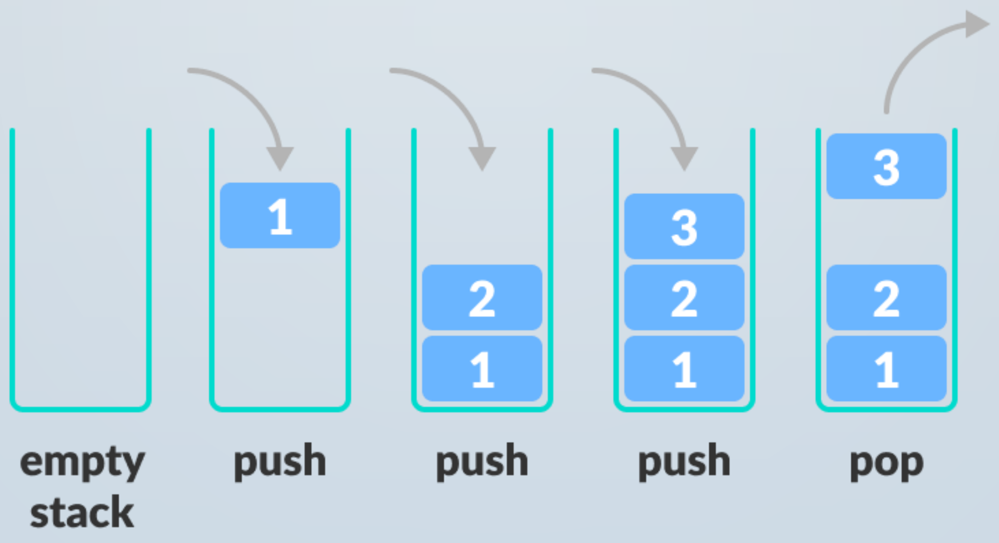
Stack

TUAN NGUYEN

Stack

A stack is a collection of objects that are inserted and removed according to the **last-in, first-out** (LIFO) principle.



Stack (1)

- A user may insert objects into a stack at any time, but may only access or remove the most recently inserted object that remains (at the so-called “top” of the stack).
- “stack” is derived from the metaphor of a stack of plates in a spring-loaded, cafeteria plate dispenser.
- Example: web browsers and back button., text editor with undo mechanism.

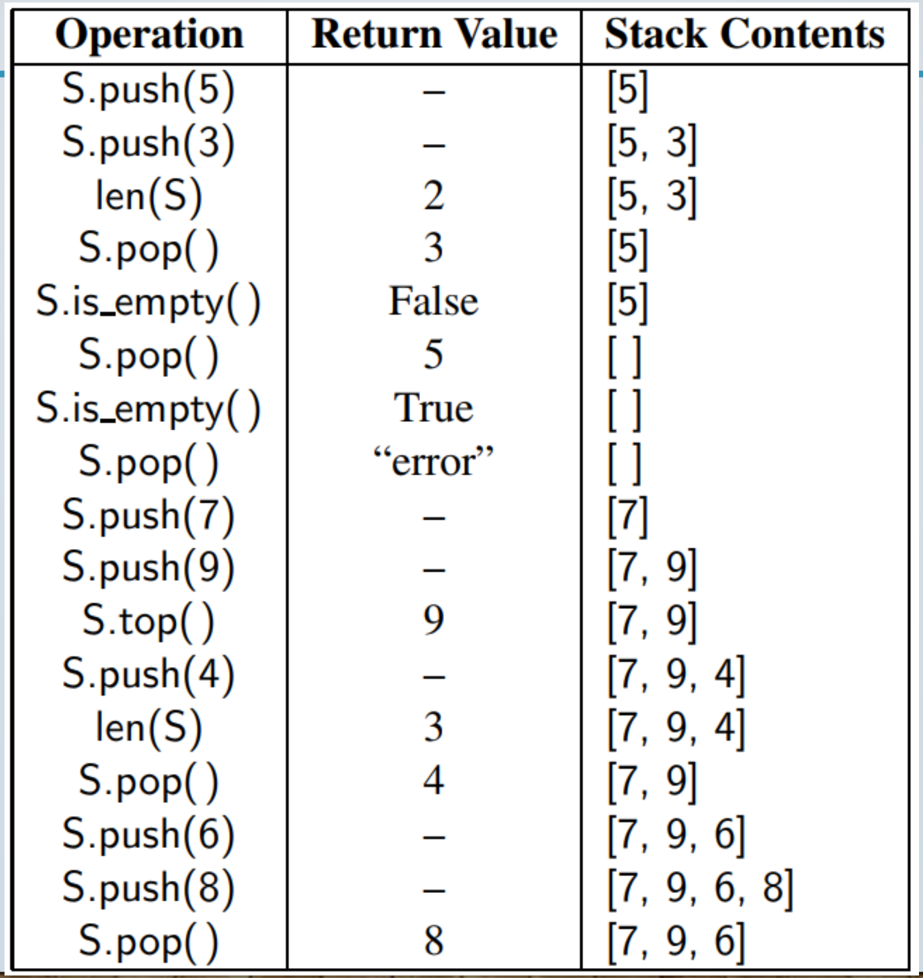
Abstract Data Type

- Important methods:
- **S.push(e)**: Add element e to the top of stack S.
- **S.pop()**: Remove and return the top element from the stack S; an error occurs if the stack is empty.

Other methods:

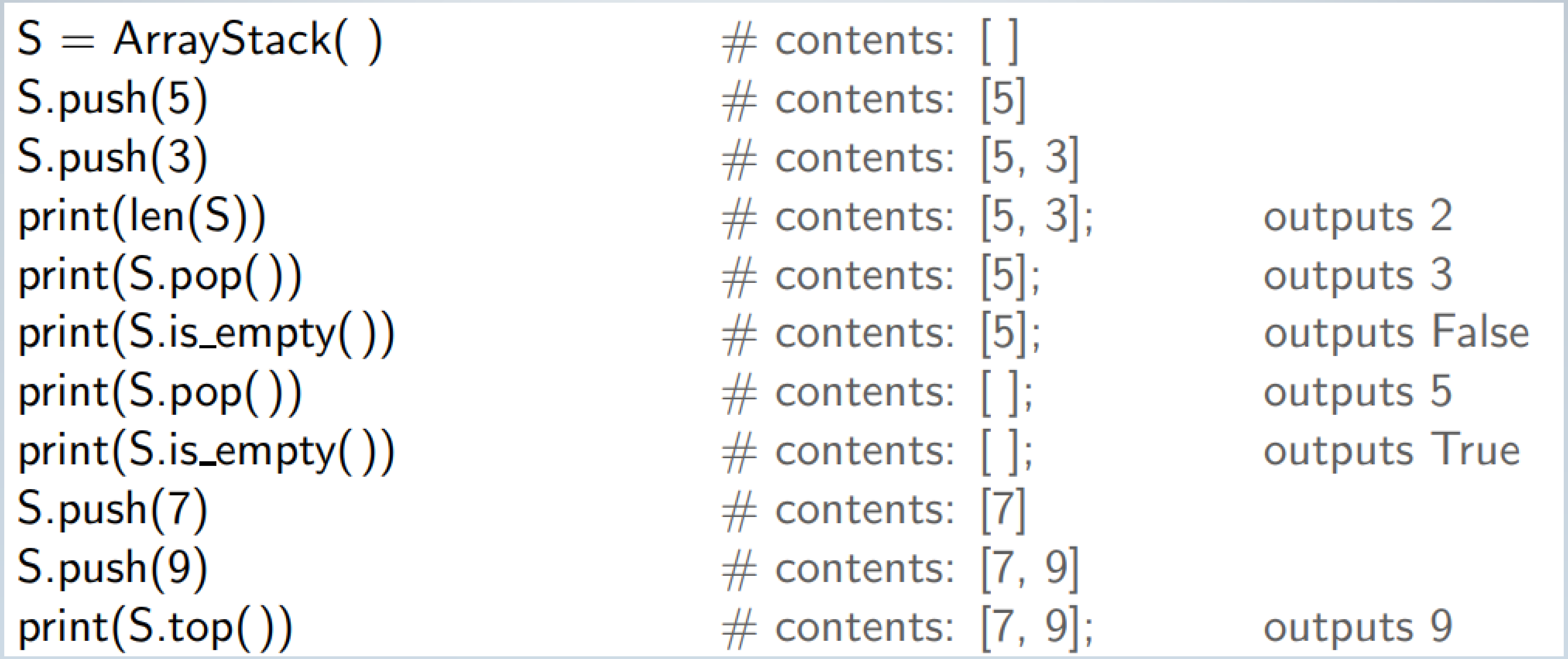
- **S.top()**: Return a reference to the top element of stack S, without removing it; an error occurs if the stack is empty.
- **S.is_empty()**: Return True if stack S does not contain any elements.
- **len(S)**: Return the number of elements in stack S; in Python, we implement this with the special method len .

Abstract Data Type (1)



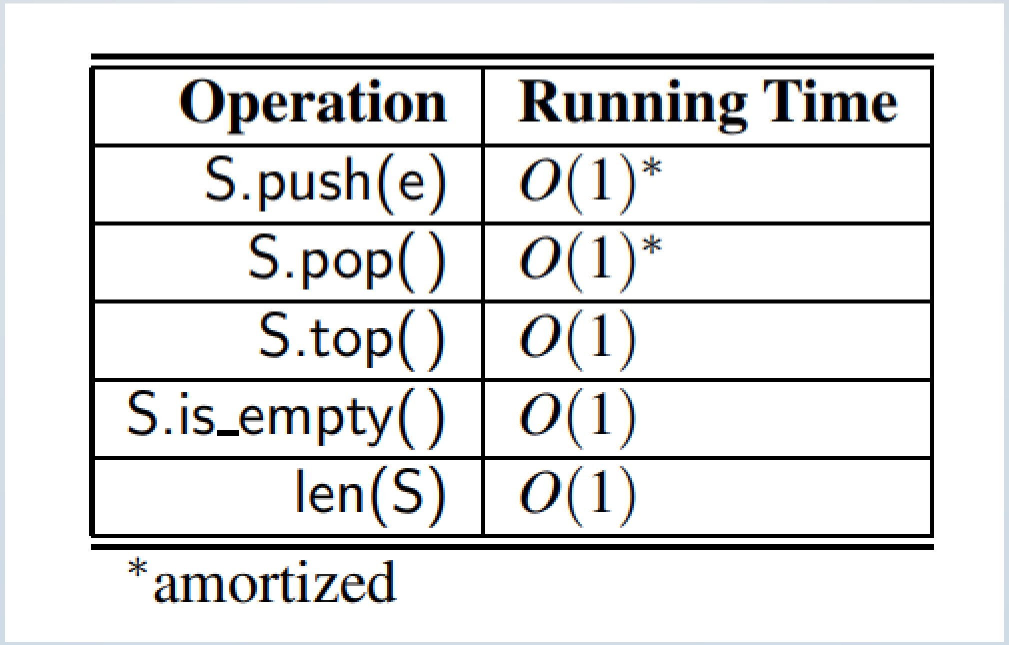
Operation	Return Value	Stack Contents
S.push(5)	–	[5]
S.push(3)	–	[5, 3]
len(S)	2	[5, 3]
S.pop()	3	[5]
S.is_empty()	False	[5]
S.pop()	5	[ ]
S.is_empty()	True	[ ]
S.pop()	"error"	[ ]
S.push(7)	–	[7]
S.push(9)	–	[7, 9]
S.top()	9	[7, 9]
S.push(4)	–	[7, 9, 4]
len(S)	3	[7, 9, 4]
S.pop()	4	[7, 9]
S.push(6)	–	[7, 9, 6]
S.push(8)	–	[7, 9, 6, 8]
S.pop()	8	[7, 9, 6]

Implement Stack Using Python List



```
S = ArrayStack( )          # contents: [ ]
S.push(5)                  # contents: [5]
S.push(3)                  # contents: [5, 3]
print(len(S))              # contents: [5, 3];     outputs 2
print(S.pop( ))            # contents: [5];        outputs 3
print(S.is_empty( ))       # contents: [5];        outputs False
print(S.pop( ))            # contents: [ ];        outputs 5
print(S.is_empty( ))       # contents: [ ];        outputs True
S.push(7)                  # contents: [7]
S.push(9)                  # contents: [7, 9]
print(S.top( ))            # contents: [7, 9];     outputs 9
```

Running Time



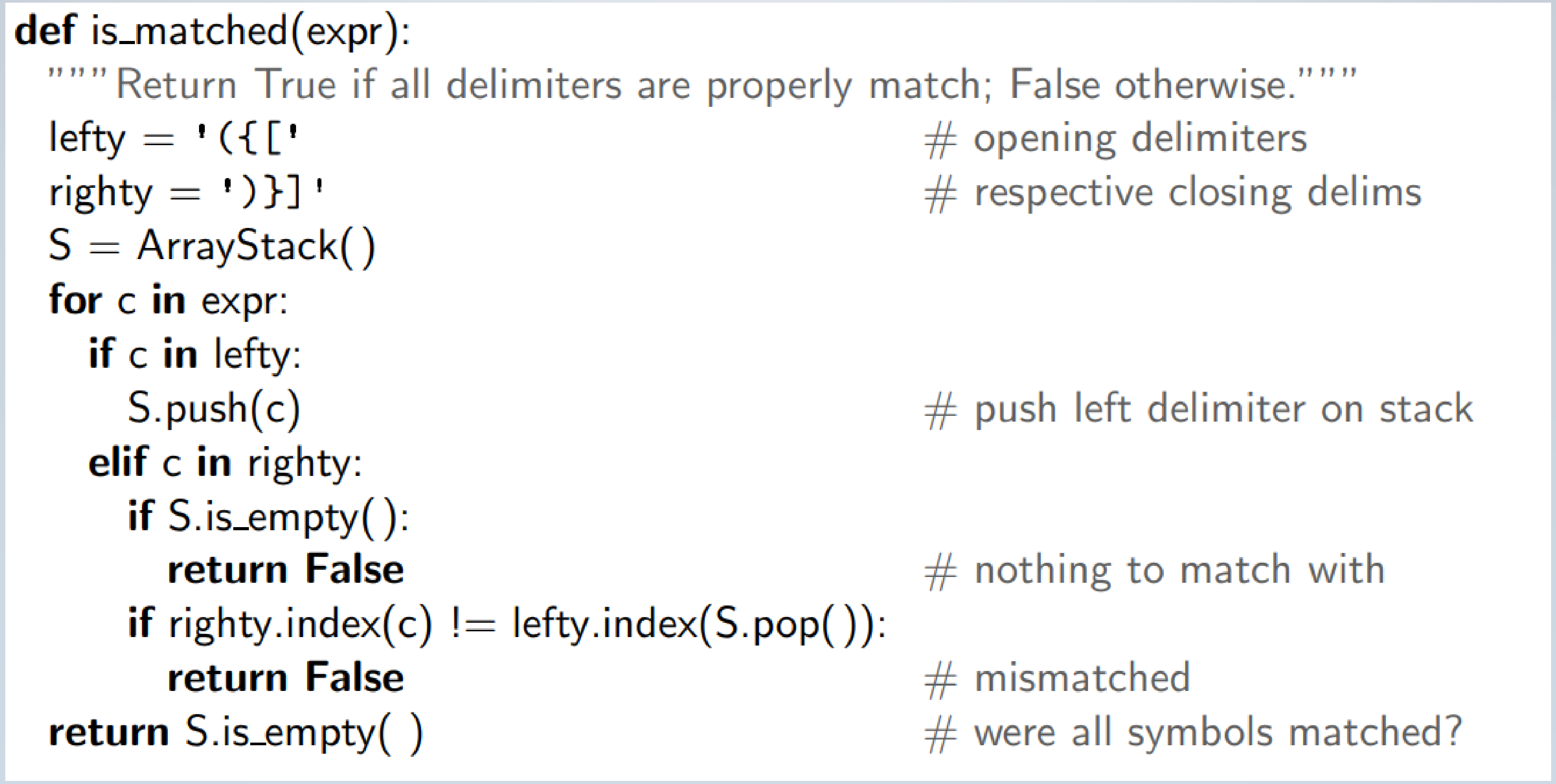
Operation	Running Time
S.push(e)	$O(1)^*$
S.pop()	$O(1)^*$
S.top()	$O(1)$
S.is_empty()	$O(1)$
len(S)	$O(1)$

*amortized

Matching Parentheses

- Pairs of grouping symbols, such as:
  - Parentheses: “(” and “)”
  - Braces: “{” and “}”
  - Brackets: “[” and “]”
- Each opening symbol must match its corresponding closing symbol. For example, a left bracket, “[,” must match a corresponding right bracket, “],” as in the expression [(5+x)-(y+z)]. The following examples further illustrate this concept:
  - Correct: ()(()){([()])}
  - Incorrect:)(()){([()])}
  - Incorrect: ({[])}
  - Incorrect: (

Matching Parentheses



```
def is_matched(expr):
  """Return True if all delimiters are properly match; False otherwise."""
  lefty = '({['                                  # opening delimiters
  righty = ')}]'                                 # respective closing delims
  S = ArrayStack()
  for c in expr:
    if c in lefty:
      S.push(c)                                  # push left delimiter on stack
    elif c in righty:
      if S.is_empty():
        return False                             # nothing to match with
      if righty.index(c) != lefty.index(S.pop()):
        return False                             # mismatched
  return S.is_empty()                            # were all symbols matched?
```