CSI33 Data Structures

Department of Mathematics and Computer Science
Bronx Community College

September 27, 2017

Outline

1 Chapter 5: Stacks and Queues
- Stacks

Outline

1. Chapter 5: Stacks and Queues
   - Stacks

The Stack ADT

A Container Class for Last-In-First-Out Access

A stack is a list-like container with access restricted to one end of the list (the top of the stack). You can

- push an item onto the stack
- pop an item off the stack (precondition: stack is not empty—`size > 0`)
- Inspect the top position (precondition: stack is not empty—`size > 0`)
- Obtain the current size of the stack.

Simple Stack Applications

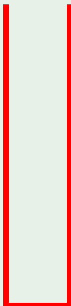
Balanced Parentheses

```
def parensBalance2(s):
   stack = Stack()
   for ch in s:
      if ch in "([{":  " push an opening marker "
         stack.push(ch)
      elif ch in ")]}":  " match closing "
         if stack.size() < 1:  " no pending open "
            return False
         else:
            opener = stack.pop()
            if opener+ch not in ["()", "[]", "{}"]:
               return False " not a matching pair"
   return stack.size() == 0 " everything matched?"
```

Simple Stack Applications

Balanced Parentheses

{[2 * (7 - 4) + 2] + 3} * 4



Simple Stack Applications

Balanced Parentheses

{[2 * (7 - 4) + 2] + 3} * 4

{

Simple Stack Applications

Balanced Parentheses

{[2 * (7 - 4) + 2] + 3} * 4

[
{

Simple Stack Applications

Balanced Parentheses

{[2 * (7 - 4) + 2] + 3} * 4

[
{

Simple Stack Applications

Balanced Parentheses

{[2 * (7 - 4) + 2] + 3} * 4

[
{

Simple Stack Applications

Balanced Parentheses

{[2 * (7 - 4) + 2] + 3} * 4

(
[
{

Simple Stack Applications

Balanced Parentheses

{[2 * (7 - 4) + 2] + 3} * 4

(
[
{

Simple Stack Applications

Balanced Parentheses

{[2 * (7 - 4) + 2] + 3} * 4

(
[
{

Simple Stack Applications

Balanced Parentheses

{[2 * (7 - 4) + 2] + 3} * 4

(
[
{

Simple Stack Applications

Balanced Parentheses

{[2 * (7 - 4) + 2] + 3} * 4

(
[
{

Simple Stack Applications

Balanced Parentheses

{[2 * (7 - 4) + 2] + 3} * 4

[
{

Simple Stack Applications

Balanced Parentheses

{[2 * (7 - 4) + 2] + 3} * 4

[
{

Simple Stack Applications

Balanced Parentheses

{[2 * (7 - 4) + 2] + 3} * 4

[
{

Simple Stack Applications

Balanced Parentheses

{[2 * (7 - 4) + 2] + 3} * 4

Simple Stack Applications

Balanced Parentheses

{[2 * (7 - 4) + 2] + 3} * 4

{

Simple Stack Applications

Balanced Parentheses

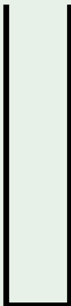
{[2 * (7 - 4) + 2] + 3} * 4

{

Simple Stack Applications

Balanced Parentheses

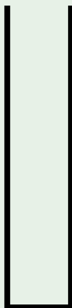
{[2 * (7 - 4) + 2] + 3} * 4



Simple Stack Applications

Balanced Parentheses

{[2 * (7 - 4) + 2] + 3} * 4



Implementing Stacks

A Python List As Concrete Representation

In Python the natural implementation of a stack is with a list.

```
class Stack(object):
  def __init__(self):
    self.items = []
  def push(self, item):
    self.items.append(item)
  def pop(self):
    return self.items.pop()
  def top(self):
    return self.items[-1]
  def size(self):
    return len(self.items)
```

An Application: Expression Manipulation

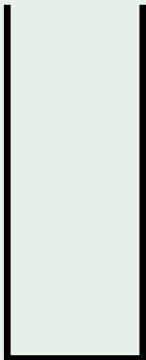
Notation For Operations

- infix: `(2 + 3) * 4`
- prefix: `* + 2 3 4`
- postfix: `2 3 + 4 *`

An Application: Expression Manipulation

Evaluating A Postfix Expression

3 4 5 + * 2 - 3 6 * +



An Application: Expression Manipulation

Evaluating a Postfix Expression

3 4 5 + * 2 - 3 6 * +

3

An Application: Expression Manipulation

Evaluating a Postfix Expression

3 4 5 + * 2 - 3 6 * +

4
3

An Application: Expression Manipulation

Evaluating a Postfix Expression

3 4 5 + * 2 - 3 6 * +

Stack
5
4
3

An Application: Expression Manipulation

Evaluating a Postfix Expression

3 4 5 + * 2 - 3 6 * +

Stack
9
3

An Application: Expression Manipulation

Evaluating A Postfix Expression

3 4 5 + * 2 - 3 6 * +

27

An Application: Expression Manipulation

Evaluating a Postfix Expression

3 4 5 + * 2 - 3 6 * +

Stack
2
27

An Application: Expression Manipulation

Evaluating A Postfix Expression

3 4 5 + * 2 - 3 6 * +

25

An Application: Expression Manipulation

Evaluating a Postfix Expression

3 4 5 + * 2 - 3 6 * +

Stack
3
25

An Application: Expression Manipulation

Evaluating a Postfix Expression

3 4 5 + * 2 - 3 6 * +

6
3
25

An Application: Expression Manipulation

Evaluating a Postfix Expression

3 4 5 + * 2 - 3 6 * +

18
25

An Application: Expression Manipulation

Evaluating a Postfix Expression

3 4 5 + * 2 - 3 6 * +

43

The Call Stack

Function Calls Can Be Nested

- function A calls function B
- function B returns
- function A continues

The Call Stack

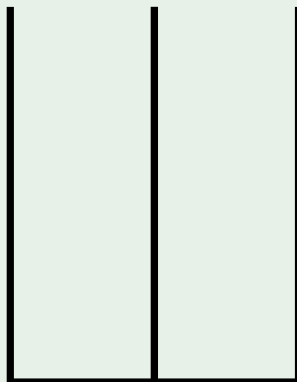
Activation Records

- Function A is running, and calls function B.
- The local variables of function A, their current values, and where function B should return to are put into an activation record.
- The activation record is pushed onto the call stack which has been allocated for the program that is running.
- When function B returns, this record is popped off the call stack and used to continue running the program.

The Call Stack

Example

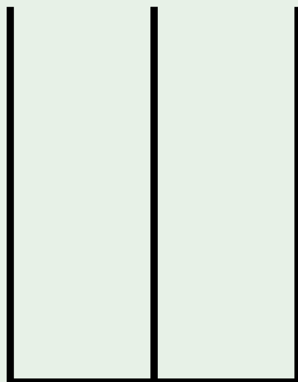
```
def A(x, y):
   1:   x2 = B(x)
   2:   y2 = B(y)
   3:   z = x2 + y2
   4:   return z
def B(n):  'squares n '
   5:   n2 = n * n
   6:   return n2
def main():
   7:   a = 3
   8:   b = 4
   9:   c = A(a, b)
   10:   print(c)
   11:   return
```



The Call Stack

Example

```
def A(x, y):
   1:   x2 = B(x)
   2:   y2 = B(y)
   3:   z = x2 + y2
   4:   return z
def B(n):  'squares n '
   5:   n2 = n * n
   6:   return n2
def main():
   7:   a = 3
   8:   b = 4
   9:   c = A(a, b)
   10:   print(c)
   11:   return
```

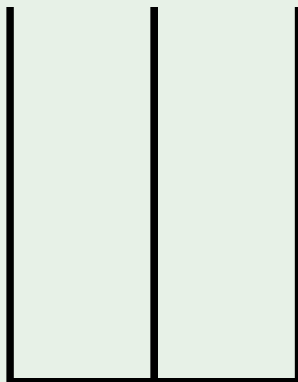


a = 3

The Call Stack

Example

```
def A(x, y):
   1:   x2 = B(x)
   2:   y2 = B(y)
   3:   z = x2 + y2
   4:   return z
def B(n):  'squares n '
   5:   n2 = n * n
   6:   return n2
def main():
   7:   a = 3
   8:   b = 4
   9:   c = A(a, b)
   10:   print(c)
   11:   return
```

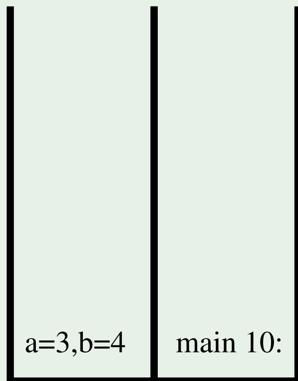


a = 3, b = 4

The Call Stack

Example

```
def A(x, y):
   1:   x2 = B(x)
   2:   y2 = B(y)
   3:   z = x2 + y2
   4:   return z
def B(n):  'squares n '
   5:   n2 = n * n
   6:   return n2
def main():
   7:   a = 3
   8:   b = 4
   9:   c = A(a, b)
   10:   print(c)
   11:   return
```

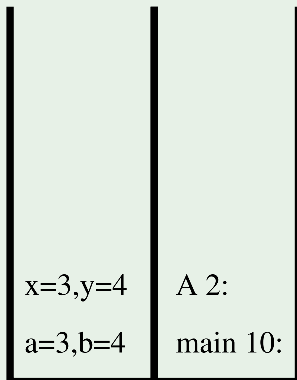


x = 3, y = 4

The Call Stack

Example

```
def A(x, y):
   1:   x2 = B(x)
   2:   y2 = B(y)
   3:   z = x2 + y2
   4:   return z
def B(n):  'squares n '
   5:   n2 = n * n
   6:   return n2
def main():
   7:   a = 3
   8:   b = 4
   9:   c = A(a, b)
   10:   print(c)
   11:   return
```

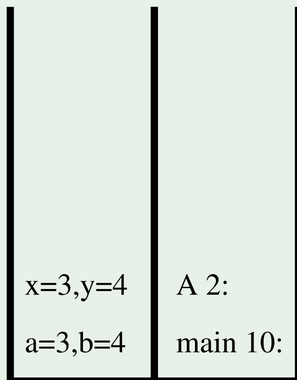


n = 3

The Call Stack

Example

```
def A(x, y):
   1:   x2 = B(x)
   2:   y2 = B(y)
   3:   z = x2 + y2
   4:   return z
def B(n):  'squares n '
   5:   n2 = n * n
   6:   return n2
def main():
   7:   a = 3
   8:   b = 4
   9:   c = A(a, b)
   10:   print(c)
   11:   return
```

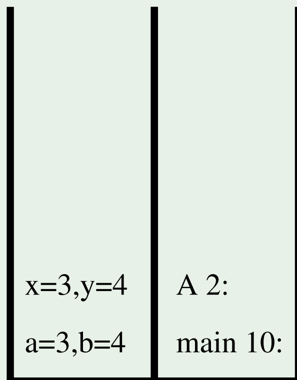


n = 3, n2 = 9

The Call Stack

Example

```
def A(x, y):
  1:   x2 = B(x)
  2:   y2 = B(y)
  3:   z = x2 + y2
  4:   return z
def B(n):  'squares n '
  5:   n2 = n * n
  6:   return n2
def main():
  7:   a = 3
  8:   b = 4
  9:   c = A(a, b)
  10:   print(c)
  11:   return
```

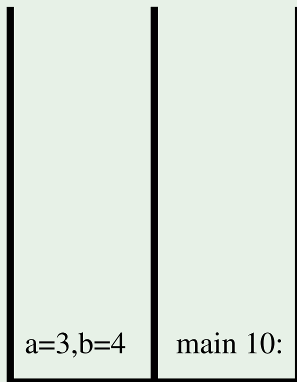


n = 3, n2 = 9

The Call Stack

Example

```
def A(x, y):
   1:   x2 = B(x)
   2:   y2 = B(y)
   3:   z = x2 + y2
   4:   return z
def B(n):  'squares n '
   5:   n2 = n * n
   6:   return n2
def main():
   7:   a = 3
   8:   b = 4
   9:   c = A(a, b)
   10:   print(c)
   11:   return
```

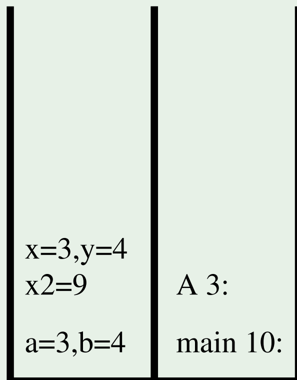


x = 3, y = 4, x2 = 9

The Call Stack

Example

```
def A(x, y):
   1:   x2 = B(x)
   2:   y2 = B(y)
   3:   z = x2 + y2
   4:   return z
def B(n):  'squares n '
   5:   n2 = n * n
   6:   return n2
def main():
   7:   a = 3
   8:   b = 4
   9:   c = A(a, b)
   10:   print(c)
   11:   return
```

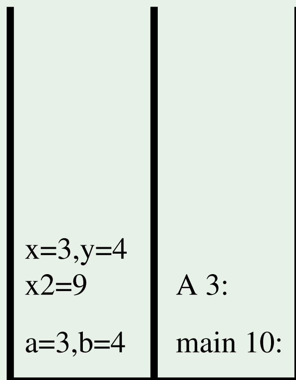


n = 4

The Call Stack

Example

```
def A(x, y):
   1:   x2 = B(x)
   2:   y2 = B(y)
   3:   z = x2 + y2
   4:   return z
def B(n):  'squares n '
   5:   n2 = n * n
   6:   return n2
def main():
   7:   a = 3
   8:   b = 4
   9:   c = A(a, b)
   10:   print(c)
   11:   return
```

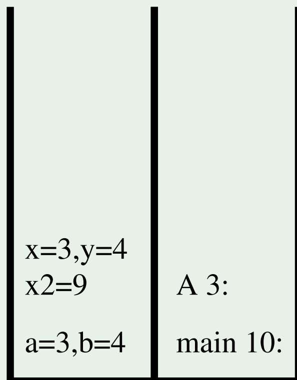


n = 4, n2 = 16

The Call Stack

Example

```
def A(x, y):
   1:   x2 = B(x)
   2:   y2 = B(y)
   3:   z = x2 + y2
   4:   return z
def B(n):  'squares n '
   5:   n2 = n * n
   6:   return n2
def main():
   7:   a = 3
   8:   b = 4
   9:   c = A(a, b)
   10:   print(c)
   11:   return
```

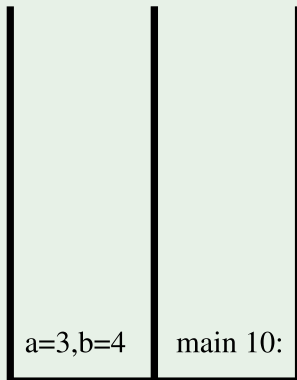


n = 4, n2 = 16

The Call Stack

Example

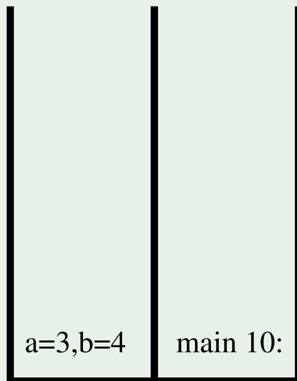
```
def A(x, y):
   1:   x2 = B(x)
   2:   y2 = B(y)
   3:   z = x2 + y2
   4:   return z
def B(n):  'squares n '
   5:   n2 = n * n
   6:   return n2
def main():
   7:   a = 3
   8:   b = 4
   9:   c = A(a, b)
   10:   print(c)
   11:   return
```



The Call Stack

Example

```
def A(x, y):
   1:   x2 = B(x)
   2:   y2 = B(y)
   3:   z = x2 + y2
   4:   return z
def B(n):  'squares n '
   5:   n2 = n * n
   6:   return n2
def main():
   7:   a = 3
   8:   b = 4
   9:   c = A(a, b)
   10:   print(c)
   11:   return
```

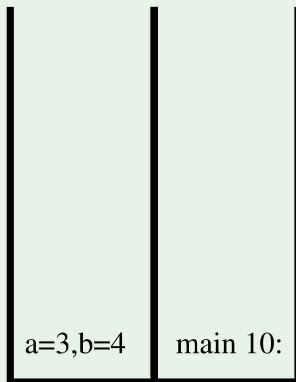


x=3,y=4,x2=9,y2=16,z=25

The Call Stack

Example

```
def A(x, y):
   1:   x2 = B(x)
   2:   y2 = B(y)
   3:   z = x2 + y2
   4:   return z
def B(n):  'squares n '
   5:   n2 = n * n
   6:   return n2
def main():
   7:   a = 3
   8:   b = 4
   9:   c = A(a, b)
   10:   print(c)
   11:   return
```



locals return

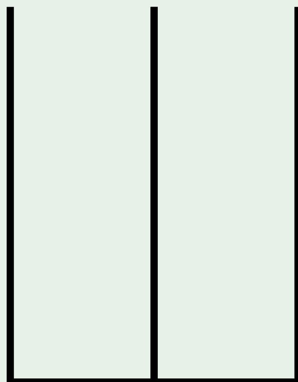
Call Stack

x=3, y=4, x2=9, y2=16, z=25

The Call Stack

Example

```
def A(x, y):
   1:   x2 = B(x)
   2:   y2 = B(y)
   3:   z = x2 + y2
   4:   return z
def B(n):  'squares n '
   5:   n2 = n * n
   6:   return n2
def main():
   7:   a = 3
   8:   b = 4
   9:   c = A(a, b)
   10:   print(c)
   11:   return
```

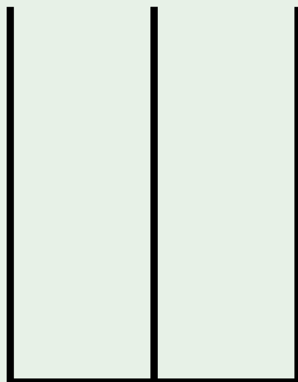


a = 3, b = 4, c = 25

The Call Stack

Example

```
def A(x, y):
   1:   x2 = B(x)
   2:   y2 = B(y)
   3:   z = x2 + y2
   4:   return z
def B(n):  'squares n '
   5:   n2 = n * n
   6:   return n2
def main():
   7:   a = 3
   8:   b = 4
   9:   c = A(a, b)
   10:   print(c)
   11:   return
```

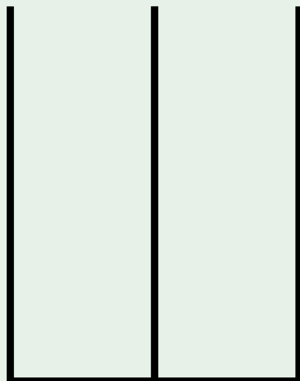


a = 3, b = 4, c = 25

The Call Stack

Example

```
def A(x, y):
   1:   x2 = B(x)
   2:   y2 = B(y)
   3:   z = x2 + y2
   4:   return z
def B(n):  'squares n '
   5:   n2 = n * n
   6:   return n2
def main():
   7:   a = 3
   8:   b = 4
   9:   c = A(a, b)
   10:   print(c)
   11:   return
```



a = 3, b = 4, c = 25