Higher-Order Functions

CSCI 1030U - Intro to Computer Science
@IntroCS

Randy J. Fortier
@randy_fortier

OntarioTech
UNIVERSITY

Outline

- Stacks and the calling stack
- Higher-order functions
    - Passing functions as arguments to other functions

Stacks and the Calling Stack

Stacks

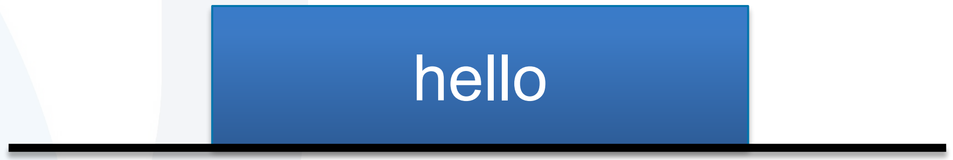
- An important data structure in computer science is the stack
- A stack is a collection of items
  - Items can only be inserted at the top of the stack
  - Items can only be removed from the top of the stack
  - Thus, a stack is a LIFO (last in, first out)
- To visualize, think of a stack of books
  - You cannot add a book to the bottom of the stack
  - Removing a book from the bottom of the stack will cause a collapse

Stacks - Push (Insertion)

- Here is an empty stack
- Let's test insertion (called a *push* in stack terminology)
- Let's push the string 'hello'

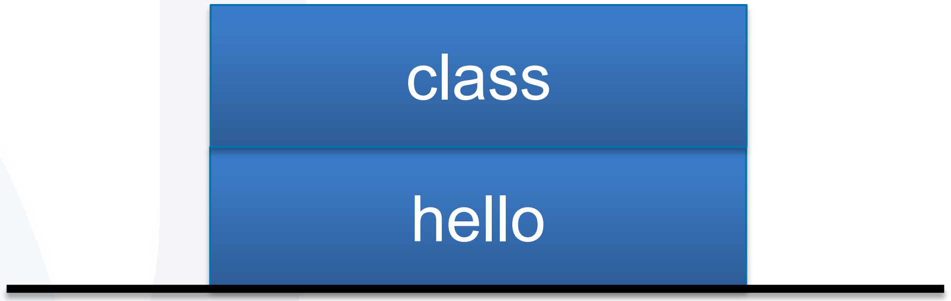
Stacks - Push (Insertion)

- Now, let's push the string 'class'

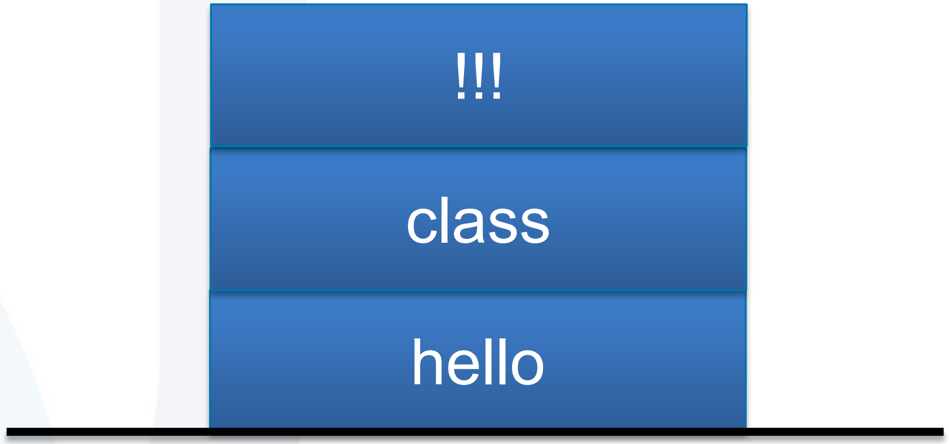


Stacks - Push (Insertion)

- Now, let's push the string '!!!'

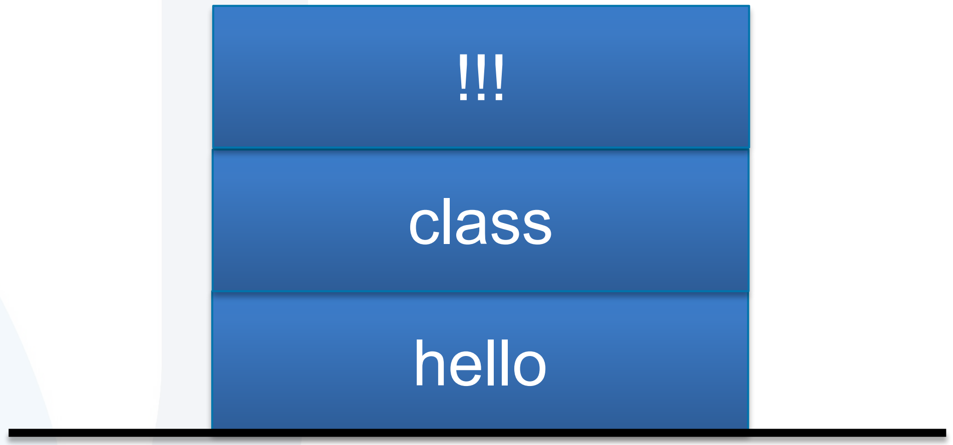


Stacks - Push (Insertion)



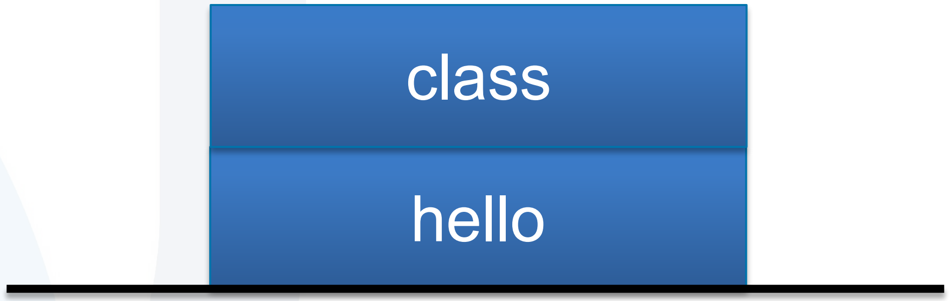
Stacks - Pop (Deletion)

- Now, let's delete (called *pop*) the top item
  - Which item pops first?



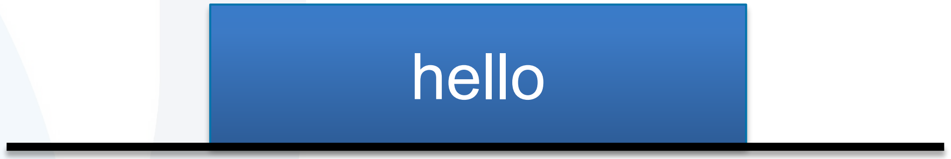
Stacks - Pop (Deletion)

- Let's delete (called *pop*) the top item again



Stacks - Pop (Deletion)

- Let's delete (called *pop*) the top item again



Stacks - Pop (Deletion)

- We are back to an empty stack
  - Notice that the order of items being popped was opposite to the order they were pushed

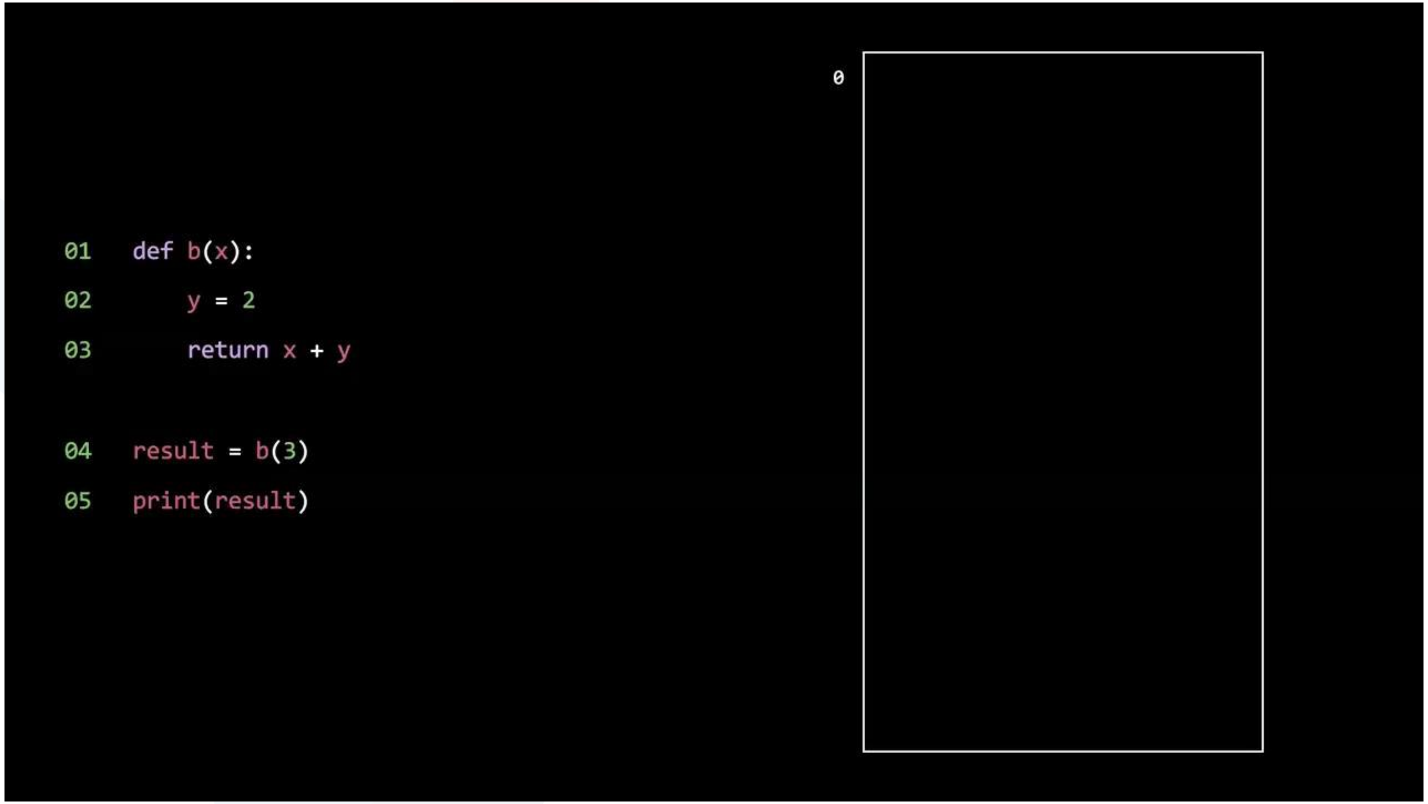
Calling Stack

- Stacks are used when functions are called
  - Each time a function is called, Python pushes a new item (called a stack frame) onto the calling stack
    - This is the way most other programming languages work
  - A stack frame contains space for all arguments and local variables

Hacker's Corner: Viewing the Stack

- When using the debugger to execute a program, you can view the calling stack

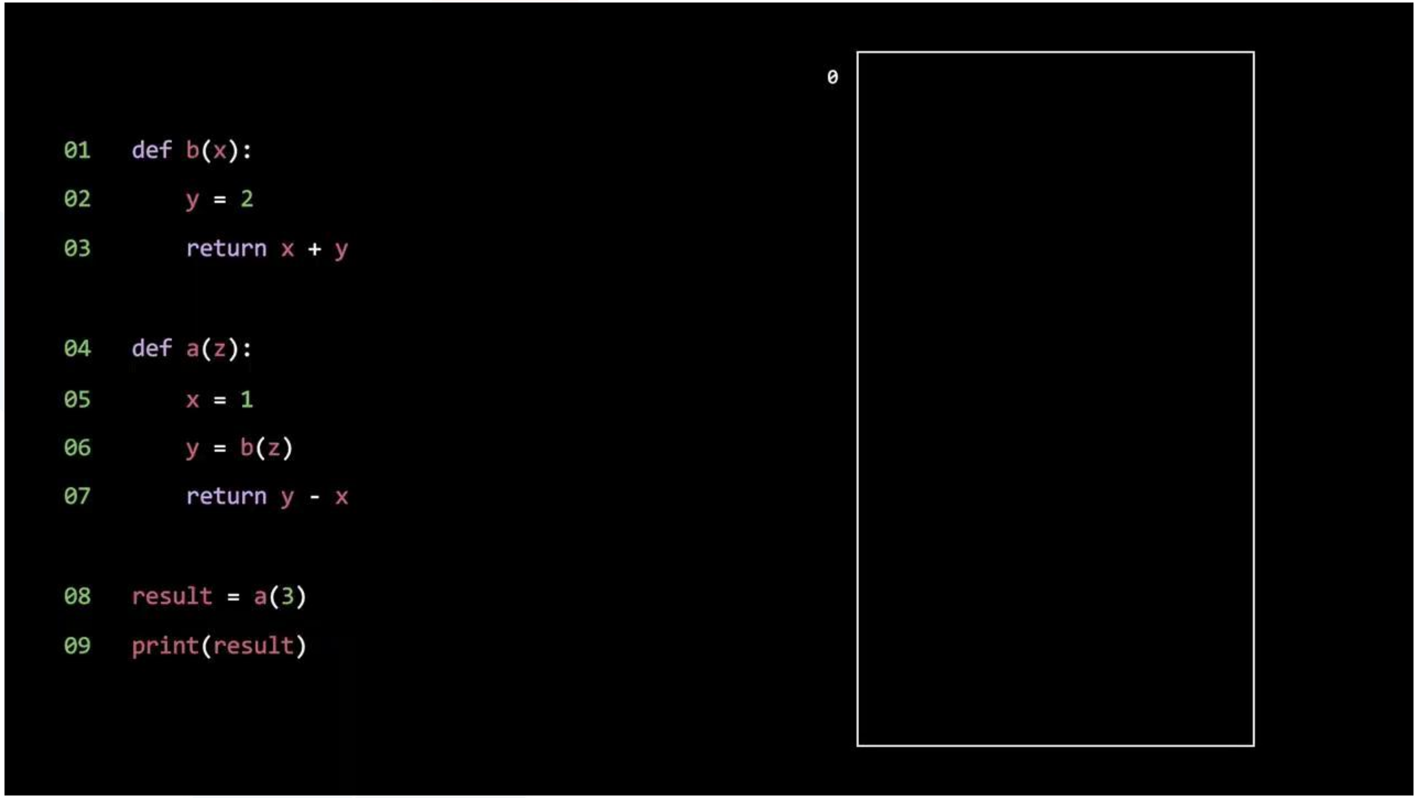
Function Calling - Video Example



```
def b(x):
    y = 2
    return x + y

result = b(3)
print(result)
```

Function Calling - Video Example



```python
def b(x):
    y = 2
    return x + y

def a(z):
    x = 1
    y = b(z)
    return y - x

result = a(3)
print(result)
```

Higher-Order Functions

Higher Order Functions

- A higher-order function is a function that can take functions as its arguments
  - In many programming languages, functions are values just like any other
  - This is handy when the structure of an algorithm is the same (e.g. navigating a tree), but some part of the operation is unique (e.g. what to do with the nodes)

Higher Order Functions - Python

- An example:

```
def traverse(elements, op):
   for element in elements:
      op(element)

def output(x):
   print(x)

traverse([1,2,3], output)
```

The `map` Function

- `map` is a function that applies (*maps*) a given function to all of the elements of a list, creating a new list from the results

```
def ftoc(f):
    return (f - 32) * 5 / 9

f_temps = [60.0, 70.0, 80.0, 90.0, 100.0]
c_temps = map(ftoc, f_temps)
```

The `map` Function

- Here is the same functionality, but using a lambda expression:

```
f_temps = [60.0, 70.0, 80.0, 90.0, 100.0]
c_temps = map(lambda f: (f - 32) * 5 / 9, f_temps)
```

The `reduce` Function

- `reduce` is a function that collapses (*reduces*) values from a list into a single value (called `foldr` in some languages)
  - e.g. add each pair of elements, repeatedly to get a sum

```
from functools import reduce
def add2(x,y):
    return x + y
sum1 = reduce(add2, [1,2,3,4,5])

sum2 = reduce(lambda x,y: x + y, [1,2,3,4,5])
```

Coding Exercise 05a.1

- Using the `reduce` function, take a list of dictionaries called `invoice_items`, and computes the total cost:
  - Each dictionary in `invoice_items` has a field named `item_price`, and another field named `quantity`
  - The total cost for each item is `item_price * quantity`
  - The function that you pass to reduce will obtain these two quantities, and add their product to the sum
    - Try it with a lambda function, if you can

The `filter` Function

- `filter` is a function that eliminates (*filters*) items based on some condition
  - e.g. find all values greater than some threshold value

```
def a_range(mark):
    return mark >= 80.0
marks = [64.5, 87.0, 55.5, 94.0, 71.5, 46.0, 100.0]
a_grades1 = filter(a_range, marks)

a_grades2 = filter(lambda mark: mark >= 80.0, marks)
```

Coding Exercise 05a.2

- Write your own version of the `filter` function, called `myfilter`, which:
  - Takes a unary function (`check`) and a list (`values`) as arguments
  - Applies the function `check` to successive each value from the list, and if the result is `True`, adds the value to the output list
- For example:

```
marks = [64.5, 87.0, 55.5, 94.0, 71.5, 46.0, 100.0]
a_grades = myfilter(lambda mark: mark > 80.0, marks)
# a_grades should be [87.0, 94.0, 100.0]
```

Wrap-up

- Stacks and the Calling Stack
- Higher-order functions
  - Passing functions as arguments to other functions

Coming Up

- Recursion
    - Recursive function calling and backtracking
    - Recursive functions and the calling stack
    - Tail recursion