Lecture 9: Memory in Python

CS 1110
Introduction to Computing Using Python

Announcements

- A1 revision process: A1 closed now on CMS for grading. Set your CMS notifications to “receive email when…” When feedback is released, expected on Mar 13 afternoon, read resubmission instructions
- A2 to be released Thursday

Where is the robot?

- Angle of the robot relative to the sensor is \( d \) degrees, where \( d \) is non-negative
- Robot is in which quadrant?
- To avoid ambiguity, use this convention:
  - 1 if \( 0 \leq d < 90 \)
  - 2 if \( 90 \leq d < 180 \)
  - 3 if \( 180 \leq d < 270 \)
  - 4 if \( 270 \leq d < 360 \)

Can solve using if-elif-elif... Other options?

Review: Nested Conditionals

Nesting Conditionals

- Separate choices into 2 general categories
- Subdivide each category into subcategories
- Subdivide each subcategory further...

```
if <above x-axis>:
  if <left of y-axis>:
  else:
else:
  if <left of y-axis>:
  else:
```

See quadrants.py

Memory in Python

[Text in fusia and extra slides were added after lecture for clarification. See slides 24 - 27.]

Global Space

- Global Space
  - What you “start with”
  - Stores global variables
  - Lasts until you quit Python

Stores global variables
Lasts until you quit Python

$\text{x} = 4$

Enter Heap Space

- Global Space
  - What you “start with”
  - Stores global variables
  - Lasts until you quit Python

- Heap Space
  - Where “folders” are stored
  - Have to access indirectly

$p \& q$ live in Global Space. Their folders live on the Heap.

Calling a Function Creates a Call Frame

What’s in a Call Frame?
- Boxes for parameters at the start of the function
- Boxes for variables local to the function as they are created

Putting it all together

- Global Space
  - What you “start with”
  - Stores global variables
  - Lasts until you quit Python

- Heap Space
  - Where “folders” are stored
  - Have to access indirectly

- Call Frames
  - Parameters
  - Other variables local to function
  - Lasts until function returns

Two Points Make a Line

```
start = shape.Point2(0,0)
stop = shape.Point2(0,0)
print("Where does the line start?")
x = input("x: ")
start.x = int(x)
y = input("y: ")
start.y = int(y)
print("The line starts at ("+x+","+y+").")
x = input("x: ")
stop.x = int(x)
y = input("y: ")
stop.y = int(y)
print("The line stops at ("+x+","+y+").")
```
Redundant Code is BAAAAD!

```
start = shape.Point2(0,0)
stop = shape.Point2(0,0)
print("Where does the line start?")
x = input("x: ")
start.x = int(x)
y = input("y: ")
start.y = int(y)
print("The line starts at ("+x+","+y+").")
print("Where does the line stop?")
x = input("x: ")
stop.x = int(x)
y = input("y: ")
stop.y = int(y)
print("The line stops at ("+x+","+y+").")
```

Let’s make a function!

```
def configure(pt, role):
    print("Where does the line " + role + "?")
x = input("x: ")
pt.x = int(x)
y = input("y: ")
pt.y = int(y)
print("The line " +role+ "s at ("+x+","+y+").")
```
Q1: what do you do next?

```
def get_coord(name):
    x = input(name + ':')
    return int(x)

def configure(pt, role):
    pt.x = get_coord('x')
    pt.y = get_coord('y')
    print("The line * role* + *s at "+str(pt.y)+ "*\n")
    start = shape.Point2(0,0)
    configure(start, "start")
```

Heap Space
```
<table>
<thead>
<tr>
<th>id1</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
</tr>
<tr>
<td>y</td>
</tr>
</tbody>
</table>
```

Global Space
```
<table>
<thead>
<tr>
<th>start</th>
</tr>
</thead>
<tbody>
<tr>
<td>id1</td>
</tr>
</tbody>
</table>
```

A: Cross out the configure call frame.
B: Create a get_coord call frame.
C: Cross out the 4 in the call frame.
D: A & B
E: B & C

The Call Stack
- The set of function frames drawn in call order
- Functions frames are “stacked”
  - Cannot remove one above w/o removing one below
- Python must keep the entire stack in memory
  - Error if it cannot hold stack (“stack overflow”)
Errors and the Call Stack

def get_coord(name):
    x = input(name +": ")
    return int(x)
def configure(pt, role):
    print("Where does the line " + role + "?")
    pt.x = get_coord("x")
    pt.y = get_coord("y")
    print("The line " + role + " is at (" + x + "," + y + ")").

start = shape.Point2(0,0)
configure(start, "start")

Traceback (most recent call last):
  File "v3.py", line 19, in <module>
    configure(start, "start")
  File "v3.py", line 14, in configure
    pt.x = get_coord("x")
NameError: name 'x1' is not defined

Q2: what does the call stack look like at this point in the execution of the code?

 Modules and Global Space

Import
• Creates a global variable (same name as module)
• Puts variables, functions of module in a folder
• Puts folder id in the global variable

Functions and Global Space

A function definition
• Creates a global variable (same name as function)
• Creates a folder for body
• Puts folder id in the global variable

Modules vs Objects

Function Definition vs. Call Frame
Storage in Python

- **Global Space**
  - What you “start with”
  - Stores global variables, modules & functions
  - Lasts until you quit Python

- **Heap Space**
  - Where “folders” are stored
  - Have to access indirectly

- **Call Frame Stack**
  - Parameters
  - Other variables local to function
  - Lasts until function returns

Don’t draw module folder, function folder

Folders that we do not draw:
- Module folder is created upon `import`, for example, `import math`
- Function folder is created with `def` (the function header), for example, `def get_feet(height_in_inches):`

Don’t draw those folders and the variables that store their ids; we only explained those folders to explain what you see in Python Tutor. **Do not draw them.**

Q3: what does the call stack look like at this point in the execution of the code?

```python
def f30:
    print("f3")
def f20:
    print("f2")
f30
f30
f30
def f10:
    print("f1")
f10
```

Choose 1 stack; in that stack cross out any frames that should have ended.