Lecture 9: Memory in Python

CS 1110
Introduction to Computing Using Python

[E. Andersen, A. Bracy, D. Gries, L. Lee, S. Marschner, C. Van Loan, W. White]
Feb 27: CS 1110: Announcements

- Last call for a one-on-one!
  - CMS: OPTIONAL: one-on-ones

- Prelim 1 is March 13. You have until March 1st, 11:59pm to register a conflict or a need for accommodation. There is no single makeup session. See website: “Assessment ➔ Exams”
  - CMS: Prelim 1 conflicts

Storage in Python – Round 1

• **Global Space**
  - What you “start with”
  - Stores global variables
  - Lasts until you quit Python
Folders Live on the Heap

\[ p = \text{shapes.Point3}(1,2,3) \]

\( p \) lives in the Global Space. Its folder lives on the Heap.
Storage in Python – Round 2

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

- **Heap Space**
  - Where “folders” are stored
  - Have to access indirectly
 Calling a Function Creates a Call Frame

```python
p = shapes.Point3(1, 2, 3)
incr_x(p)
```
What goes in a Call Frame?

\[
p = \text{shapes.Point3}(1,2,3)
\]

```python
def incr_x(the_point):
    the_point.x = the_point.x + 1
incr_x(p)
```

(1) Boxes for parameters **at the start of the function**
(2) Boxes for variables local to the function **as they are created**
Storage in Python – Round 3

- **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
Frames and Helper Functions

- Functions can call each other!
- Each call creates a *new call frame*
- Function that exists mainly to call other functions is often called a *helper function*
def last_name_first(n):
    """Returns: copy of <n> but in the form <last-name>, <first-name>

    Precondition: <n> is in the form <first-name> <last-name>
    with one or more blanks between the two names. """

    space_index = n.find(' ')
    first = n[:space_index]
    last = n[space_index+1:].strip()
    return last+', '+first

• last_name_first('Haruki Murakami') gives 'Murakami, Haruki'
• last_name_first('Haruki    Murakami') gives '    Murakami, Haruki'
Frames and Helper Functions

```python
def first_name(s):
    
    """Prec: see last_name_first""

    end = s.find(' ')

    return s[0:end]

def last_name_first(s):
    
    """Precondition: s in the form <first-name> <last-name>""

    first = first_name(s)
    last = last_name(s)

    return last + ', ' + first

def last_name(s):
    
    """Prec: see last_name_first""

    end = s.rfind(' ')

    return s[end+1:]
```

`rfind` gets the last instance of substring.
def last_name_first(s):
    """Precondition: s in the form <first-name> <last-name>"""
    first = first_name(s)
    last = last_name(s)
    return last + ', ' + first

def first_name(s):
    """Prec: see last_name_first"""
    end = s.find(' ')
    return s[0:end]

last_name_first('Haruki Murakami')
def last_name_first(s):
    """Precondition: s in the form <first-name> <last-name>""
    first = first_name(s)
    last = last_name(s)
    return last + ',' + first

def first_name(s):
    """Prec: see last_name_first""
    end = s.find(' ')
    return s[0:end]

last_name_first('Haruki Murakami')
```python
def last_name_first(s):
    
    
    """Precondition: s in the form <first-name> <last-name>""

    first = first_name(s)
    last = last_name(s)
    return last + ',' + first

def first_name(s):
    
    """Prec: see last_name_first""

    end = s.find(' ')
    return s[0:end]
```

```
last_name_first('Haruki Murakami')
```
def last_name_first(s):
    """Precondition: s in the form <first-name> <last-name>""
    first = first_name(s)
    last = last_name(s)
    return last + ',' + first

def first_name(s):
    """Prec: see last_name_first""
    end = s.find(' ')
    return s[0:end]

last_name_first('Haruki Murakami')
def last_name_first(s):
    
    """Precondition: s in the form <first-name> <last-name>"""
    
    first = first_name(s)
    last = last_name(s)
    return last + ', ' + first

def first_name(s):
    
    """Prec: see last_name_first"""
    
    end = s.find(' ')
    return s[0:end]

last_name_first('Haruki Murakami')
Answer: What Happens Next?

def last_name_first(s):
    """Precondition: s in the form "<first-name> <last-name>"""
    first = first_name(s)
    last = last_name(s)
    return last + ',' + first

def first_name(s):
    """Prec: see last_name_first""
    end = s.find(' ')
    return s[0:end]

last_name_first('Haruki Murakami')
def last_name_first(s):
    """Precondition: s in the form <first-name> <last-name>""
    first = first_name(s)
    last = last_name(s)
    return last + '.' + first

def last_name(s):
    """Prec: see last_name_first""
    end = s.rfind(' ')
    return s[end+1:]

last_name_first('Haruki Murakami')
def last_name_first(s):
    
    """Precondition: s in the form <first-name> <last-name>"""
    first = first_name(s)
    last = last_name(s)
    return last + '. ' + first

def last_name(s):
    
    """Prec: see last_name_first"""
    end = s.rfind(' ')
    return s[end+1:]

last_name_first('Haruki Murakami')
The Call Stack

- 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”)

```
function1
function2
function3
function4
function5
```
def happy_birthday():
    print(“Happy Birthday”)

def dear():
    print(“Dear James”)

def to_you():
    print(“to you”)

def line_with_name():
    happy_birthday()
    dear()

def basic_line():
    happy_birthday()
    to_you()

def song():
    basic_line()
    basic_line()
    line_with_name()
    basic_line()
song()
def happy_birthday():
    print("Happy Birthday")

def dear():
    print("Dear James")

def to_you():
    print("to you")

def line_with_name():
    happy_birthday()
    dear()

def basic_line():
    happy_birthday()
    to_you()

def song():
    basic_line()
    basic_line()
    basic_line()
    line_with_name()
    basic_line()
    song()
def happy_birthday():
    print(“Happy Birthday”)

def dear():
    print(“Dear James”)

def to_you():
    print(“to you”)

def line_with_name():
    happy_birthday()
    dear()

def basic_line():
    happy_birthday()
    to_you()

def song():
    basic_line()
    basic_line()
    line_with_name()
    basic_line()
    song()
Call Stack Example (4)

```python
def happy_birthday():
    print("Happy Birthday")

def dear():
    print("Dear James")

def to_you():
    print("to you")

def line_with_name():
    happy_birthday()
    dear()

def basic_line():
    happy_birthday()
    to_you()

def song():
    basic_line()
    basic_line()
    line_with_name()
    basic_line()
    song()
```

Happy Birthday
def happy_birthday():
    print("Happy Birthday")

def dear():
    print("Dear James")

def to_you():
    print("to you")

def line_with_name():
    happy_birthday()
    dear()

def basic_line():
    happy_birthday()
    to_you()

def song():
    basic_line()
    basic_line()
    line_with_name()
    basic_line()
    song()
def happy_birthday():
    print("Happy Birthday")

def dear():
    print("Dear James")

def to_you():
    print("to you")

def line_with_name():
    happy_birthday()
    dear()

def basic_line():
    happy_birthday()
    to_you()

def song():
    basic_line()
    basic_line()
    line_with_name()
    basic_line()
    song()
```python
def happy_birthday():
    print("Happy Birthday")

def dear():
    print("Dear James")

def to_you():
    print("to you")

def line_with_name():
    happy_birthday()
    dear()

def basic_line():
    happy_birthday()
    to_you()

def song():
    basic_line()
    basic_line()
    line_with_name()
    basic_line()
    song()
```

Happy Birthday
to you

Call Frame Stack

```
song     14

basic_line     12
to_you

RETURN None
```
```python
def happy_birthday():
    print("Happy Birthday")

def dear():
    print("Dear James")

def to_you():
    print("to you")

def line_with_name():
    happy_birthday()
    dear()

def basic_line():
    happy_birthday()
    to_you()

def song():
    basic_line()
    basic_line()
    line_with_name()
    basic_line()
    song()
```

Happy Birthday
to you
def happy_birthday():
    print("Happy Birthday")

def dear():
    print("Dear James")

def to_you():
    print("to you")

def line_with_name():
    happy_birthday()
    dear()

def basic_line():
    happy_birthday()
    to_you()

def song():
    basic_line()
    basic_line()
    line_with_name()
    basic_line()
    song()
def happy_birthday():
    print("Happy Birthday")

def dear():
    print("Dear James")

def to_you():
    print("to you")

def line_with_name():
    happy_birthday()
    dear()

def basic_line():
    happy_birthday()
    to_you()

def song():
    basic_line()
    basic_line()
    line_with_name()
    basic_line()
    song()

Happy Birthday
to you
Errors and the Call Stack

```python
def happy_birthday():
    print("Happy Birthday")
def dear():
    print("Dear "+name)
def to_you():
    print("to you")
def line_with_name():
    happy_birthday()
dear()
def basic_line():
    happy_birthday()
to_you()
def song():
    basic_line()
basic_line()
line_with_name()
basic_line()
song()

Happy Birthday
to you
Happy Birthday
to you
Happy Birthday

Traceback (most recent call last):
  File "birthday_error.py", line 18, in <module>
    song()
  File "birthday_error.py", line 16, in song
    line_with_name()
  File "birthday_error.py", line 9, in line_with_name
dear()
  File "birthday_error.py", line 4, in dear
    print("Dear "+name)
NameError: name 'name' is not defined
```

Make sure you can see line numbers in Komodo.
Preferences ➔ Editor
def happy_birthday():
    print("Happy Birthday")
def dear():
    print("Dear " + name)
def to_you():
    print("to you")
def line_with_name():
    happy_birthday()
dear()
def basic_line():
    happy_birthday()
to_you()
def song():
    basic_line()
basic_line()
line_with_name()
basic_line()
song()

Happy Birthday
to you
Happy Birthday
to you
Happy Birthday

Traceback (most recent call last):
  File "birthday_error.py", line 18, in <module>
    song()
  File "birthday_error.py", line 16, in song
    line_with_name()
  File "birthday_error.py", line 9, in line_with_name
dear()
  File "birthday_error.py", line 4, in dear
    print("Dear " + name)
NameError: name 'name' is not defined
Functions and Global Space

A function definition

- Creates a global variable (same name as function)
- Creates a **folder** for body
- Puts folder id in variable

```
INCHES_PER_FT = 12
def get_feet(ht_in_inches):
    return ht_in_inches // INCHES_PER_FT
```

See for yourself: [https://tinyurl.com/get-feet](https://tinyurl.com/get-feet)
Function Definition vs. Call Frame

**Global Space**
- (memory for function call goes here)

**Heap Space**
- (Function definition goes here)

**Call Frame**
- memory for function call goes here

*It’s alive!*

```python
INCHES_PER_FOOT = 12

def get_feet(height_in_inches):
    return height_in_inches // INCHES_PER_FOOT

feet = get_feet(68)
print("you are at least "+str(feet)+" feet tall!")
```
import

- Creates a global variable (same name as module)
- Puts variables, functions in a folder
- Puts folder id in variable

Global Space

math  id5

Heap Space

id5

pi  3.141592

e  2.718281

functions

module
import shapes

def incr_x(pt):
    pt.x = pt.x + 1

p = shapes.Point3(1,2,3)
incr_x(p)

Narration:
Python knows there is a shapes module
And it knows where to find it (→ on the Heap).
import shapes

def incr_x(pt):
    pt.x = pt.x + 1

p = shapes.Point3(1, 2, 3)
incr_x(p)

Narration:
Python knows there is a function called `incr_x`
And it knows where to find the definition (→ on the Heap).
import shapes

def incr_x(pt):
    pt.x = pt.x + 1

p = shapes.Point3(1, 2, 3)
incr_x(p)

Narration:
Python just created a point \texttt{p}.
\texttt{p} lives in the Global Space. Its folder lives on the Heap.
import shapes

def incr_x(pt):
    pt.x = pt.x + 1

p = shapes.Point3(1,2,3)
incr_x(p)

Narration:
Python just created a call frame for the function `incr_x`
Parameter `pt` lives in the call frame. Its value is whatever
the argument to `incr_x` was.
import shapes

def incr_x(pt):
    pt.x = pt.x + 1

p = shapes.Point3(1,2,3)
incr_x(p)

Narration:
Python is about to execute the first instruction in the function \texttt{incr\_x}
import shapes

def incr_x(pt):
    pt.x = pt.x + 1

p = shapes.Point3(1,2,3)
incr_x(p)

Narration:
Python just executed the lone instruction in the function **incr_x**. Since there is no return statement, the return value of NONE is created in the call frame.
import shapes

def incr_x(pt):
    pt.x = pt.x+1

p = shapes.Point3(1,2,3)
incr_x(p)

Narration:
Python just returned from the function `incr_x`. Its call frame is destroyed because the function is done.
Storage in Python (Final Version!)

• **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
print("Welcome to the Star Wars Name Generator!")
print("Inputs must be 3 letters or longer. What is your...")
first = input("...First name? ")
middle = input("...Middle name? ")
last = input("...Last name? ")
town = input("...Hometown? ")

def make_name(name1, name2):
    name = name1[0:3]+name2[0:3].lower()
    return name

new_first = make_name(first, last)
new_last = make_name(middle, town)
print("Your Star Wars name is: "+new_first+" "+new_last)
print("Welcome to the Star Wars Name Generator!")
print("Inputs must be 3 letters or longer. What is your...")
first = input("...First name? ")
middle = input("...Middle name? ")
last = input("...Last name? ")
town = input("...Hometown? ")

def make_name(name1, name2):
    name = name1[0:3]+name2[0:3].lower()
    return name

new_first = make_name(first, last)
new_last = make_name(middle, town)
print("Your Star Wars name is: "+new_first+" "+new_last)
print("Welcome to the Star Wars Name Generator!")
print("Inputs must be 3 letters or longer. What is your...")
first =  input("...First name? ")
middle = input("...Middle name? ")
last = input("...Last name? ")
town = input("...Hometown? ")

def make_name(name1, name2):
    name = name1[0:3]+name2[0:3].lower()
    return name

new_first = make_name(first, last)
new_last = make_name(middle, town)
print("Your Star Wars name is: "+new_first+" "+new_last)

Question: What if this variable were called first?