CS134 Lecture 13: Scope
Announcements & Logistics

- **Lab 04 Feedback is out!** Can you interpret TestResults.txt?
- Lab 03 Graded feedback is out
- **HW 5** will due tonight @ 10pm
- **Lab 4 Part 2** due Wednesday/Thursday 10pm
- **Midterm reminders:**
  - **Review:** **Monday 3/11** from 7-9pm
  - **Exam Thurs 3/14** from 6-7:30pm OR 8-9:30pm
  - Both exam and review are in Bronfman Auditorium

Do You Have Any Questions?
Last Time: Aliasing

- Attempts to change immutable objects (e.g., strings) produce clones
  - Changes to clones do not affect originals
    - No aliasing!
- We can create aliases of mutable objects
  - Aliases refer to the same object, so changes to that object through any alias affect value that other aliases observe
- For the list data type, \(+\) is sneakily replaced by \(\texttt{.append()}\)
  - This mutates the list!

Goal was to demystify surprising behavior: nothing in computer science is magic!
Today's Plan

- **Scope**: variables, functions, objects have limited accessibility/visibility.
  - Understanding how this works helps us make decisions about where to define variables/functions/objects

Goal is to again demystify surprising behavior: nothing in computer science is magic!
def my_func(val):
    val = val + 1
    print('local val', val)
    return val

val = 3
new_val = my_func(val)
print('global val', val)
def my_func (val):
    val = val + 1
    print('local val', val)
    return val

val = 3
new_val = my_func(val)
print('global val', val)
def my_func(val):
    val = val + 1
    print('local val', val)
    return val

val = 3
new_val = my_func(val)
print('global val', val)
def my_func (val):
    val = val + 1
    print('local val', val)
    return val

val = 3
new_val = my_func(val)
print('global val', val)
def my_func(val):
    val = val + 1
    print('local val', val)
    return val

val = 3
new_val = my_func(val)
print('global val', val)

Function frame destroyed (and all local variables lost) after return from call

Information flow out of a function is only through return statements!
What gets printed to the screen?

```python
da = 3
b = 4

def square(x):
    return x * x

sum_sq = square(a) + square(b)
c = sum_sq ** 0.5

print(c)
```

What is printed here?
Understanding Scope

```python
a = 3
b = 4

def square(x):
    return x * x

sum_sq = square(a) + square(b)
c = sum_sq ** 0.5
print(c)
```
a = 3
b = 4

def square(x):
    return x * x

sum_sq = square(a) + square(b)
Understanding Scope

```python
da = 3
db = 4

def square(x):
    return x * x

sum_sq = square(3) + square(b)
c = sum_sq ** 0.5
print(c)
```
Understanding Scope

```
a = 3
b = 4

def square(x):
    return x * x

sum_sq = square(3) + square(4)
c = sum_sq ** 0.5
print(c)
```
a = 3
b = 4

def square(x):
    return x * x

sum_sq = square(3) + square(4)

c = sum_sq ** 0.5
print(c)
a = 3
b = 4

def square(x):
    return x * x

sum_sq = square(3) + square(4)

c = sum_sq ** 0.5
print(c)
Understanding Scope

```
a = 3  
b = 4

def square(x):
    return x * x

sum_sq = square(3) + square(4)
```

Function frame destroyed (and all local variables lost) after return from function call.
a = 3
b = 4

def square(x):
    return x * x

sum_sq = 9 +
    square(4)

c = sum_sq ** 0.5
print(c)
Understanding Scope

```python
da = 3
b = 4
def square(x):
    return x * x

sum_sq = 9 +
    square(4)
c = sum_sq ** 0.5
print(c)
```

Global scope:
- `a`: 3
- `b`: 4
- `sum_sq`: -

Square frame:
- `x`: 3
- `return 9`

Square frame:
- `x`: 4
- `return x * x`
a = 3
b = 4
def square(x):
    return x * x
sum_sq = 9 +
    square(4)

c = sum_sq ** 0.5
print(c)
Understanding Scope

```python
da = 3
db = 4
def square(x):
    return x * x
sum_sq = 9 + square(4)
c = sum_sq ** 0.5
print(c)
```

- **Global scope**
  - `a = 3`
  - `b = 4`
- **square frame**
  - `x` with value 3
  - `return 9`

- **square frame**
  - `x` with value 4
  - `return x * x`
def square(x):
    return x * x

sum_sq = 9 + 16

c = sum_sq ** 0.5
print(c)
Understanding Scope

```python
a = 3
b = 4

def square(x):
    return x * x

sum_sq = 25
```

![Scope Diagram]

- **Global Scope**
  - `a = 3`
  - `b = 4`
  - `sum_sq = 25`

- **Function Scope**
  - `square(x)`: Some code

```
print(c)
```
Understanding Scope

```python
da = 3
db = 4

def square(x):
    return x * x

sum_sq = 25

c = sum_sq ** 0.5

print(c)
```

Global scope
- `a`: 3
- `b`: 4
- `sum_sq`: 25
- `c`: -

Some code
Understanding Scope

```
a = 3
b = 4

def square(x):
    return x * x

sum_sq = 25

c = 25 ** 0.5  
```

<table>
<thead>
<tr>
<th>Global scope</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
</tr>
<tr>
<td>b</td>
</tr>
<tr>
<td>sum_sq</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>square</th>
</tr>
</thead>
</table>

| Some code |

print(c)
a = 3
b = 4

def square(x):
    return x * x

sum_sq = 25

c = 5

print(c)

Finally, 5 is printed
Local Parameter Names

```python
a = 3
b = 4

def square(a):
    return a * a

sum_sq = square(a) + square(b)
c = sum_sq ** 0.5

print(c)
```

What if we change this?

Does it change the behavior?
Local Parameter Names

```
a = 3
b = 4

def square(a):
    return b * b

sum_sq = square(a) + square(b)
c = sum_sq ** 0.5

print(c)
```

How about this change?

Will it throw NameError?
Understanding Scope

a = 3
b = 4

def square(a):
    return b * b

sum_sq = square(a) + square(b)
c = sum_sq ** 0.5
print(c)
a = 3
b = 4

```python
def square(a):
    return b * b

sum_sq = square(a) + square(b)
```

Understanding Scope
Global scope
square
Some code

def square(a):
    return b * b

sum_sq = square(3) + square(b)

c = sum_sq ** 0.5
print(c)

Understanding Scope
Understanding Scope

```python
a = 3
b = 4

def square(a):
    return b * b

sum_sq = square(3) + square(4)

print(sum_sq ** 0.5)
```
Understanding Scope

```python
da = 3
db = 4
def square(a):
    return b * b

sum_sq = square(3) + square(4)
c = sum_sq ** 0.5
print(c)
```
a = 3
b = 4

def square(a):
    return b * b

sum_sq = square(3) + square(4)

c = sum_sq ** 0.5
print(c)
a = 3
b = 4

def square(a):
    return b * b

sum_sq = square(3) + square(4)

c = sum_sq ** 0.5
print(c)
Understanding Scope

```python
a = 3
b = 4

def square(a):
    return b * b

sum_sq = 16 +
    square(4)

c = sum_sq ** 0.5
print(c)
```

Global scope

```
square
Some code

Global scope
a 3
b 4
sum_sq -

square frame

x 3

return 9

square frame

a 4

return b * b
```
a = 3
b = 4
def square(a):
    return b * b

sum_sq = 16 +
    square(4)
Understanding Scope

```python
a = 3
b = 4

def square(a):
    return b * b

sum_sq = 16 +
    square(4)

c = sum_sq ** 0.5
print(c)
```

```
Square frame
---
a 4
return 4 * 4
---

Global scope
- a
  3
- b
  4
- sum_sq
  -

Square
---
some code
---
```
a = 3
b = 4

```python
def square(a):
    return b * b

sum_sq = 16 +
square(4)
```

Output:

```
4
```
Understanding Scope

\[
\begin{align*}
a &= 3 \\
b &= 4 \\
def\ square(a): \\
    &\quad \text{return } b \times b \\
\end{align*}
\]

\[
\begin{align*}
\text{sum}_{\text{sq}} &= 16 + \\
&\quad 16 \\
\end{align*}
\]
Understanding Scope

```
a = 3
b = 4

def square(a):
    return b * b

sum_sq = 32
```

**Global scope**

<table>
<thead>
<tr>
<th>a</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>4</td>
</tr>
<tr>
<td>sum_sq</td>
<td>32</td>
</tr>
</tbody>
</table>

**square**

Some code
a = 3
b = 4

def square(a):
    return b * b

sum_sq = 32

c = sum_sq ** 0.5

print(c)

Understanding Scope
a = 3
b = 4

def square(a):
    return b * b

sum_sq = 32

c = 32 ** 0.5

print(c)
a = 3
b = 4

def square(a):
    return b * b

sum_sq = 32
c = 5.656854249492381

print(c)

Finally, 5.656854249492381 is printed
Takeaway: Local Before Global

When python encounters a new term, like a variable or function name, it first looks locally, before looking higher up.

If it can't find the value assigned to the term, you get a NameError.
More Examples in Notebook
What gets printed to the screen?

```python
def mystery(num):
    return multiplier * num
multiplier = 2
answer = mystery(5)
print(answer)
```

??
What gets printed to the screen?

```python
def mystery(num):
    return multiplier * num
multiplier = 2
answer = mystery(5)
print(answer)
```

- `multiplier` is recorded as 3 on the Global Frame
- Then the `mystery()` blueprint is recorded on the Global Frame
- Then `multiplier` is re-assigned the value 2 on the Global Frame
- ...
What gets printed to the screen?

```python
list = 2468
list_str = list("whoops")
print(list, list_str)
```
What gets printed to the screen?

list = 2468
list_str = list("whoops")
print(list, list_str)

TypeError: 'list' object is not callable

- list is a python keyword, in the Global Frame
- list = ... reassigns the value of list in the Global Frame
  - It's no longer the keyword, it's now an integer object
- So you can't call list(...) as the built-in list-casting function!
- ...This is why we don't use python keywords as variable names.
Helpful Tool for Learning How python Executes Code