CIS 122

Functions Under the Surface
Functions Revisited

● We now have the power to write our own functions

```python
def plusOne(x):
    """Adds one to x""
    return x+1
```

● Who cares?
  ○ We could just write the same code outside a function...
    ■ y = plusOne(x)
    ■ y = x+1
  ○ Why do we need functions?
Functions Revisited

- Functions simplify coding
  - Easier to solve small problems
  - Construct building blocks

- Reduce redundancy
  - Don't write the same 5 lines of code over and over
  - Write one function and call it 5 times

- Explain code
  - Descriptive function names
A Capital Idea

- Let's write a function to capitalize a letter
  - Requires some background knowledge
  - How does Python represent letters?

- Under the surface, characters are just numbers
  - 'A' → 65
  - 'a' → 97
  - '%'

A Capital Idea

- We can convert from one to the other

- `ord` method converts characters to numbers
  ```python
  >>> ord('a')
  97
  ```

- `chr` method converts numbers to characters
  ```python
  >>> chr(97)
  'a'
  ```
A Capital Idea

● What's the difference between a lower-case letter and an upper-case letter?

● What sequence of operations would convert a lower-case letter to an upper-case letter?

● Let's write a function!
def capitalize(lowerCaseC):
    """Capitalizes lowerCaseC"""

    lowerCaseN = ord(lowerCaseC)
    upperCaseN = lowerCaseN - 32
    upperCaseC = chr(upperCaseN)
    return upperCaseC
Stack Diagrams

- We've seen two different ways to instantiate variables
  - Variable assignment
    - numDots = 5
  - Function calls
    - capitalize('a')

- How does python keep track of which variables exist?
def foo(x):
    y = x+1
    z = x+y
    return z

a = 5
b = foo(a)
c = a+b
def foo(x):
    y = x + 1
    z = x + y
    return z

__main__
a = 5
b = foo(a)
c = a + b
def foo(x):
    y = x + 1
    z = x + y
    return z

__main__
foo → <function object>

a = 5
b = foo(a)
c = a + b
def foo(x):
    y = x+1
    z = x+y
    return z

__main__
    foo → <function object>
    a → 5

a = 5
b = foo(a)
c = a+b
def foo(x):
    y = x + 1
    z = x + y
    return z

_a = 5
b = foo(a)
c = a + b

__main__

foo → <function object>
a → 5
b → ???
Stack Diagrams

def foo(x):
    y = x+1
    z = x+y
    return z

__main__
    foo → <function object>
    a → 5
    b → ???

a = 5
b = foo(a)
c = a+b
def foo(x):
    y = x+1
    z = x+y
    return z

__main__
foo → <function object>
a → 5
b → ???

a = 5
b = foo(a)
c = a+b
def foo(x):
    y = x+1
    z = x+y
    return z

__main__
foo → <function object>
    a → 5
    b → ???

    foo
    x → 5
    y → 6

a = 5
b = foo(a)
c = a+b
def foo(x):
    y = x+1
    z = x+y
    return z

__main__
    foo → <function object>
    a → 5
    b → ???

foo
    x → 5
    y → 6
    z → 11

a = 5
b = foo(a)
c = a+b
def foo(x):
    y = x+1
    z = x+y
    return z

__main__
    foo → <function object>
    a → 5
    b → ???

a = 5
b = foo(a)
c = a+b

foo
    x → 5
    y → 6
    z → 11
def foo(x):
    y = x+1
    z = x+y
    return z

__main__
foo → <function object>
a → 5
b → 11

a = 5
b = foo(a)
c = a+b

foo
x → 5
y → 6
z → 11
def foo(x):
    y = x+1
    z = x+y
    return z

a = 5
b = foo(a)
c = a+b

__main__
foo → <function object>
a → 5
b → 11
c → 16

foo
x → 5
y → 6
z → 11
Stack Diagrams

- Code doesn't always run linearly
  - During function calls, other code is put on hold
  - Python creates a new **stack frame** in memory
  - These stack frames can nest

- Who's seen the movie Inception?
Variable Scoping

- Variables exist within a specific scope
  - Only make sense within a certain context

- Variables within a function cannot be seen from outside
  - Don't overwrite outside variables
  - Deleted when function ends
def foo(x):
    z = x + 1
    return z

x = 5
y = foo(6)
def foo(x):
    __main__
    z = x + 1
    return z

x = 5
y = foo(6)
def foo(x):
    z = x + 1
    return z

x = 5
y = foo(6)
Variable Scoping

def foo(x):
    z = x + 1
    return z

x = 5
y = foo(6)
Variable Scoping

def foo(x):
    z = x + 1
    return z

x = 5
y = foo(6)

__main__

foo → <function object>
x → 5
y → ???
def foo(x):
    z = x + 1
    return z

x = 5
y = foo(6)

__main__
foo → <function object>
x → 5
y → ???
def foo(x):
    z = x + 1
    return z

x = 5
y = foo(6)

___main___

foo → <function object>
    x → 5
    y → ???

foo
    x → 6
def foo(x):
    z = x + 1
    return z

x = 5
y = foo(6)

__main__

foo → <function object>
    x → 5
    y → ???

foo
    x → 6
    z → 7
def foo(x):
    z = x + 1
    return z

x = 5
y = foo(6)

__main__
foo → <function object>
    x → 5
    y → ???

foo
    x → 6
    z → 7
def foo(x):
    z = x + 1
    return z

x = 5
y = foo(6)
Variable Scoping

• Why is variable scoping important?
  ○ Lots of built in functions in Python
  ○ We don't know (or care) how they're written
  ○ My code shouldn't depend on someone else's variable names!