# CSCI 141


Lecture 23
Mutable objects and Functions

# Announcements

# Announcements

- Please get started on A5.

# Announcements

- Please get started on A5.
- Slip days apply to A5, **but**:

# Announcements

- Please get started on A5.
- Slip days apply to A5, **but**:
  - **No late submissions accepted after Thursday 12/5 at 10pm**

# Announcements

- Please get started on A5.
- Slip days apply to A5, **but**:
  - **No late submissions accepted after Thursday 12/5 at 10pm**
- Now is the time to start organizing your study plan for the final exam.

# Goals

- Understand how mutable objects interact with function calls and scope:
  - Objects do not live inside the "boxes" that define scope
  - References to objects can cross "box" boundaries.
- Be able to draw memory diagrams for programs that involve function calls and mutable objects.

# QOTD

```
a = [ 3, 4, 5 ]                  [3, 4, 5]
a.insert(0, 4)                   [4, 3, 4, 5]
a[2:] = a[ 1:4 ]                 [4, 3, 3, 4, 5]
a.remove(4)                      [3, 3, 4, 5]
a.append(a.index(5))             [3, 3, 4, 5, 3]
del a[ 1 ]                       [3, 4, 5, 3]
print(len(a))                    4
print(4 not in a)                False
print(a[ -2 ])                   5
```

# QOTD

```
a = [3]
b = a
a.append(4)
c = a[0]
d = b
a.extend((17, 19))
x = a[-2:]
e = x + [4]
```

- How many lists are created? **4**

- How many variables point to the same list as a?

# QOTD

```
a = [3]
b = a
a.append(4)
c = a[0]
d = b
a.extend((17, 19))
x = a[-2:]
e = x + [4]
```

- How many lists are created? **4**

- How many variables point to the same list as a? **2**

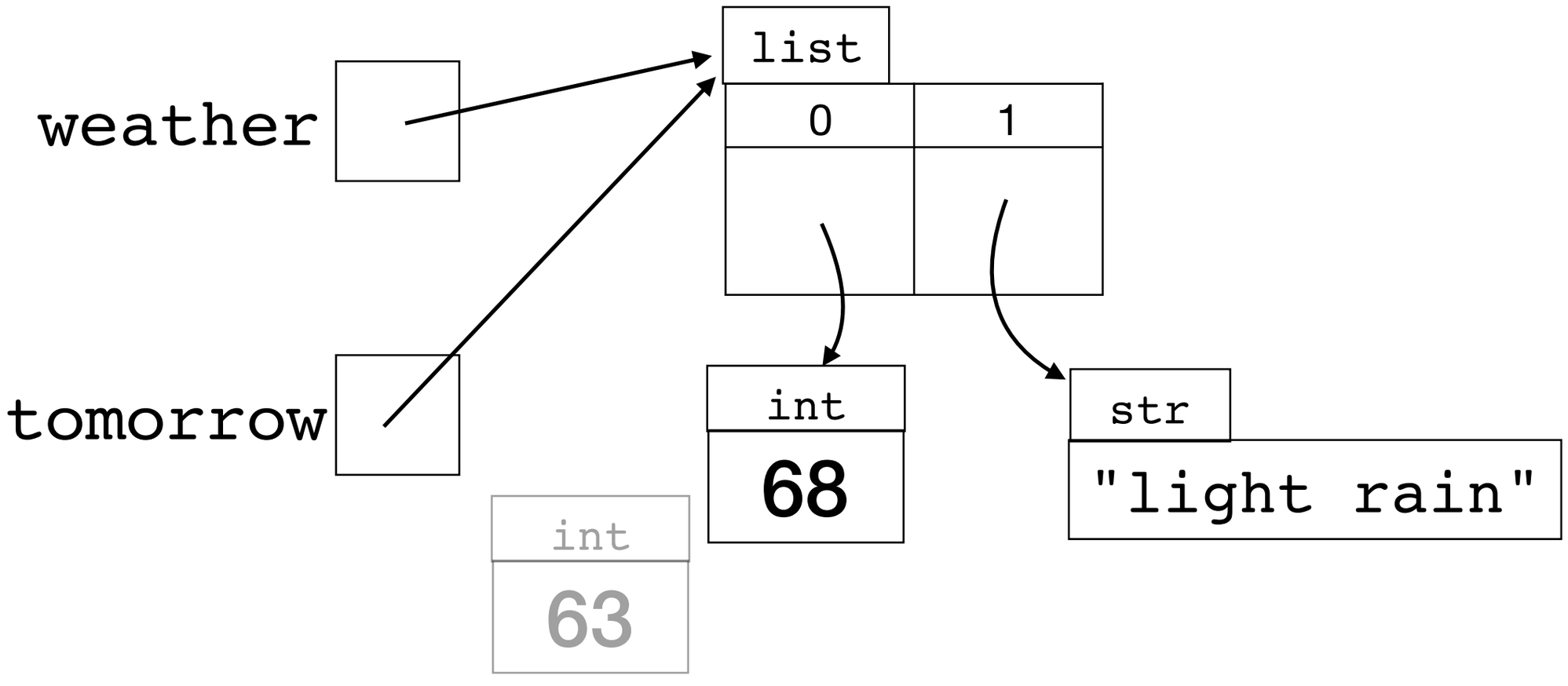
# Monday's worksheet

- Let's write some `copy_list` functions.

# Last time: Mutability

```
weather = [63, "light rain"]
tomorrow = weather
tomorrow[0] = 68
print(weather[0])
```

**State after the above is executed:**



# Implications of Mutability

- Last time: more than one variable (or list element) can contain *references* to the same object.

- Today: variables obey scope (i.e., live in a certain "box".

  - **Objects don't**: they exist outside the "box" framework.

  - **References can cross "box" boundaries.**

# Mutable Objects and Functions

Recall the steps to execute a function call:

1. Evaluate all arguments
2. Draw a local "box" inside the global one
3. Assign argument values to parameter variables in the local box
4. Execute the function body
5. When done, erase the local box
6. Replace the function call with its return value

```
def xtty(x, y):
    """ return x ** y """
    return x ** y

a = 3
b = 2
print(xtty(a, b))
```

# Mutable Objects and Functions

Recall the steps to execute a function call:

1. Evaluate all arguments
2. Draw a local "box" inside the global one
3. Assign argument values to parameter variables in the local box
4. Execute the function body
5. When done, erase the local box
6. Replace the function call with its return value

```
def xtty(x, y):
    """ return x ** y """
    return x ** y

a = 3
b = 2
print(xtty(a, b))
```

# Back to copy_list...

1. Evaluate all arguments
2. Draw a local "box" inside the global one
3. Assign argument values to parameter variables in the local box
4. Execute the function body
5. When done, erase the local box
6. Replace the function call with its return value

```
def copy_list(in_list):
    """ Return a new list
    object containing the
    same elements as in_list.
    """
    copy = []
    for element in in_list:
        copy.append(element)

    return copy
```

# Mutable Objects and Functions

When you pass a list into a function, you're actually passing a *reference* to the list:

# Mutable Objects and Functions

**(or any mutable object!)**

When you pass a list into a function, you're actually passing a *reference* to the list:

# Mutable Objects and Functions

**(or any mutable object!)**

When you pass a list into a function, you're actually passing a *reference* to the list:

```
def z1(a_list):
    a_list[0] = 0

a = [1, 1, 1]
z1(a)
print(a)
```

# Mutable Objects and Functions

When you pass a list into a function, you're actually passing a *reference* to the list:

```
def z1(a_list):
    a_list[0] = 0

a = [1, 1, 1]
z1(a)
print(a)
```

# Mutable Objects and Functions

When you pass a list into a function, you're actually passing a *reference* to the list:

```
def z1(a_list):
    a_list[0] = 0

a = [1, 1, 1]
z1(a)
print(a)
```

`a_list` points to the **same** object as the global variable `a`

# Mutable Objects and Functions

When you pass a list into a function, you're actually passing a *reference* to the list:

```
def z2(a_list):
    a_list = []

a = [1, 1, 1]
z2(a)
print(a)
```

# Mutable Objects and Functions

When you pass a list into a function, you're actually passing a *reference* to the list:

```
def z2(a_list):
    a_list = []

a = [1, 1, 1]
z2(a)
print(a)
```

The local variable `a_list` is reassigned to point to a **new** (different) list

# Mutable Objects and Functions

When you pass a list into a function, you're actually passing a *reference* to the list:

```
def z2(a_list):
    a_list = []

a = [1, 1, 1]
z2(a)
print(a)
```

The local variable `a_list` is reassigned to point to a **new** (different) list

The list referenced by `a` is **unchanged**.

# Mutable Objects and Functions

When you pass a list into a function, you're actually passing a *reference* to the list:

```
def z3(x):
    a_list = [x, x, x]
    return a_list
b = 2
a = z3(b)
print(a)
```

# Mutable Objects and Functions

When you pass a list into a function, you're actually passing a *reference* to the list:

```
def z3(x):
    a_list = [x, x, x]
    return a_list
b = 2
a = z3(b)
print(a)
```

The function creates a **new** list, with the local variable `a_list` referring to it.

# Mutable Objects and Functions

When you pass a list into a function, you're actually passing a *reference* to the list:

```
def z3(x):
    a_list = [x, x, x]
    return a_list
b = 2
a = z3(b)
print(a)
```

The function creates a **new** list, with the local variable `a_list` referring to it.

The **reference** to the list is returned and assigned to `a`.

# Mutable Objects and Functions

```
def z0(y):
    y[0] = 4
    return y

b = [5, 6]
c = z0(b)
print(b[0], c[0])
```

What does this code print?

# Mutable Objects and Functions

```
def z0(y):
    y[0] = 4
    return y

b = [5, 6]
c = z0(b)
print(b[0], c[0])
```

What does this code print?

A. 4 4
B. 4 5
C. 5 4
D. 5 5

# finding a value in a list

```
def find(v, lst):
    """ Return the index of the first
        occurrence of v in lst.
        Return -1 if v is not in the list.
    Precondition: lst is a list. """
```

# finding a value in a *sorted* list

```
def find(v, sorted_lst):
    """ Return the index of the first occurrence
    of v in lst.
    Return -1 if v is not in the list.
    Precondition: lst is a list of things that
    can be compared with the < operator, and is
    in sorted order (i.e. lst[i] <= lst[i+1] for
    all i in range(len(lst)-1) """
```

# Write remove_all(v, lst)

```
def remove_all(v, lst):
    """ Remove ALL occurrences of v from lst.
        Precondition: lst is a list. """
```