CSc 110, Spring 2018

Lecture 23: lists as Parameters

Adapted from slides by Marty Stepp and Stuart Reges

lizclimo.tumblr.com
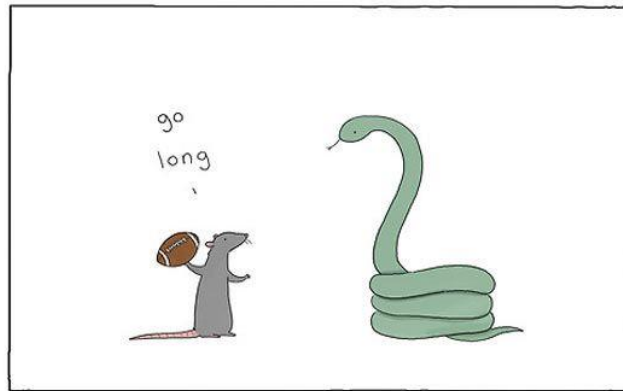
"list mystery" problem

- **traversal**: An examination of each element of an list.

- What element values are stored in the following list?

```
a = [1, 7, 5, 6, 4, 14, 11]
for i in range(0, len(a) - 1):
    if a[i] > a[i + 1]:
        a[i + 1] = a[i + 1] * 2
```

index	0	1	2	3	4	5	6
value	1	7	10	12	8	14	22

List reversal question

- Write code that reverses the elements of a list.

    - For example, if the array initially stores:
      `[11, 42, -5, 27, 0, 89]`

    - Then after your reversal code, it should store:
      `[89, 0, 27, -5, 42, 11]`

        - The code should work for a list of any size.

        - Hint: think about swapping various elements...

Algorithm idea

- Swap pairs of elements from the edges; work inwards:

index	0	1	2	3	4	5
value	89	0	27	-5	42	11

Swapping values

```
def main():
    a = 7
    b = 35

    # swap a with b?
    a = b
    b = a

    print(a, b)
```

- What is wrong with this code? What is its output?

- The red code should be replaced with:

```
    temp = a
    a = b
    b = temp
```

List reverse question 2

- Turn your list reversal code into a `reverse` function.
  - Accept the list of integers to reverse as a parameter.

    ```
    numbers = [11, 42, -5, 27, 0, 89]
    reverse(numbers)
    ```

  - How do we write functions that accept lists as parameters?
  - Will we need to return the new list contents after reversal?
    ...

A `swap` function?

- Does the following `swap` function work? Why or why not?

```
def main():
    a = 7
    b = 35

    # swap a with b?
    swap(a, b)

    print(a, b)


def swap(a, b):
    temp = a
    a = b
    b = temp
```

Mutability

- **Mutability**: The ability to be changed or mutated

  - `int`s, `float`s, `str`s and `bool`s are immutable.
  - lists and objects are mutable

Immutable types

- `int`s, `float`s, `str`s and `bool`s are immutable.
- Modifying the value of one variable does not affect others.

```
x = 5
y = x              # x = 5, y = 5
y = 17             # x = 5, y = 17
x = 8              # x = 8, y = 17
```
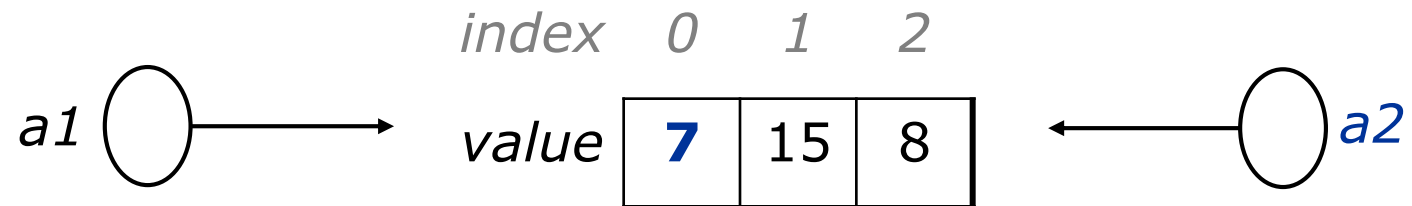
Mutable types

- lists and `DrawingPanel` are mutable.
- Modifying the value of one variable **does** affect others.

```
a1 = [4, 15, 8]
a2 = a1              # refer to same list as a1
a2[0] = 7
print(a1)            # [7, 15, 8]
```

index   0   1   2

a1 ◯ ⟶ value | **7** | 15 | 8 | ⟵ ◯ a2

Mutability and objects

- Lists and objects are mutable. Why?
  - *efficiency.* Copying large objects slows down a program.
  - *sharing.* It's useful to share an object's data among functions.

```
panel1 = DrawingPanel(80, 50)
panel2 = panel1    # same window
panel2.draw_rect(0, 0, 80, 50, "cyan")
```
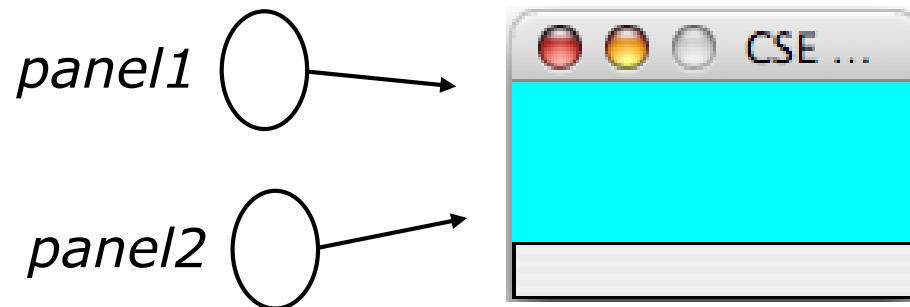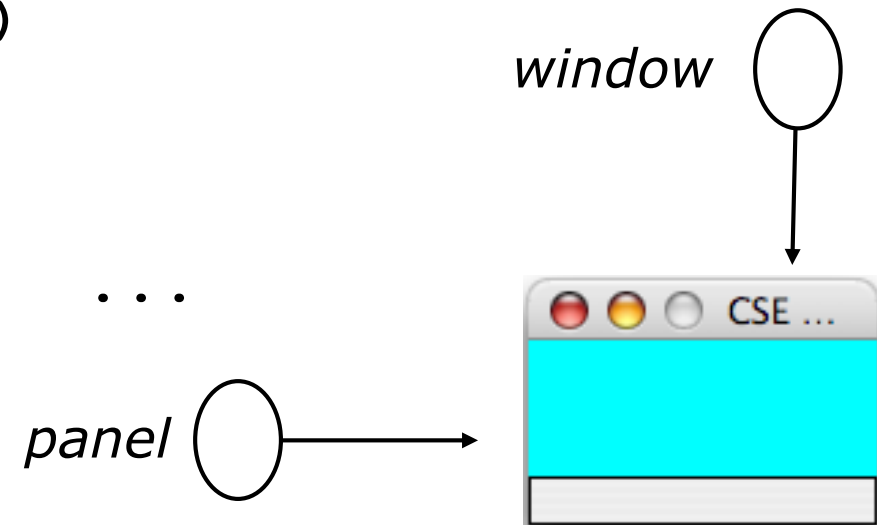
Objects as parameters

- When a mutable object is passed as a parameter the function can change it.
  - If the parameter is modified, it *will* affect the original object.

```
def main():
    window = DrawingPanel(80, 50)
    window.draw_rect(0, 0, 80, 50, "yellow")
    example(window)


def example(panel):
    panel.draw_rect(0, 0, 80, 50, "cyan")      ...
```
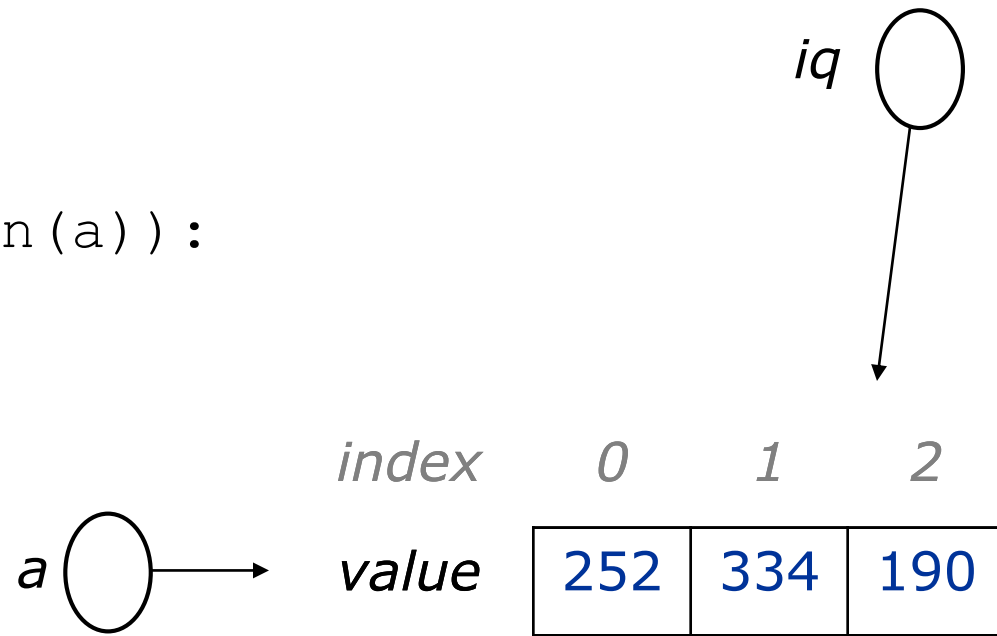
window

panel

Lists as parameters

- Lists are mutable too.
  - Changes made in the function are also seen by the caller.

```
def main():
    iq = [126, 167, 95]
    increase(iq)
    print(iq)

def increase(a):
    for i in range(0, len(a)):
        a[i] = a[i] * 2
```

iq

- Output:
  `[252, 334, 190]`

index	0	1	2

a

value

252	334	190

List reverse question 2

- Turn your list reversal code into a `reverse` function.
  - Accept the list of integers to reverse as a parameter.

  ```
  numbers = [11, 42, -5, 27, 0, 89]
  reverse(numbers)
  ```

- Solution:
  ```
  def reverse(numbers):
      for i in range(0, len(numbers) // 2):
          temp = numbers[i]
          numbers[i] = numbers[len(numbers) - 1 - i]
          numbers[len(numbers) - 1 - i] = temp
  ```

List parameter questions

- Write a function `swap` that accepts a list of integers and two indexes and swaps the elements at those indexes.

```
a1 = [12, 34, 56]
swap(a1, 1, 2)
print(a1)          # [12, 56, 34]
```

- Write a function `swap_all` that accepts two lists of integers as parameters and swaps their entire contents.

  - Assume that the two lists are the same length.

```
a1 = [12, 34, 56]
a2 = [20, 50, 80]
swap_all(a1, a2)
print(a1)   # [20, 50, 80]
print(a2)   # [12, 34, 56]
```

List parameter answers

```python
# Swaps the values at the given two indexes.
def swap(a, i, j):
    temp = a[i]
    a[i] = a[j]
    a[j] = temp


# Swaps the entire contents of a1 with those of a2.
def swap_all(a1, a2):
    for i in range(0, len(a1)):
        temp = a1[i]
        a1[i] = a2[i]
        a2[i] = temp
```

List return question

- Write a function `merge` that accepts two lists of integers and returns a new list containing all elements of the first list followed by all elements of the second.

```
a1 = [12, 34, 56]
a2 = [7, 8, 9, 10]

a3 = merge(a1, a2)
print(a3)
# [12, 34, 56, 7, 8, 9, 10]
```

- Write a function `merge3` that merges 3 lists similarly.

```
a1 = {12, 34, 56]
a2 = {7, 8, 9, 10]
a3 = {444, 222, -1]

a4 = merge3(a1, a2, a3)
print(a4)
# [12, 34, 56, 7, 8, 9, 10, 444, 222, -1]
```

List return answer 1

```
# Returns a new list containing all elements of a1
# followed by all elements of a2.
def merge(a1, a2):
    result = [0] * (len(a1) + len(a2))

    for i in range(0, len(a1)):
        result[i] = a1[i]
    for i in range(0, len(a2)):
        result[len(a1) + i] = a2[i]

    return result
```

List return answer 2

```python
# Returns a new list containing all elements of a1,a2,a3.
def merge3(a1, a2, a3):
    a4 = [0] * (len(a1) + len(a2) + len(a3)

    for i in range(0, len(a1)):
        a4[i] = a1[i]
    for i in range(0, len(a2)):
        a4[len(a1) + i] = a2[i]
    for i in range(0, len(a3)):
        a4[len(a1) + len(a2) + i] = a3[i]

    return a4


# Shorter version that calls merge.
def merge3(a1, a2, a3):
    return merge(merge(a1, a2), a3)
```