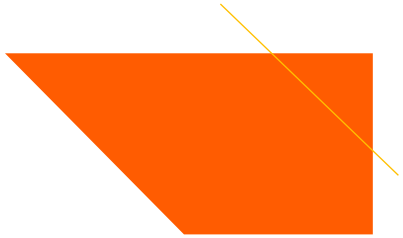
# Introduction to Problem Solving in Python

COSI 10A

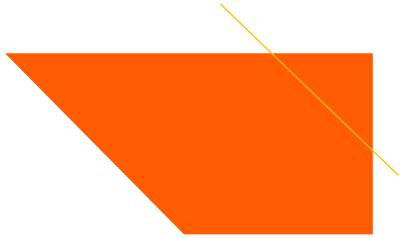


# Class objectives

- More on Lists
- Reference semantics (Section 7.3)

# List functions

| Function | Description |
|---|---|
| `append(x)` | Add an item to the end of the list. Equivalent to `a[len(a):] = [x]`. |
| `extend(L)` | Extend the list by appending all the items in the given list. Equivalent to `a[len(a):] = L` |
| `insert(i, x)` | Inserts an item at a given position. i is the index of the element before which to insert, so `a.insert(0, x)` inserts at the front of the list. |
| `remove(x)` | Removes the first item from the list whose value is *x*. Errs if there is no such item. |
| `pop(i)` | Removes the item at the given position in the list, and returns it. `a.pop()` removes and returns the last item in the list. |
| `clear()` | Remove all items from the list. |
| `index(x)` | Returns the index in the list of the first item whose value is *x*. Errs if there is no such item. |
| `count(x)` | Returns the number of times *x* appears in the list. |
| `sort()` | Sort the items of the list |
| `reverse()` | Reverses the elements of the list |
| `copy()` | Return a copy of the list. |



# List reversal

- 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...

# Flawed algorithm

- What's wrong with this code?

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

# reverse the list
for i in range(0, len(numbers)):
    temp = numbers[i]
    numbers[i] = numbers[len(numbers) - 1 - i]
    numbers[len(numbers) - 1 - i] = temp
```

- The loop goes too far and un-reverses the array!  Fixed version:

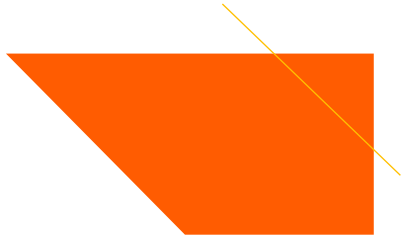
```
for i in range(0, len(numbers) // 2):
    temp = numbers[i]
    numbers[i] = numbers[len(numbers) - 1 - i]
    numbers[len(numbers) - 1 - i] = temp
```

# List reverse question

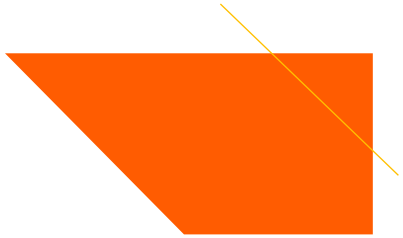
- Turn your list reversal code into a reverse function
  - Accept the list of integers to reverse as a parameter

```
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?
- …



# Mutability



# Mutability

- **Mutability is** 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

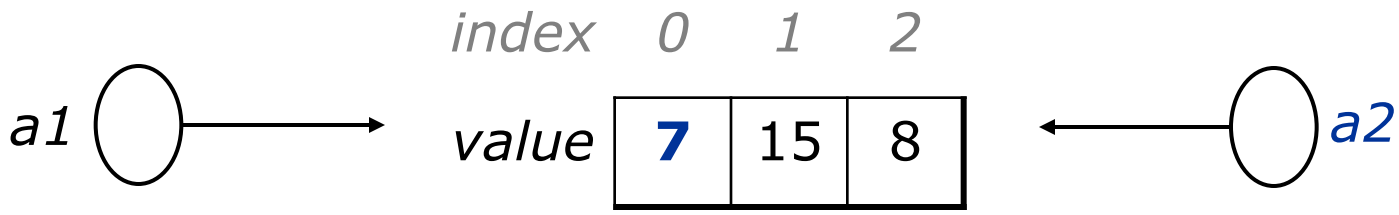
- `int`s, `float`s, `str`s and `bool`s are immutable.
- Modifying the value of one variable does not affect others.

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

# Mutable types

- Lists 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]
```



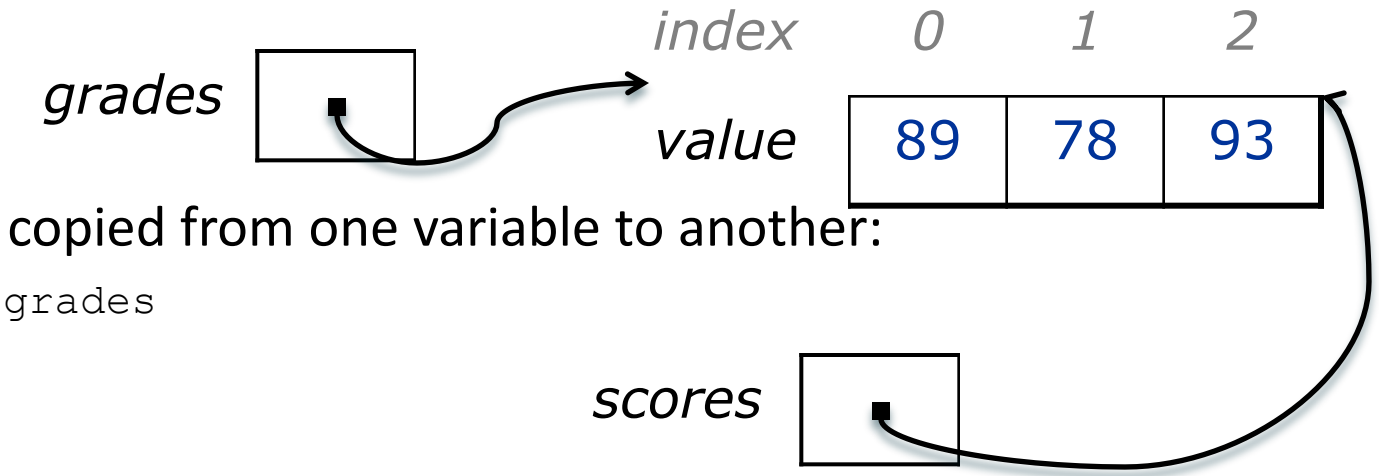
# Value/Reference Semantics

- Variables of type `int`, `float`, `boolean`, store values directly:


- Values are copied from one variable to another:

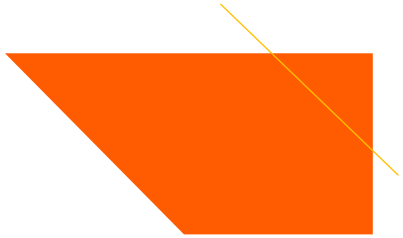
```
cats = age
```


- Variables of object types store references to memory:




- References are copied from one variable to another:

```
scores = grades
```



# 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

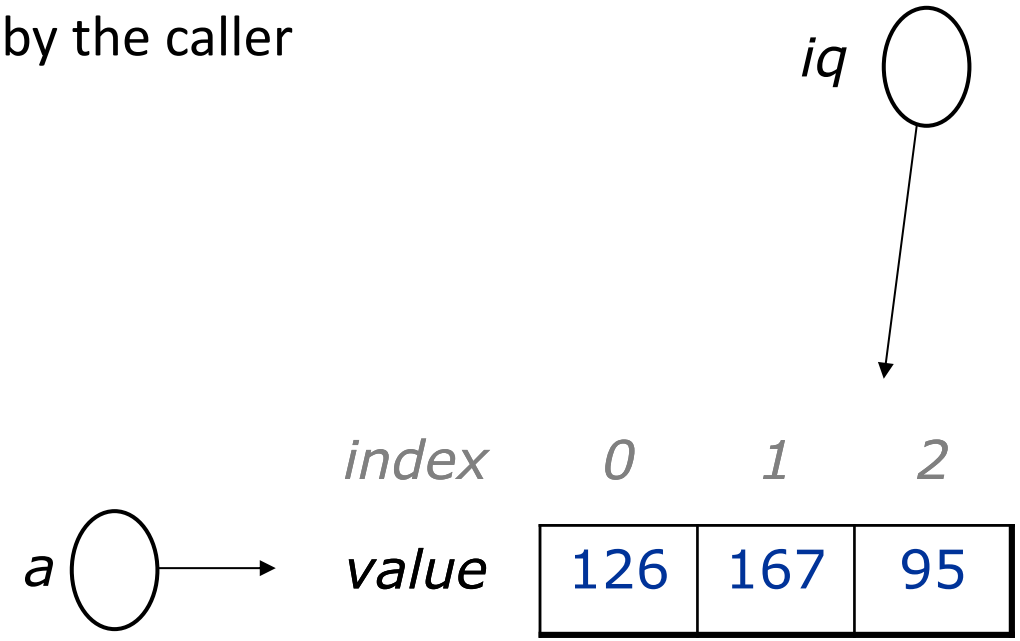
# Objects (lists) as parameters

- 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.

- 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
```



# Objects (lists) as parameters

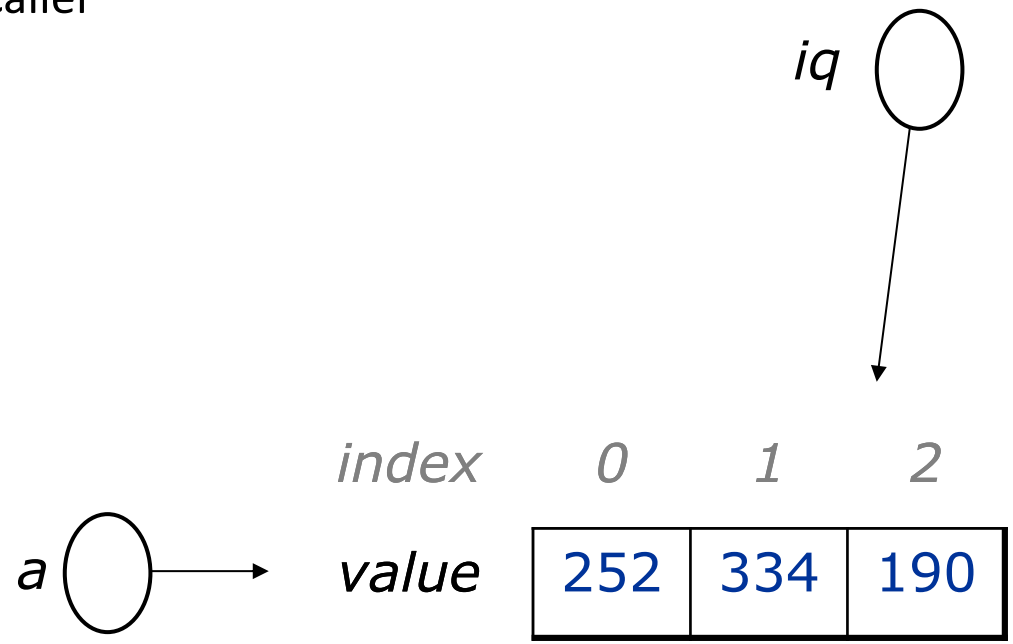
- When a mutable object is passed as a parameter the function can change it.
  - If the parameter is modified, it *will* affect the original object.

- 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
```

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



# 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
```

# 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)
```

# 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

```
# 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
```