Mutable objects behave differently than immutable objects when passed as arguments to a function.
Review:

When a function runs, it defines a new namespace. The names in the function’s namespace are only available to the function.

Passing arguments by reference: the first argument passes its namespace reference to the first parameter, second to second, and so on.

What gets passed?
First: passing an *immutable* object.  
(Review)

```python
my_int = 25
my_function(my_int)
print(my_int)
def my_function(an_int):
    print(an_int)
```
Both namespaces now refer to the same object: the reference got passed.

What happens if the function changes that variable?

\[
\begin{align*}
\text{my_int} &= 25 \\
\text{my_function(my_int)} \\
\text{print(my_int)} \\
\text{def my_function(an_int):} \\
\quad \text{an_int} &= 37 \\
\quad \text{print(an_int)} \\
\end{align*}
\]

- \text{an_int} references an \textit{immutable} object, which cannot be changed.
- So a new int object is created for \text{an_int} to reference.
The object (int) is immutable: it wasn’t changed.
The function namespace simply updated its reference to a new object.
The reference in the calling program was unaffected.

Passing a *mutable* object: a list.
Local Objects

A function has its own namespace ("local"), so
- a parameter is in the function’s namespace so any use outside the function is an error.
- any variable assigned in the function is in the function’s namespace so it is not available outside of the function.
a = 25
my_function(a)
print(a)

def my_function(b):
    a = 37
    print(a, b)

main namespace

Python objects

my_function namespace

Reexamine examples in www.pythontutor.com
Car Talk Puzzler

The Palindromic Odometer

Driving along, Terry notices that the last four digits on the odometer are palindromic. A mile later, the last five digits are palindromic. A mile later, the middle four digits are palindromic. One mile after that, all six are palindromic. What was the odometer reading when Terry first looked at it?
making a main function

- it is common to create the main function that starts the whole program running
- now when we run our file, it defines the main function which we must call manually.

extensive use of continue

- checks for ‘non-palindromes’ under the required circumstances and continues
- makes the process more efficient.
checking time

import time
start = time.time()
... do stuff ...
end = time.time()

print('It took:', end-start, 'seconds')

refactoring

- what if you want to check a different approach to palindrome?
- It is easy to refactor this program. Provide a new function with a different definition to see the effect.
- functions make refactoring easier
Lists are a big deal!

- The use of lists in Python is a major part of its power.
- Python provides support to make common list tasks easier.
Constructing a list using *list comprehension*

\[
[\ n \quad \text{for } n \quad \text{in} \quad \text{range}(1,5) \ ]
\]

returns \[1,2,3,4\]

what we iterate through \[n\]

what we collect \[1\]
delimit with \([\ ]\)

modifying what we collect

\[
[\ n**2 \quad \text{for } n \quad \text{in} \quad \text{range}(1,6) \ ]
\]

what we collect \[n**2\]

what we iterate through \[1,6\]

returns \[1, 4, 9, 16, 25\]
multiple collects

\[
[x+y \quad \text{for } x \text{ in } \text{range}(1,4) \quad \text{for } y \text{ in } \text{range}(1,4)]
\]

It is as if we had done the following:

```python
myList = []
for x in range(1,4):
    for y in range(1,4):
        myList.append(x+y)
```

returns \([2,3,4,3,4,5,4,5,6]\]

modifying what gets collected

```python
[c \quad \text{for } c \text{ in } "Hi There Mom" \quad \text{if } c\text{.isupper()}]
```

what we iterate through

what we keep

returns ['H', 'T', 'M']
Exercise: list comprehension

Use list comprehension in defining functions:

- odd(n): returns the list of odd numbers from 1 to n, inclusive; e.g., odd(11) returns [1, 3, 5, 7, 9, 11]

- squares(n): returns the first n perfect squares; e.g., squares(5) returns [1, 4, 9, 16, 25]

- sq(n): returns the perfect squares that are between 1 and n, inclusive; e.g., sq(5) returns [1, 4]

- Do the same using
  - sets
  - dictionaries