ITP 115 – Programming in Python

Variable References
and
Function Parameters

Discussion of Variable References

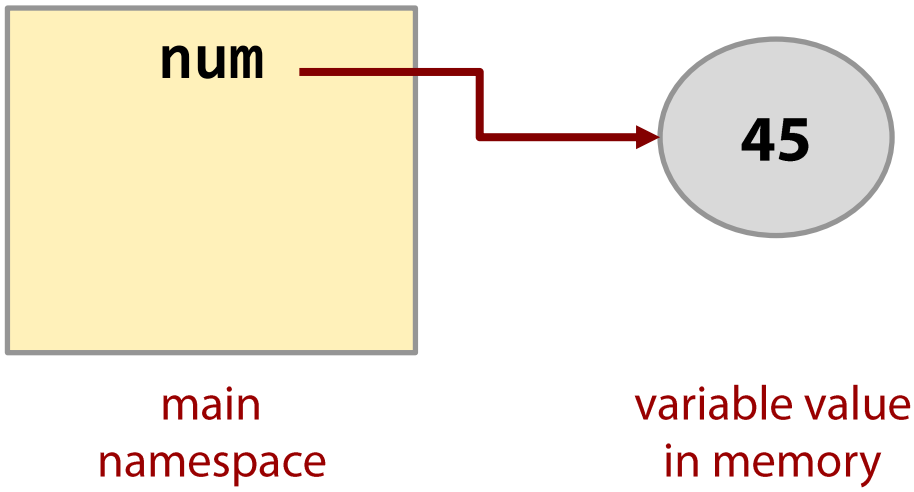
- A variable does not *actually* store the data you assign to it
- A variable instead stores a **reference** to the computer's memory where the data is stored
- The **reference** exists in the current namespace

Discussion of Variable References

- When you see

  ```
  num = 45
  ```

  you should imagine…



Immutable Objects

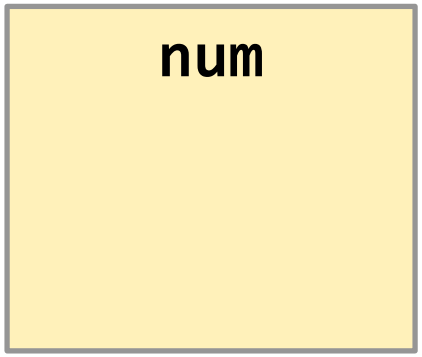
- Immutable objects can't be changed
- When you re-assign an immutable object, it creates a new one

- Immutable objects are
  - *strings*
  - *ints*
  - *floats*
  - *tuples*

Immutable Objects

- What happens in the following?

```
num = 45
num = 81
```



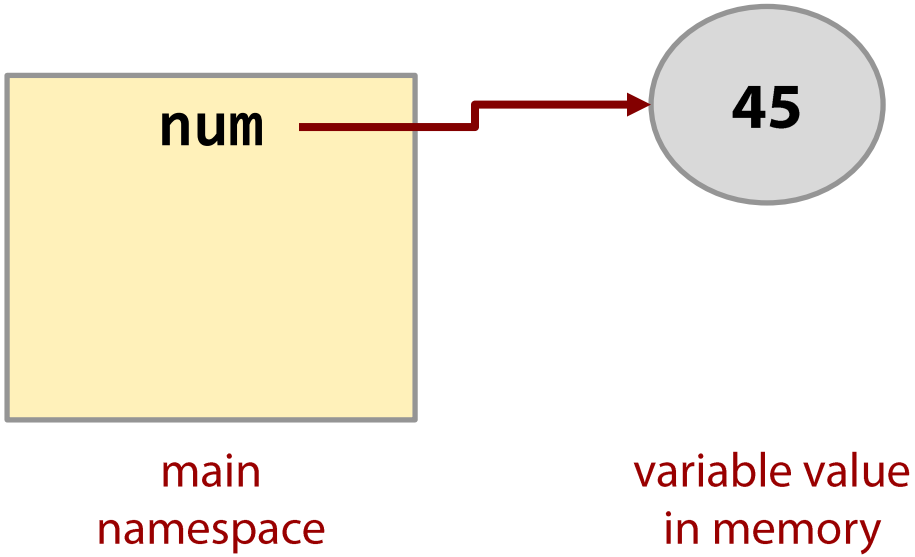
main namespace

variable value in memory

Immutable Objects

- What happens in the following?

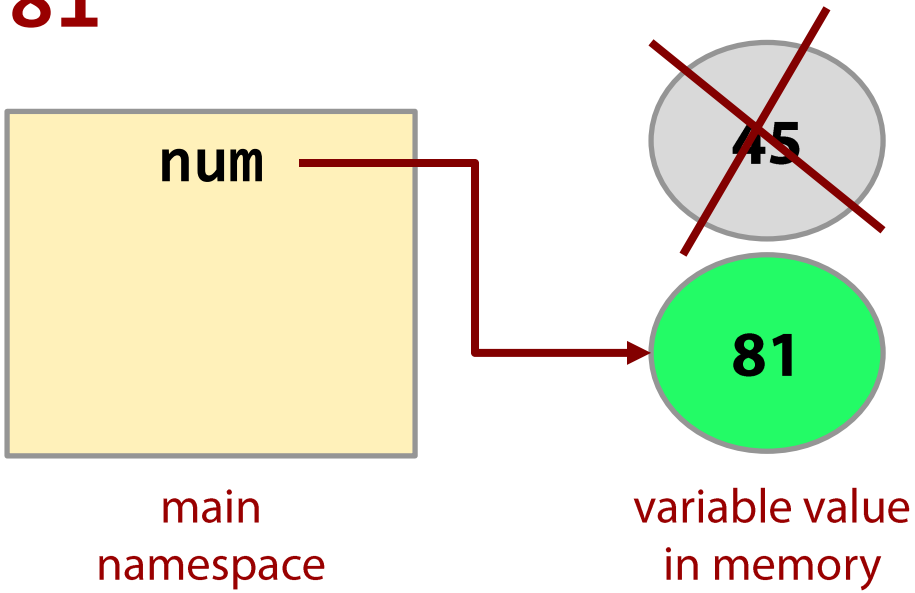
```
num = 45
num = 81
```



Immutable Objects

- What happens in the following?

```
num = 45
num = 81
```

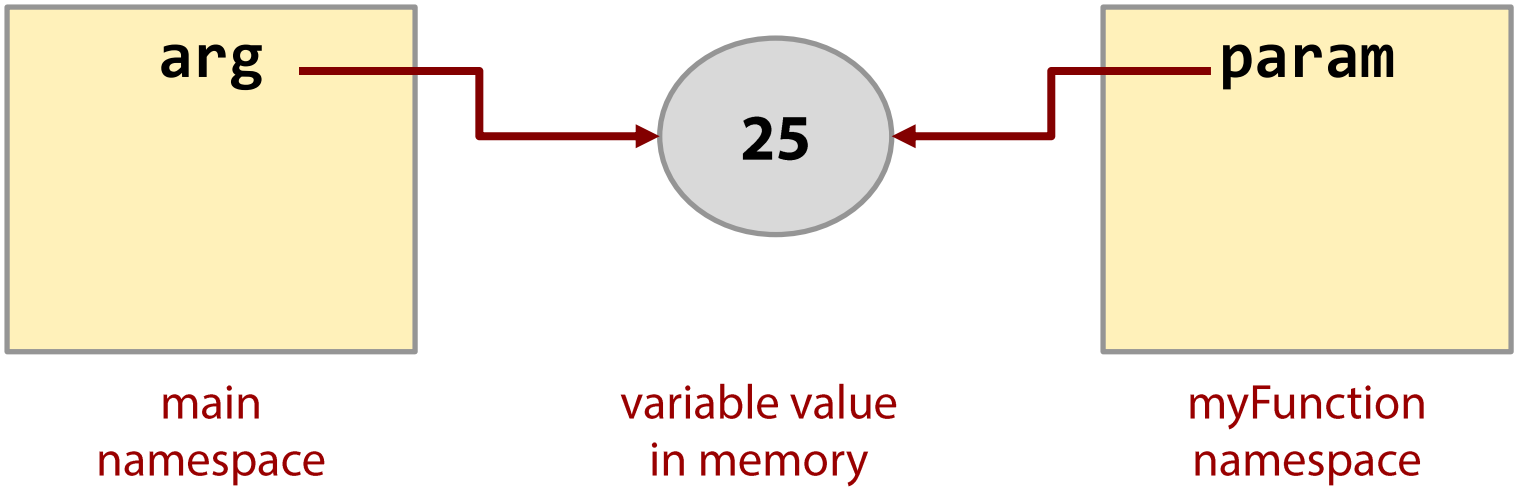


Passing Immutable Objects

- What happens we pass immutable objects to a function?
    - A *copy* of the reference is made
        - But both the original and the copy point to same data in memory
    - Since the object is immutable, any changes to its value *inside* the function **will not affect** the original variable

```
arg = 25
myFunction(arg)
print(arg)
```

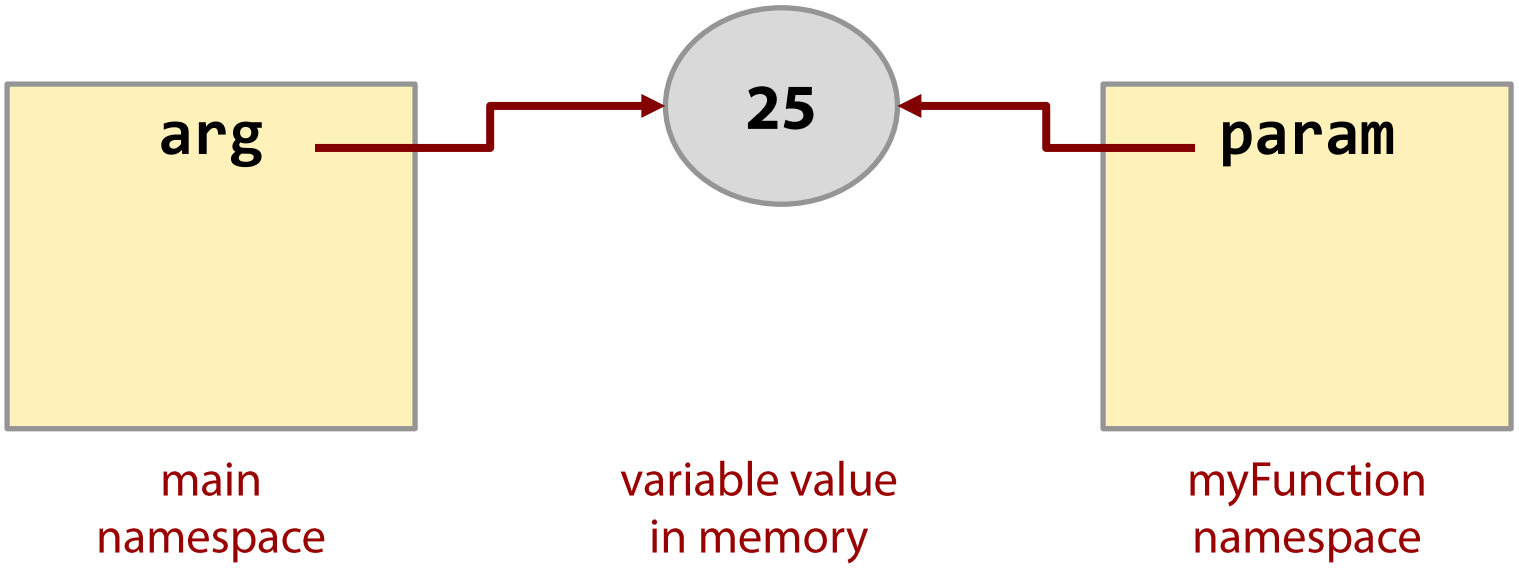
```
def myFunction(param):
    print(param)
```



What if an immutable object changes in a function?

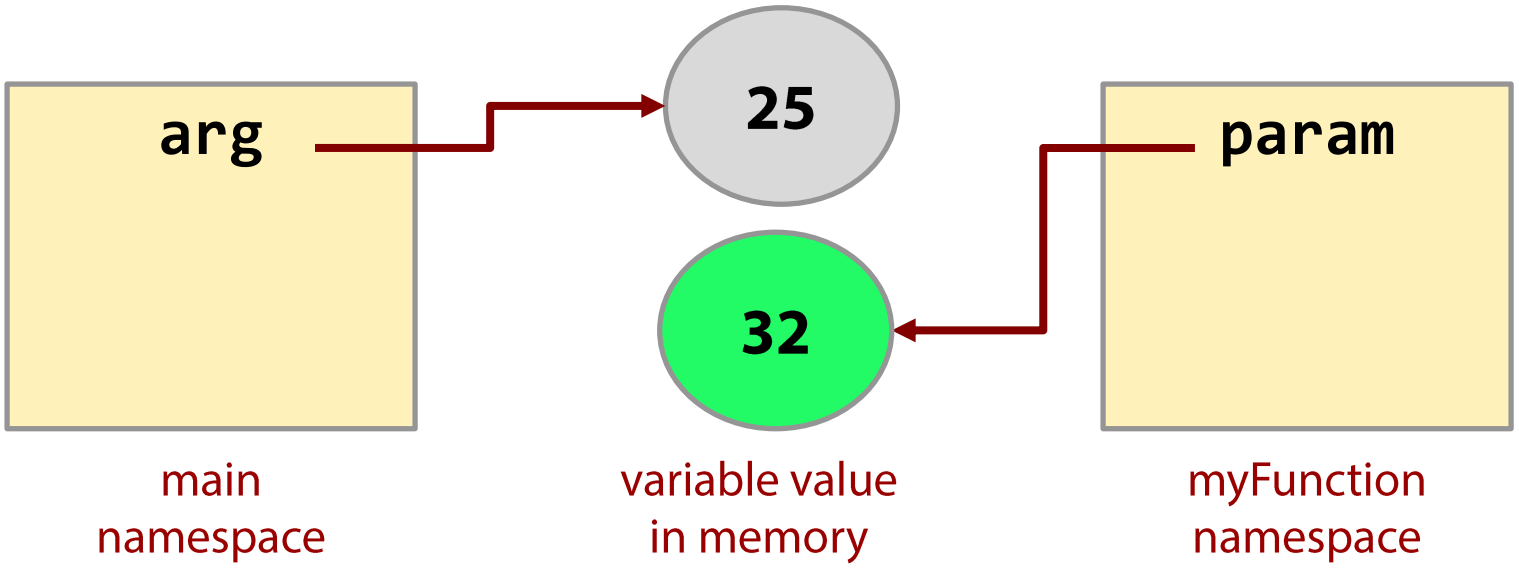
```
arg = 25
myFunction(arg)
print(arg)
```

```
def myFunction(param):
    param = 32
    print(param)
```



```
arg = 25
myFunction(arg)
print(arg)
```

```
def myFunction(param):
    param = 32
    print(param)
```



Mutable Objects

- Mutable objects **can** be changed
  - **`append, del, remove, []`**

- However, if you re-assign an mutable object, it **still creates a new one**

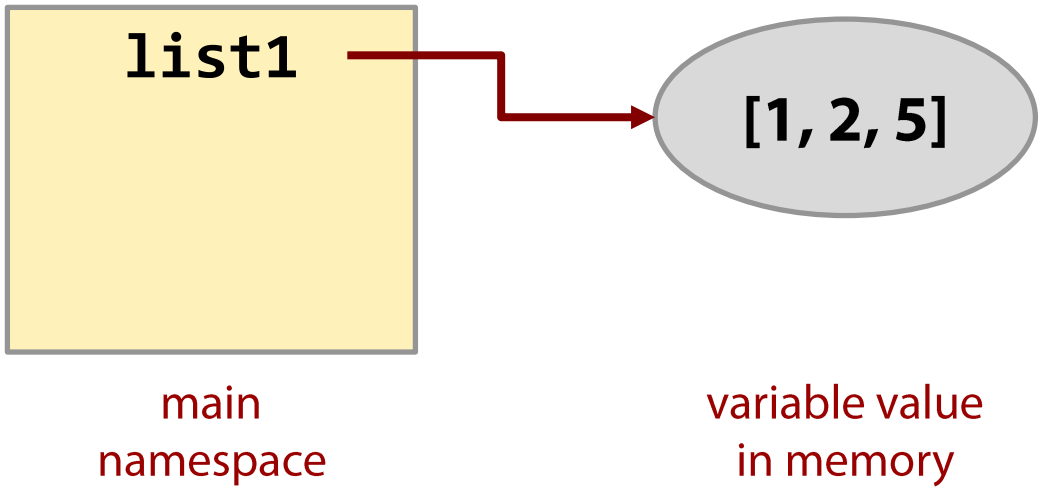
- Mutable objects
  - *lists*
  - *dictionaries (later)*

Mutable Objects

- When you see

  ```
  list1 = [1, 2, 5]
  ```

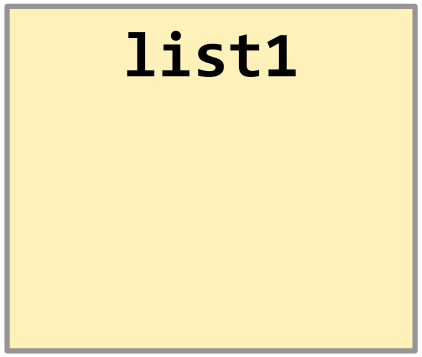
  you should imagine…



Mutable Objects

- What happens in the following?

```
list1 = [1, 2, 5]
list1[0] = 9
```

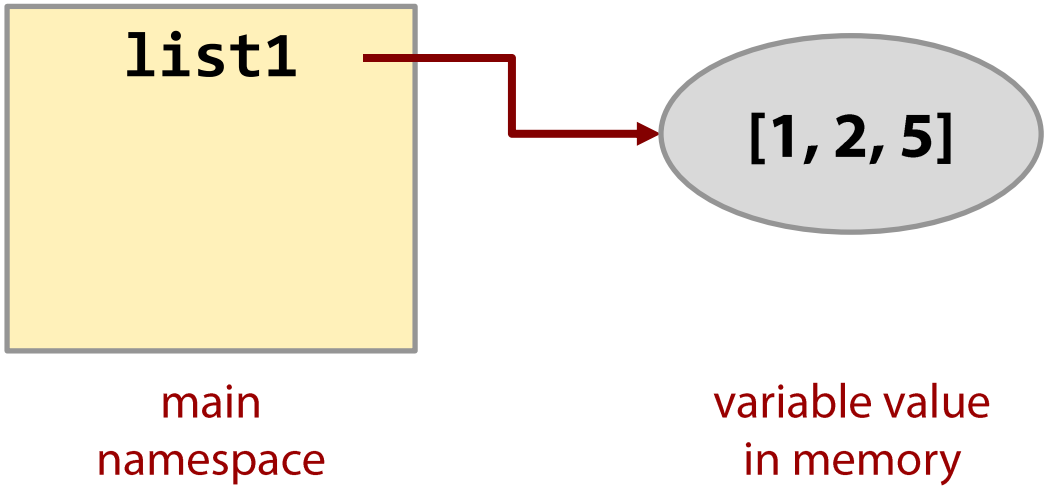


main
namespace

Mutable Objects

- What happens in the following?

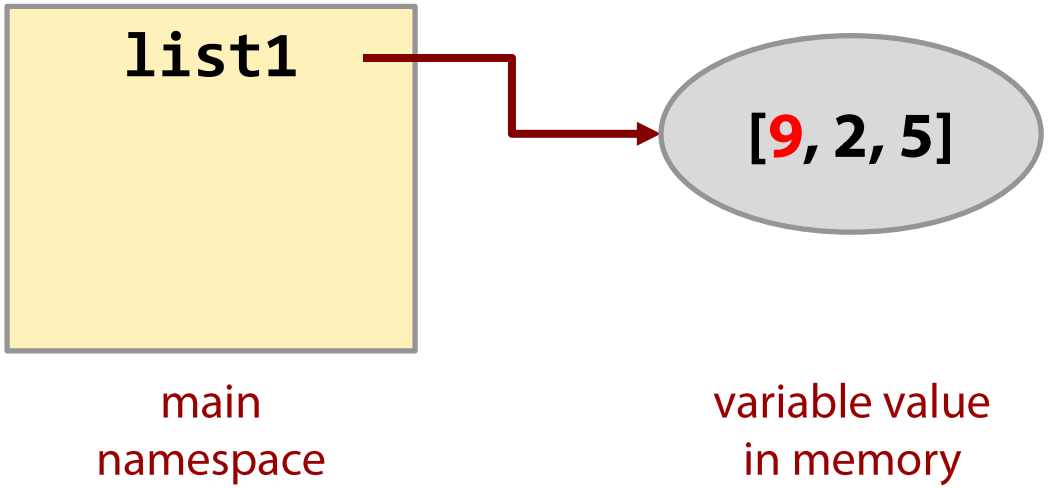
```
list1 = [1, 2, 5]
list1[0] = 9
```



Mutable Objects

- What happens in the following?

```
list1 = [1, 2, 5]
list1[0] = 9
```

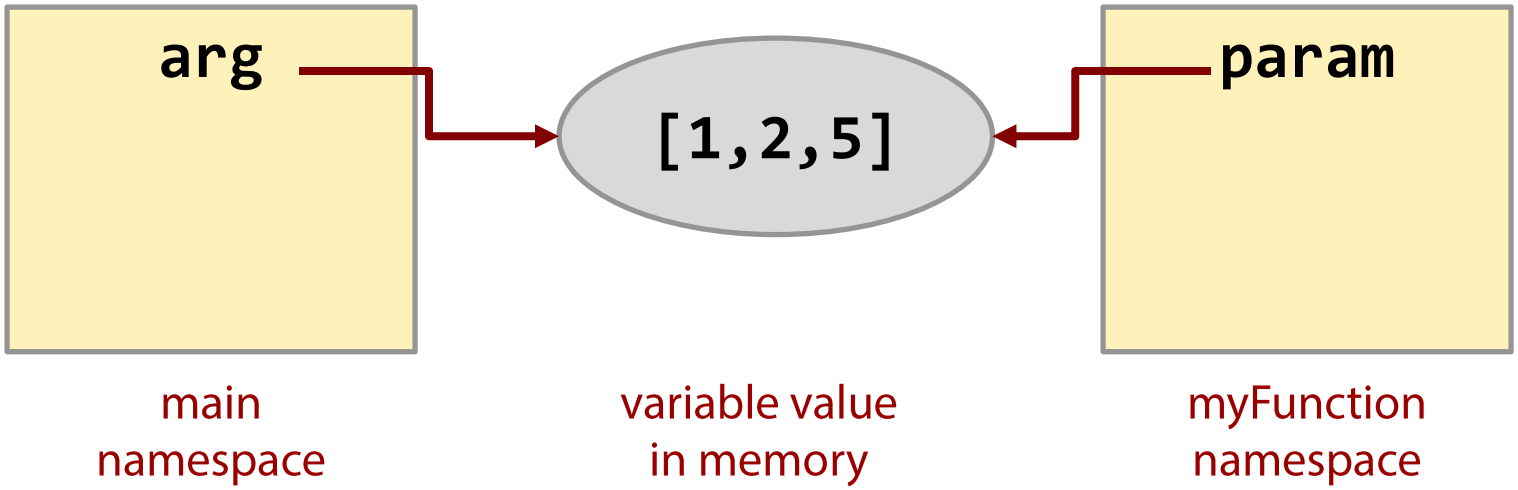


Passing Mutable Objects

- What happens we pass mutable objects to a function?
    - A *copy* of the reference is made
        - But the original and the copy point to same data in memory
    - However, since object is mutable, any changes to its value *inside* the function **will affect** the original variable

```
arg = [1,2,5]
myFunction(arg)
print(arg)
```

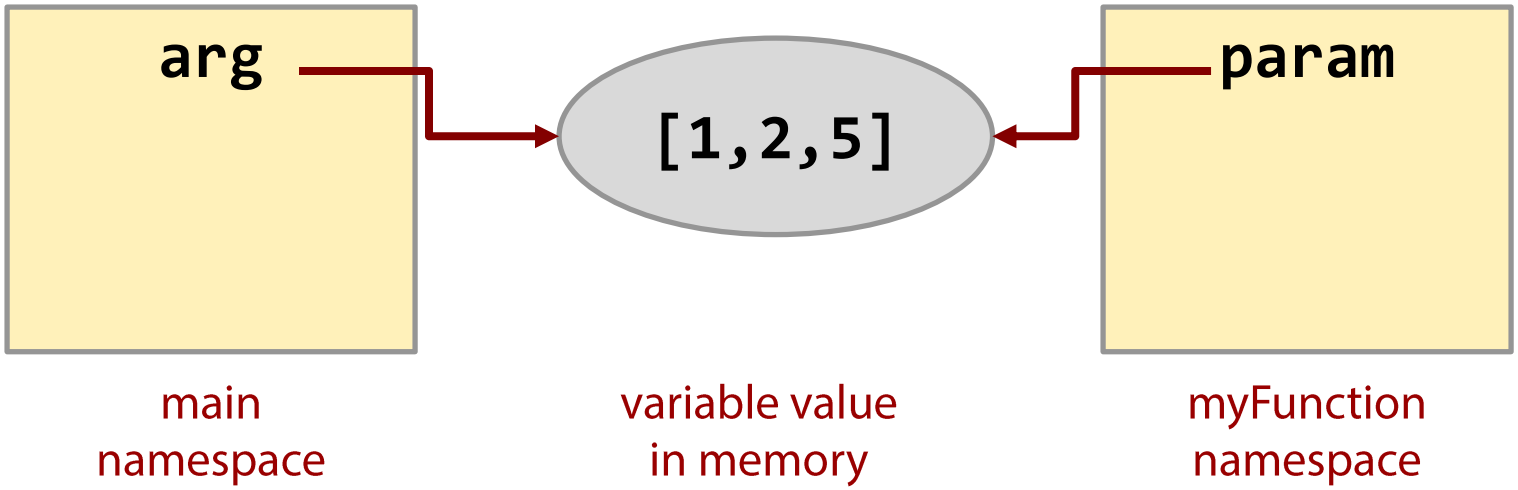
```
def myFunction(param):
    print(param)
```



What if an mutable object changes in a function?

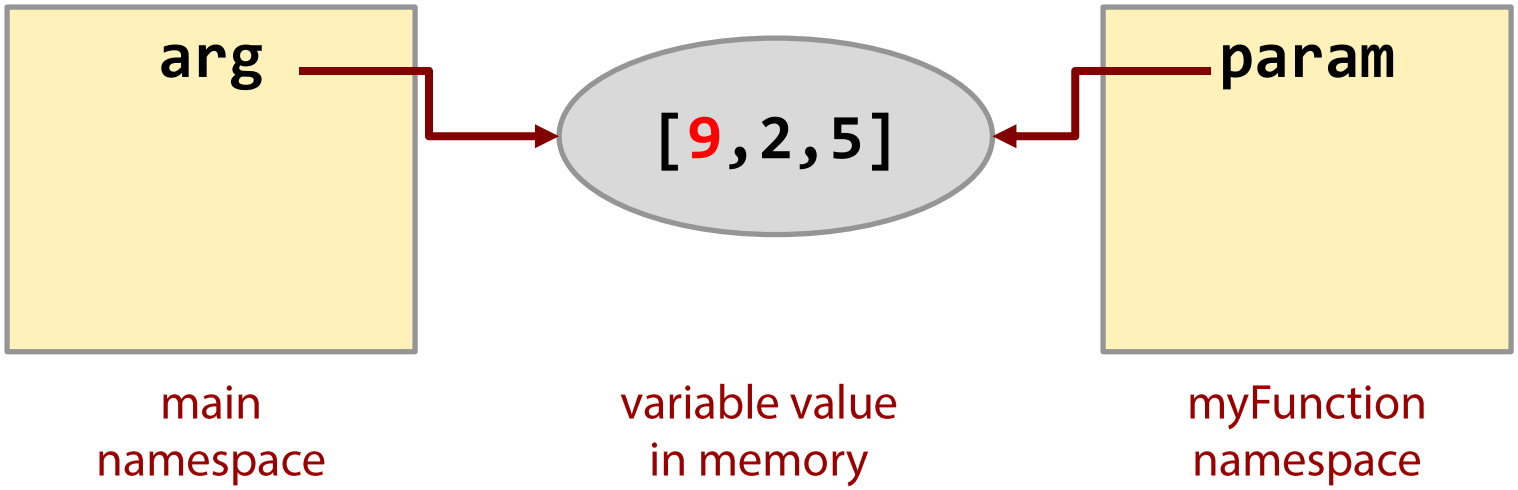
```
arg = [1,2,5]
myFunction(arg)
print(arg)
```

```
def myFunction(param):
    param[0] = 9
    print(param)
```



```
arg = [1,2,5]
myFunction(arg)
print(arg)
```

```
def myFunction(param):
    param[0] = 9
    print(param)
```



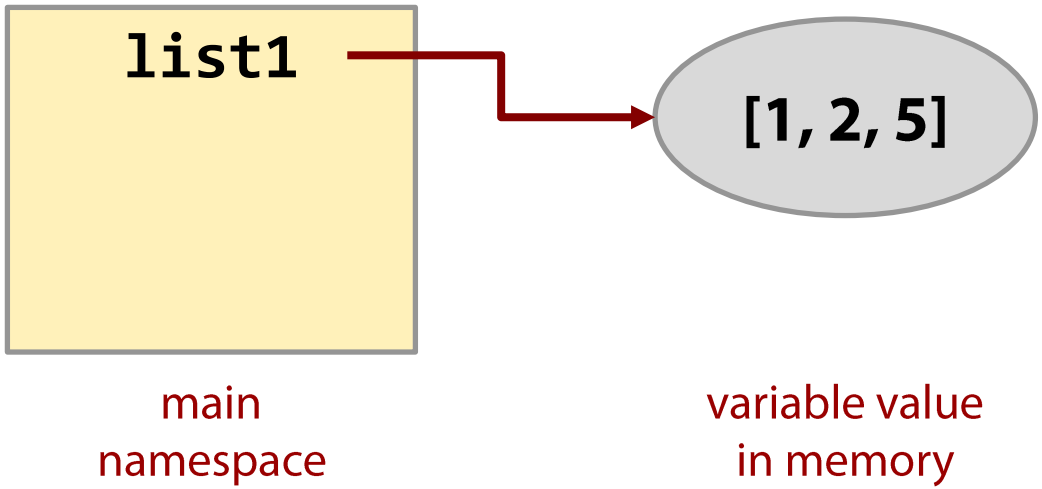
But wait!

Mutable Objects

- When you see

  ```
  list1 = [1, 2, 5]
  ```

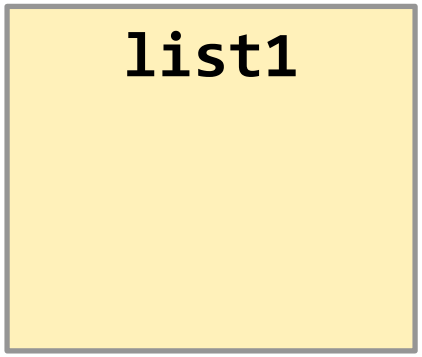
  you should imagine…



Mutable Objects

- What happens in the following?

```
list1 = [1, 2, 5]
list1 = [4, 3]
```

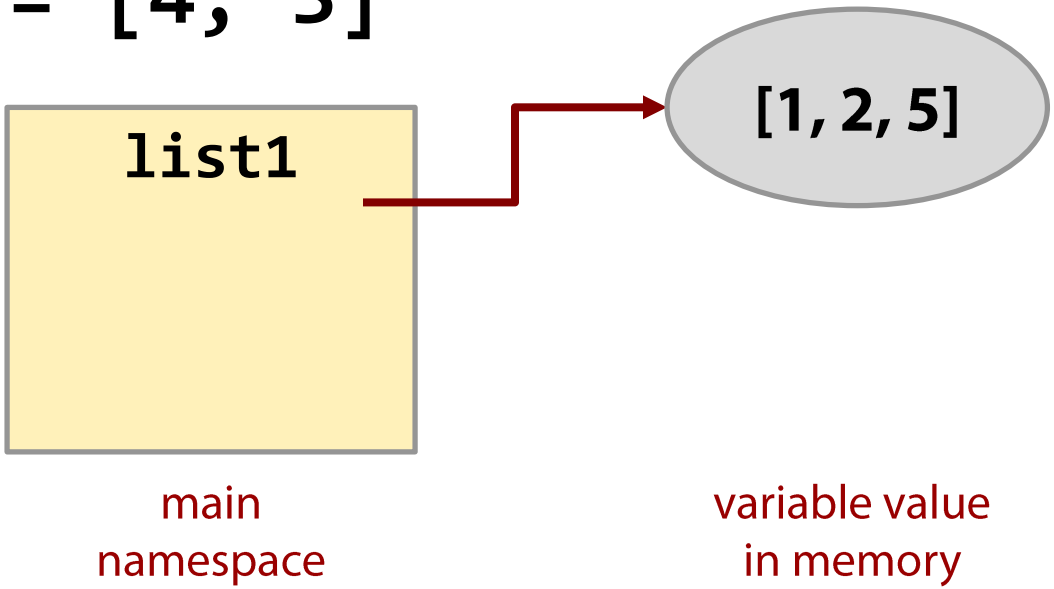


main
namespace

Mutable Objects

- What happens in the following?

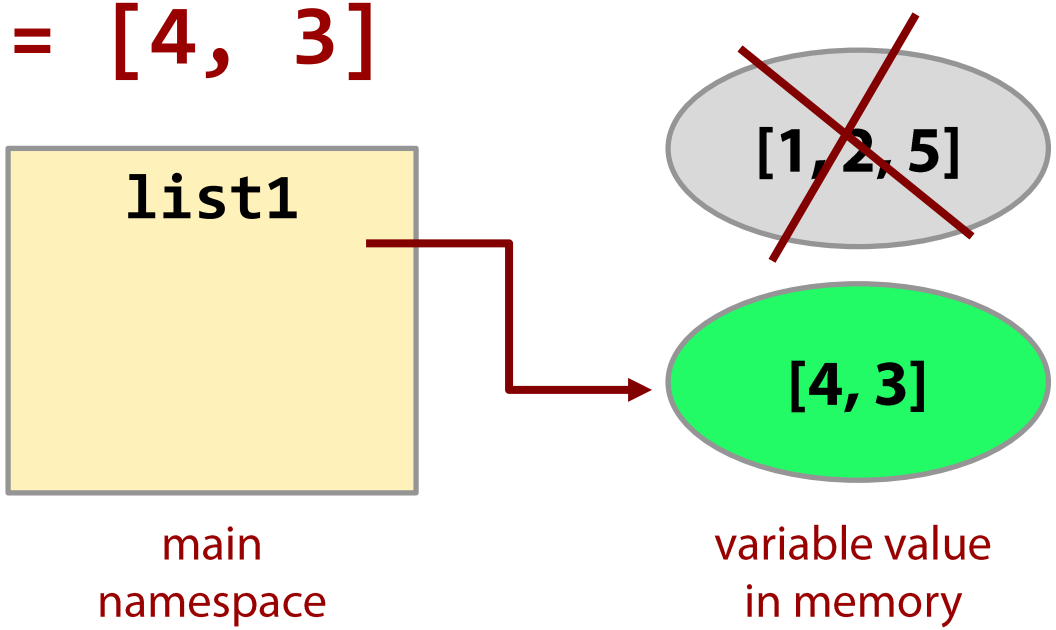
```
list1 = [1, 2, 5]
list1 = [4, 3]
```



Mutable Objects

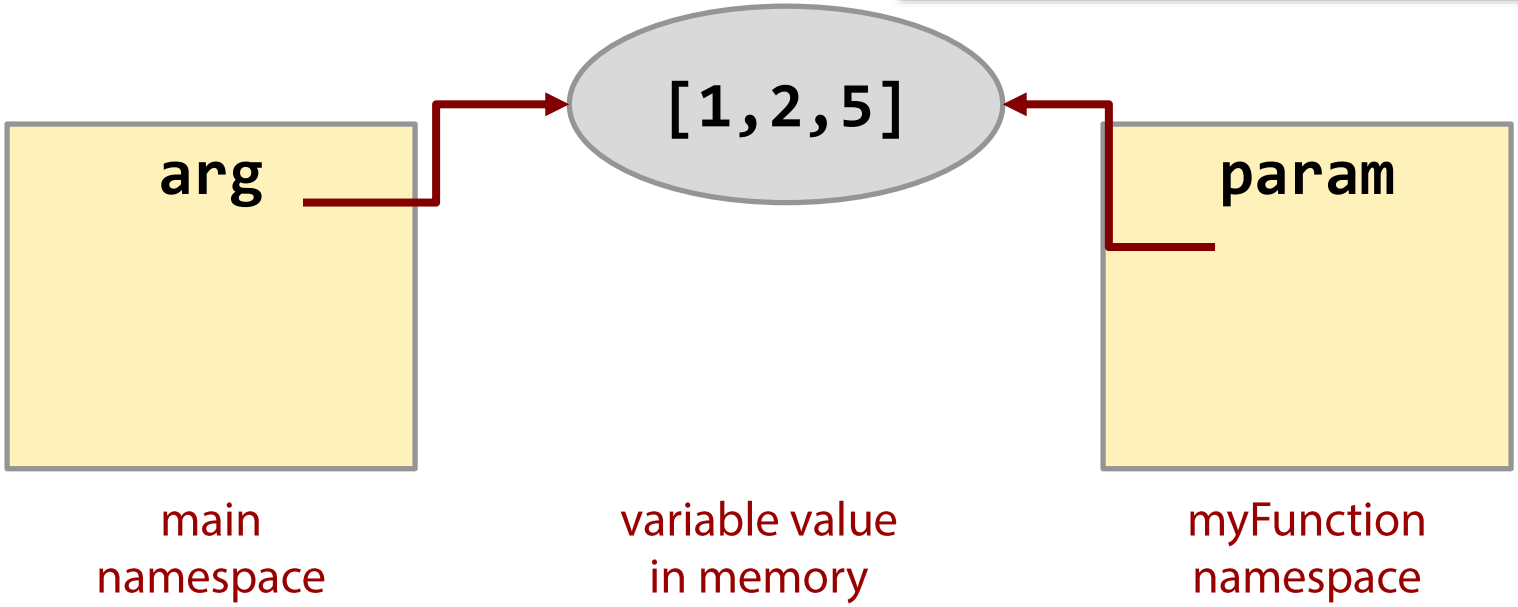
- What happens in the following?

```
list1 = [1, 2, 5]
list1 = [4, 3]
```



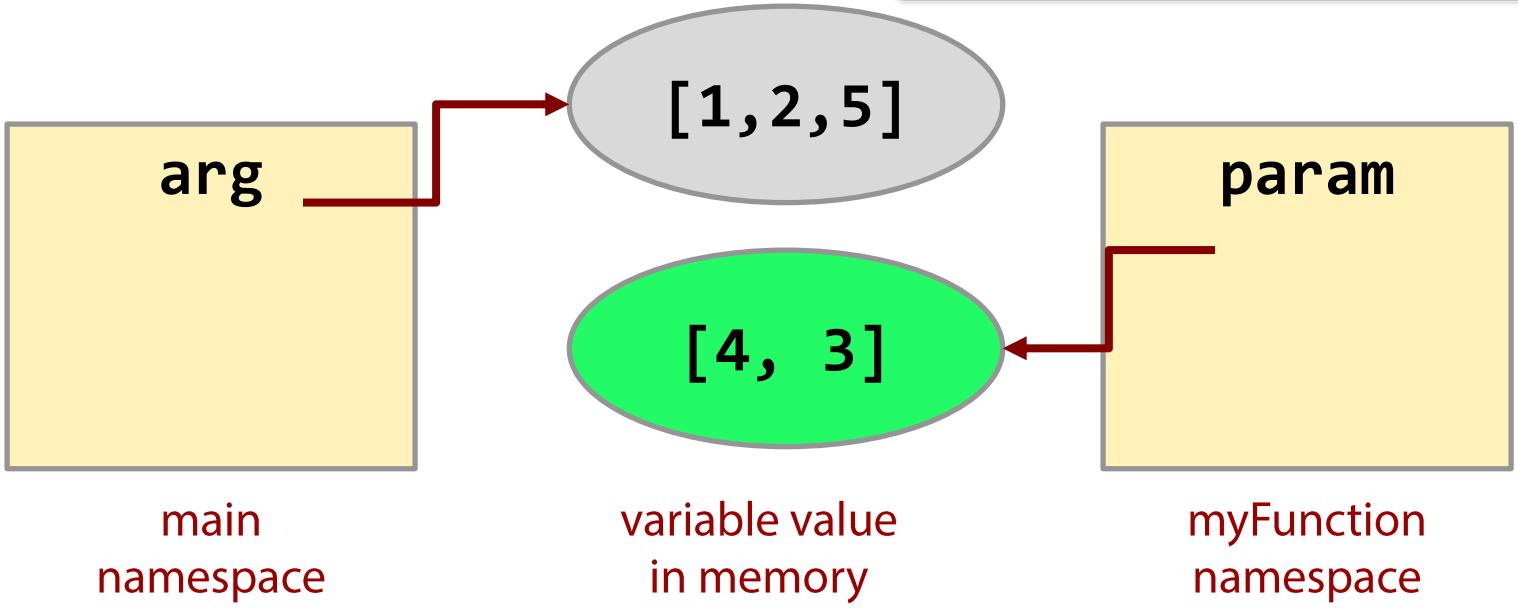
```
arg = [1,2,5]
myFunction(arg)
print(arg)
```

```
def myFunction(param):
    param[0] = 9
    print(param)
```



```
arg = [1,2,5]
myFunction(arg)
print(arg)
```

```
def myFunction(param):
    param = [4, 3]
    print(param)
```



Aren't List Mutable?

- Every time you use assignment, Python **creates a new variable** (mutable or immutable)

```
num = 45
num = 81
```

```
list1 = [1, 2, 5]
list1 = [4, 3]
```

- However, with mutable objects, you can modify them without creating a new variable

```
list1 = [1, 2, 5]
list1[0] = 9
```

```
list2 = [4, 7]
list2.append(8)
```

When you pass variable to a function

- Immutable variables are **not** affected by any changes made within the function

- Mutable variables **may** be affected by changes in the function
  - Assignment (=) will **not** affect the original variables
  - Modifying operations *([ ], **append**, **del**, etc.)* **do** affect the original variable