Lists

Brahm Capoor
Code in Place 2023, Stanford University

Variables have limitations

```
def add_two_numbers(num1, num2):
    return num1 + num2
```

Variables have limitations

```
def add_two_numbers(num1, num2):
    return num1 + num2


def add_three_numbers(num1, num2, num3):
    return num1 + num2 + num3
```

Variables have limitations

```
def add_two_numbers(num1, num2):
    return num1 + num2


def add_three_numbers(num1, num2, num3):
    return num1 + num2 + num3


def add_four_numbers(num1, num2, num3, num4):
    return num1 + num2 + num3 + num4
```

By the end of this lesson...

```
def add_many_numbers(num1, num2, ..., last_num):
    return num1 + num2 + ... last_num
```

This isn't valid code, but we'll learn how to write a function that can do the same thing!

Learning Goals

1. Understand what a data structure is
2. Writing code to use lists
3. Understanding lists as parameters

Data Structures & Our First List

What is a data structure?

So far, all our variables in the class have been contained **exactly one value**

```
my_num = 42
```

Data structures allow us to store **multiple values** in one variable

Our First Lists

```
list_of_nums = [42, 37, 28, 2]

names = ["Brahm", "Waymond", "Gwen"]

empty_list = []
```

Lists begin and end with brackets and consist of **elements**

Elements are separated by **commas**

Accessing Elements in a list

```
names = ["Brahm", "Waymond", "Gwen"]
```

0 1 2

The position of an element in a list is called its **index**. List indices start from 0.

Accessing Elements in a list

```
names = ["Brahm", "Waymond", "Gwen"]
```

0 1 2

```
first_element = names[0]  # "Brahm"
```

Accessing Elements in a list

```
names = ["Brahm", "Waymond", "Gwen"]
```

0 1 2

```
third_element = names[2]  # "Gwen"
```

Accessing Elements in a list

```
names = ["Brahm", "Waymond", "Gwen"]
```

0 1 2

```
last_element = names[-1]  # "Gwen"
```

Negative indices **count backwards** from the end of the list

Updating Elements in a List

```
names = ["Brahm", "Rebecca", "Gwen"]
```

0 1 2


```
names[1] = "Rebecca"
```

Getting the number of elements

```
names = ["Brahm", "Rebecca", "Gwen"]
```

0 1 2

```
num_names = len(names)  # 3
```

Doubling a list

How can we go from this….

```
numbers = [1, 2, 3, 4]
```

… to this?

```
numbers = [2, 4, 6, 8]
```

Doubling a list

```
numbers = [1, 2, 3, 4]

for i in range(len(numbers)):
    elem_at_index = numbers[i]
    numbers[i] = 2 * elem_at_index


print(numbers)  # prints [2, 4, 6, 8]
```

`range(len(numbers))` is the same as `range(4)`...

Doubling a list

```
numbers = [1, 2, 3, 4]

for i in range(len(numbers)):
    elem_at_index = numbers[i]
    numbers[i] = 2 * elem_at_index


print(numbers)  # prints [2, 4, 6, 8]
```

...which gives us the numbers 0, 1, 2 and 3...

Doubling a list

```
numbers = [1, 2, 3, 4]
```

...which are the indices of this list...

```
for i in range(len(numbers)):
    elem_at_index = numbers[i]
    numbers[i] = 2 * elem_at_index


print(numbers)  # prints [2, 4, 6, 8]
```

Doubling a list

```
numbers = [1, 2, 3, 4]

for i in range(len(numbers)):
    elem_at_index = numbers[i]
    numbers[i] = 2 * elem_at_index


print(numbers)  # prints [2, 4, 6, 8]
```

...so this `for` loop loops through **each index** in the list...

Doubling a list

```
numbers = [1, 2, 3, 4]

for i in range(len(numbers)):
    elem_at_index = numbers[i]
    numbers[i] = 2 * elem_at_index


print(numbers)  # prints [2, 4, 6, 8]
```

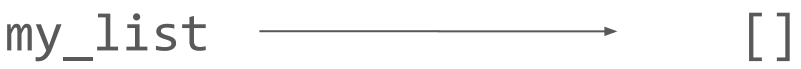
...so this `for` loop loops through **each index** in the list...

...and inside the loop, we get each element, double it, and put it back in the list

List Operations

Adding to a list

```
my_list = []
```



Adding to a list

```
my_list.append(42)
```

The append function adds an element **to the end** of a list

```
my_list  ──────────────────→  [42]
```

Adding to a list

```
my_list.append(100)
```

```
my_list ──────────────────→ [42, 100]
```

Adding to a list

```
my_list.append(28)
```

```
my_list  ──────────────────→  [42, 100, 28]
```

Removing from a list

```
x = my_list.pop()
```

The pop function removes an element **from the end** of a list and **returns** it

```
my_list ──────────────→  [42, 100]

   x    ──────────────→     28
```

Removing from a list

```
x = my_list.pop()
```

my_list → [42]

x → 100

Removing from a list

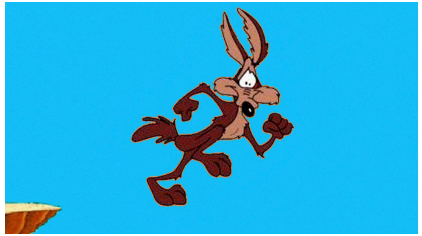
```
x = my_list.pop()
```

my_list → []

x → 42

Removing from a list

```
x = my_list.pop()
```



my_list → []

x → 42

IndexError: pop from empty list

Does a list contain an element?

```
my_list = [42, 100, 10]
if 42 in my_list:
    print("List has element!")
else:
    print("Element not present")
```

```
my_list = [42, 100, 10]
if -3 in my_list:
    print("List has element!")
else:
    print("Element not present")
```

`element in my_list` evaluates to `True` if `element` is in `my_list` and `False` otherwise

A new for loop

```
numbers = [1, 2, 3, 4]

for i in range(len(numbers)):
    elem = numbers[i]
    print(elem)
```

```
numbers = [1, 2, 3, 4]

for elem in numbers:
    print(elem)
```

These two `for` loops iterate over each of the elements in the list **in the same order**

A new for loop

```
numbers = [1, 2, 3, 4]

for i in range(len(numbers)):
    elem = numbers[i]
    print(elem)
```

Use this loop when you need access to **the indices** of the elements

```
numbers = [1, 2, 3, 4]

for elem in numbers:
    print(elem)
```

Use this loop when you only need **the values** of the elements

A new for loop

```
for element in collection:
    # do something with element
```

A list is one type of *collection* and in the next lesson, you'll encounter another!

Back to add_many_numbers

```
def add_many_numbers(num1, num2, ..., last_num):
    return num1 + num2 + ... last_num
```

Back to add_many_numbers

We can use a list to represent arbitrarily many numbers

```
def add_many_numbers(num1, num2, ..., last_num):
    return num1 + num2 + ... last_num
```

Back to add_many_numbers

```
def add_many_numbers(numbers):
    # TODO
```

How do we find the sum of the elements in a list?

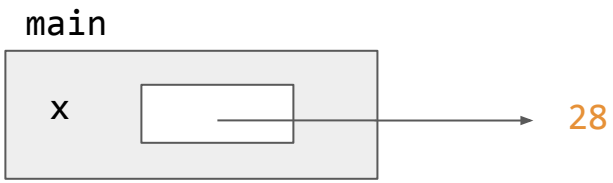
Interlude: The Python Memory Model

A tracing problem

```
def main():
    x = 28
    change_value(x)
    print(x)  # what gets printed here?


def change_value(n):
    n = 42
```

A tracing problem

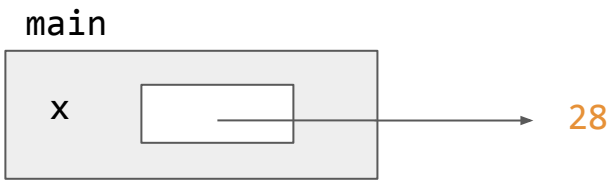
```
def main():
    x = 28
    change_value(x)
    print(x)

def change_value(n):
    n = 42
```



At the start of the program, the `main` function's x variable stores the value 28

A tracing problem

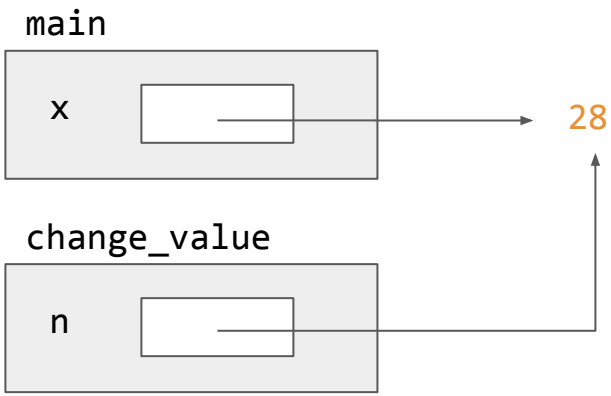
```
def main():
    x = 28
    change_value(x)
    print(x)

def change_value(n):
    n = 42
```



Now, it's time to call the `change_value` function

A tracing problem

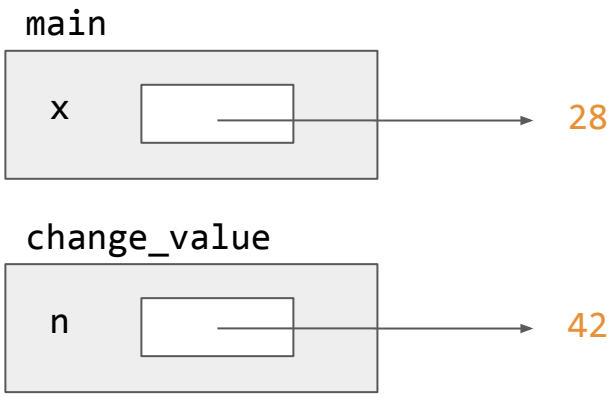
```
def main():
    x = 28
    change_value(x)
    print(x)

def change_value(n):
    n = 42
```



change_value's n parameter
also has the value 28

A tracing problem

```
def main():
    x = 28
    change_value(x)
    print(x)
```

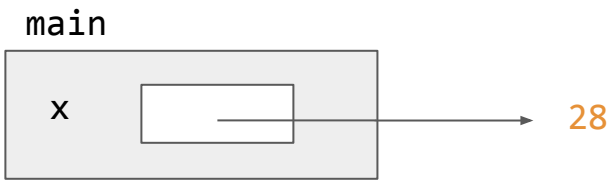



```
def change_value(n):
    n = 42
```

Now, *only* `change_value`'s variable is set to 42...

A tracing problem

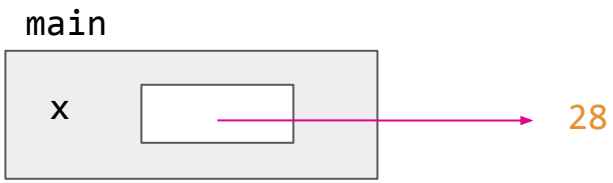
```
def main():
    x = 28
    change_value(x)
    print(x)

def change_value(n):
    n = 42
```



So when we return to `main`, it's x variable is still equal to 28…

A tracing problem

```
def main():
    x = 28
    change_value(x)
    print(x)

def change_value(n):
    n = 42
```



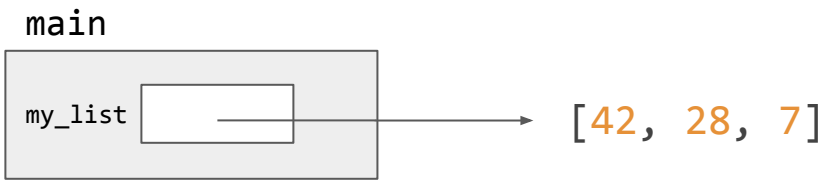
...so that's what we `print`!

Another tracing problem

```
def main():
    my_list = [42, 28, 7]
    change_value(my_list)
    print(my_list)  # what gets printed here?


def change_value(lst):
    lst.append(42)
```

Another tracing problem

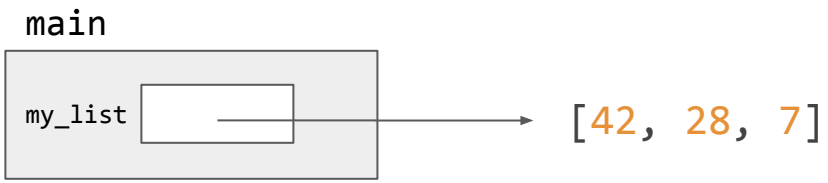
```
def main():
    my_list = [42, 28, 7]
    change_value(my_list)
    print(my_list)

def change_value(lst):
    lst.append(42)
```



At the start of the program, the `main` function's `my_list` variable stores the value `[42, 28, 7]`

Another tracing problem

```
def main():
    my_list = [42, 28, 7]
    change_value(my_list)
    print(my_list)
```

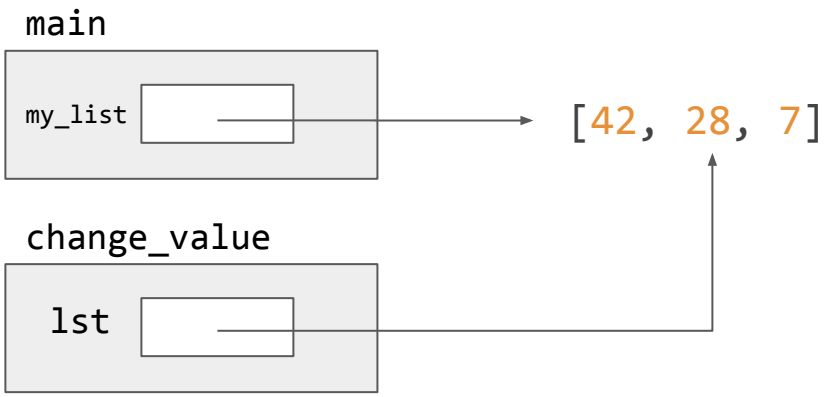



```
def change_value(lst):
    lst.append(42)
```

Now, it's time to call the `change_value` function

Another tracing problem

```
def main():
    my_list = [42, 28, 7]
    change_value(my_list)
    print(my_list)
```



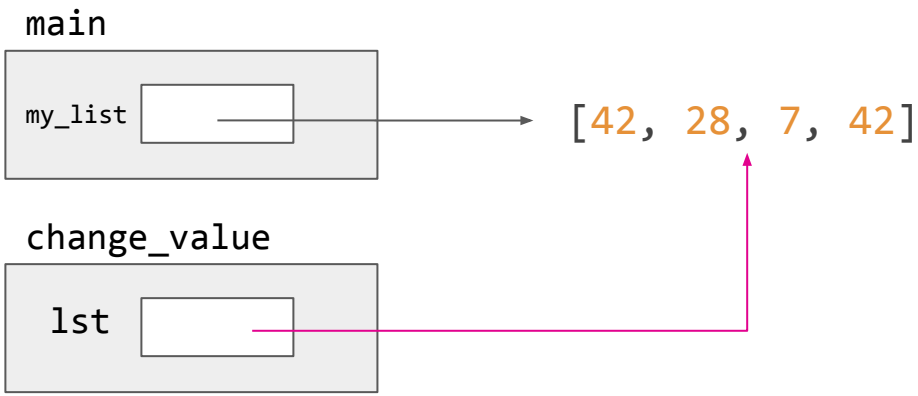

```
def change_value(lst):
    lst.append(42)
```

change_value's `lst` parameter also has the value `[42, 28, 7]`

Another tracing problem

```
def main():
    my_list = [42, 28, 7]
    change_value(my_list)
    print(my_list)
```

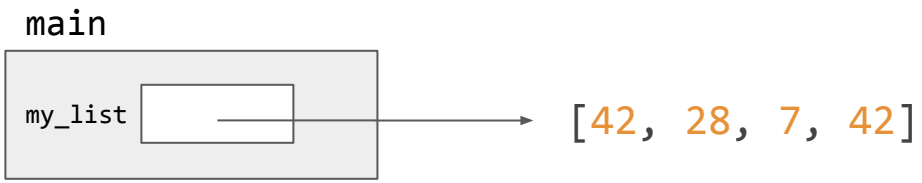
```
def change_value(lst):
    lst.append(42)
```



Now, `change_value` follows the arrow from `lst` and appends 42 to the list

Another tracing problem

```
def main():
    my_list = [42, 28, 7]
    change_value(my_list)
    print(my_list)
```

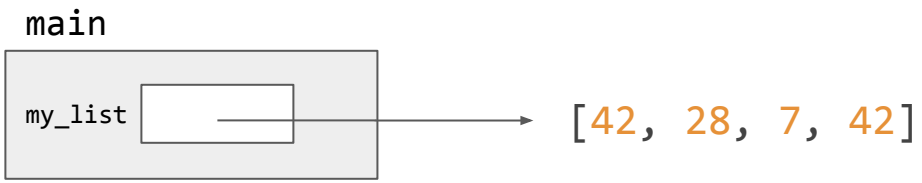



```
def change_value(lst):
    lst.append(42)
```

So when we return to `main`, the list remains modified…

Another tracing problem

```
def main():
    my_list = [42, 28, 7]
    change_value(my_list)
    print(my_list)
```

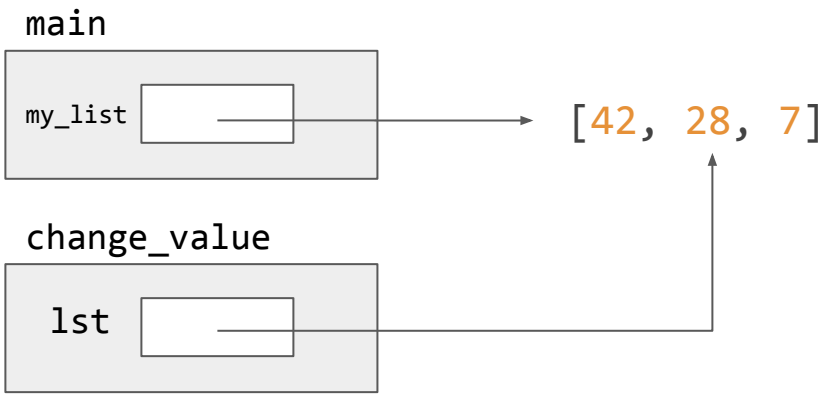



```
def change_value(lst):
    lst.append(42)
```

...so that's what we `print`!

What if we made a new list instead?

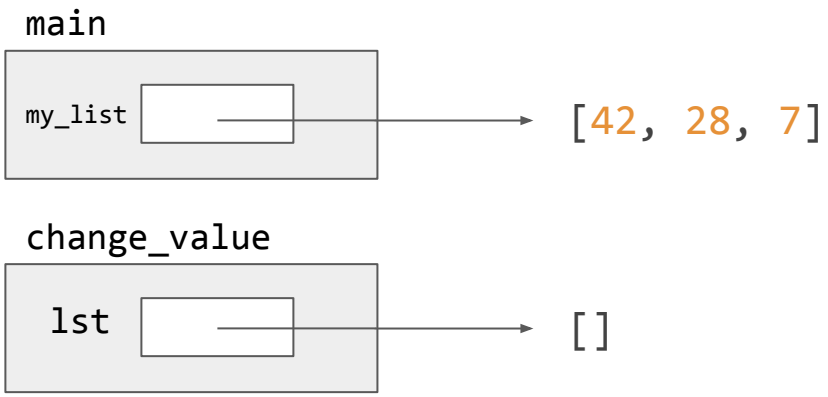
```
def main():
    my_list = [42, 28, 7]
    change_value(my_list)
    print(my_list)
```




```
def change_value(lst):
    lst = []
```

What if we made a new list instead?

```
def main():
    my_list = [42, 28, 7]
    change_value(my_list)
    print(my_list)
```

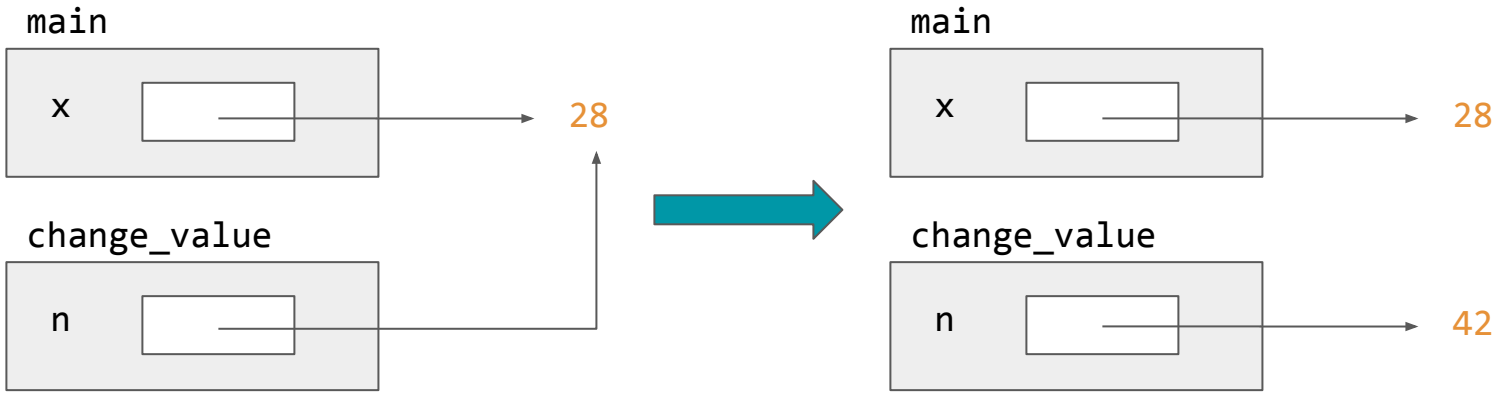



```
def change_value(lst):
    lst = []
```

What's going on here?

Variables like integers, floats and strings are **immutable**

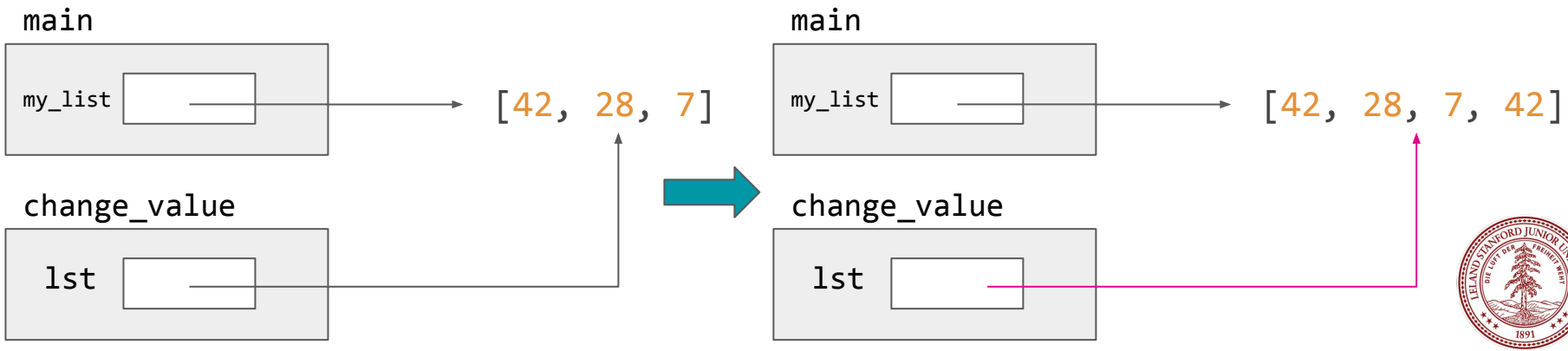
The designers of Python decided that you ***can't modify the value of an immutable variable, except by setting it equal to a new value***



What's going on here?

Lists and most other data structures are ***mutable***

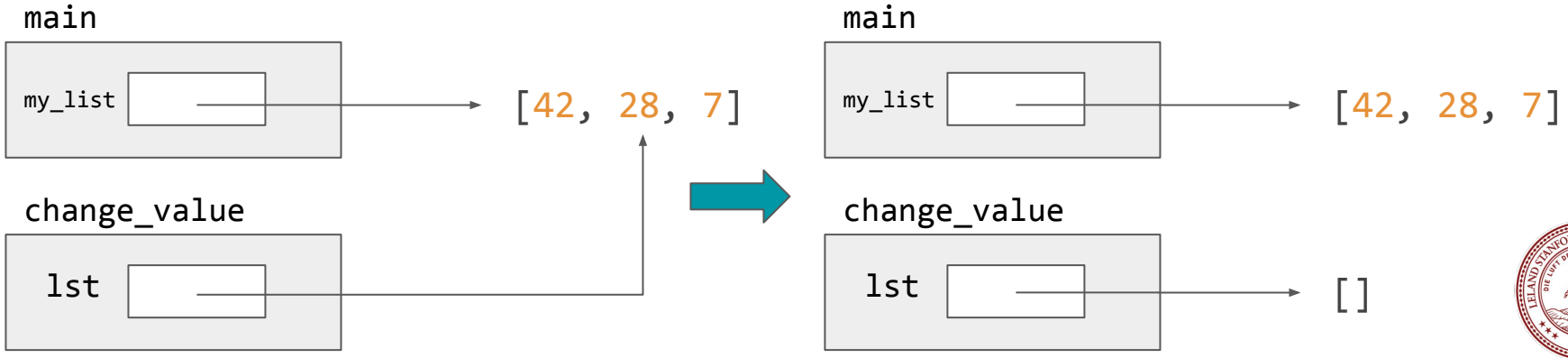
You can modify the value of a list without creating a new one (for example, by appending to it)



What's going on here?

The only way to make a new list is to ***explicitly create a new one***

```
def change_value(lst):
    lst = []
```



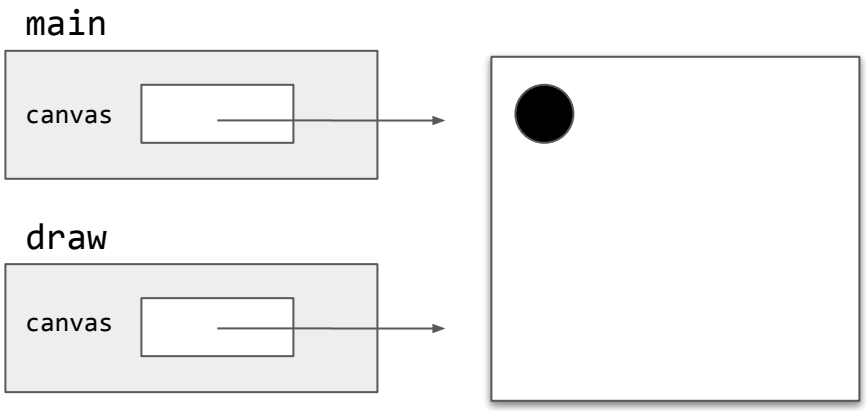
Mutability and immutability

	Example Types	Parameter Behaviour	When do we get a new one?
Immutable	`int, float, bool, string`	Parameter values are ***not*** modified	Anytime we change it at all (e.g. adding to an `int`)
Mutable	`list` (and others™)	Parameter values ***can be*** modified (e.g.. a list can be appended to or popped from inside another function)	Anytime we explicitly create a new one using the = sign

Immutable: The only way to change what's in the suitcase is by buying a new suitcase
Mutable: You can put things in the suitcase or take them out

Where else have we seen this?

```
def main():
    canvas = Canvas(400, 400)
    draw(canvas)

def draw(canvas):
    canvas.create_oval(10, 10, 10, 10)
```



Back to the for loop

```
numbers = [1, 2, 3, 4]

for i in range(len(numbers)):
    numbers[i] += 1
```

numbers → [1, 2, 3, 4]

Modifies the elements ***in the list***

```
numbers = [1, 2, 3, 4]

for elem in numbers:
    elem += 1
```

numbers → [1, 2, 3, 4]

elem → 1

Modifies a ***copy of each element***

After the 1st loop

```
numbers = [1, 2, 3, 4]

for i in range(len(numbers)):
    numbers[i] += 1
```

numbers → [2, 2, 3, 4]

Modifies the elements ***in the list***

```
numbers = [1, 2, 3, 4]

for elem in numbers:
    elem += 1
```

numbers → [1, 2, 3, 4]

elem → 2

Modifies a ***copy of each element***

After the 2nd loop

```
numbers = [1, 2, 3, 4]

for i in range(len(numbers)):
    numbers[i] += 1
```

numbers → [2, 3, 3, 4]

Modifies the elements ***in the list***

```
numbers = [1, 2, 3, 4]

for elem in numbers:
    elem += 1
```

numbers → [1, 2, 3, 4]

elem → 3

Modifies a ***copy of each element***

After the 3rd loop

```
numbers = [1, 2, 3, 4]

for i in range(len(numbers)):
    numbers[i] += 1
```

numbers → [2, 3, 4, 4]

Modifies the elements ***in the list***

```
numbers = [1, 2, 3, 4]

for elem in numbers:
    elem += 1
```

numbers → [1, 2, 3, 4]

elem → 4

Modifies a ***copy of each element***

After the 4th loop

```
numbers = [1, 2, 3, 4]

for i in range(len(numbers)):
    numbers[i] += 1
```

numbers → [2, 3, 4, 5]

Modifies the elements ***in the list***

```
numbers = [1, 2, 3, 4]

for elem in numbers:
    elem += 1
```

numbers → [1, 2, 3, 4]

elem → 5

Modifies a ***copy of each element***

A Tour of Lists

What have we seen already?

```
len(my_list)          # returns the number of elements in a list
my_list.append(42)    # adds an element to the end of a list
my_list.pop()         # removes last element and returns it
```

Is a list empty?

```
my_list = [42]
if my_list:
    print("List has elements!")
else:
    print("List is empty!")
```

```
my_list = []
if my_list:
    print("List has elements!")
else:
    print("List is empty!")
```

Using a list as a condition evaluates to `True` if it has elements and `False` if it's empty.

How else can you remove elements?

```
my_list = [42, 100, 10]
removed = my_list.pop(1)
```

my_list → [42, 10]

removed → 100

my_list.pop can take a parameter which specifies which index to remove from

```
my_list = [42, 100, 10, 42]
my_list.remove(42)
```

my_list → [100, 10, 42]

my_list.remove removes the first instance of an element in a list

How else can you add elements?

```
my_list = [42, 100, 10]
another = [2, 3, 4]
my_list.extend(another)
```

my_list → [42, 100, 10, 2, 3, 4]

another → [2, 3, 4]

my_list.extend adds all the elements from one list to my_list

```
my_list = [42, 100, 10]
another = [2, 3, 4]
combined = my_list + another
```

my_list → [100, 10, 42]

another → [2, 3, 4]

combined → [42, 100, 10, 2, 3, 4]

Adding two lists creates a new list with the elements from each

How else can you add elements?

```
my_list = [42, 100, 10]
another = [2, 3, 4]
my_list.extend(another)
```

my_list → [42, 100, 10, 2, 3, 4]

another → [2, 3, 4]

my_list.extend adds all the elements from one list to my_list

my_list += another will work the same way as extend

```
my_list = [42, 100, 10]
another = [2, 3, 4]
combined = my_list + another
```

my_list → [100, 10, 42]

another → [2, 3, 4]

combined → [42, 100, 10, 2, 3, 4]

Adding two lists creates a new list with the elements from each

Getting and using indices

```
my_list = [42, 100, 10, 100]
idx = my_list.index(100)
```

my_list → [42, 100, 10]

idx → 1

my_list.index finds the first index of an element in a list

```
my_list = [42, 100, 10]
my_list.insert(1, 27)
```

my_list → [42, 27, 100, 10]

my_list.insert inserts an element at a specified index and shifts the others down

Functions on lists

```
>>> my_list = [42, -7, 250, 12, 3]

>>> max(my_list)  # return the largest element in the list
250
>>> min(my_list)  # return the smallest element in the list
-7
>>> sum(my_list)  # return the sum of the elements in the list
300
```