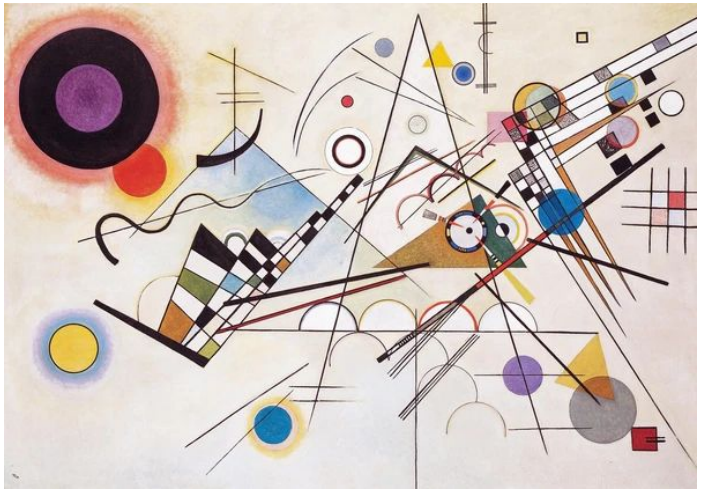


# JamCoders: Week 1

Lecture 4B:

- Mutability
- More Lists

# List Recap

# List Slicing

Slicing works for lists just like for strings (because lists and strings are both types of sequences, and slicing works for all sequences)!

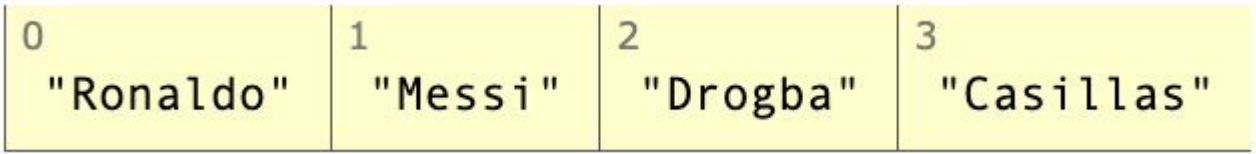
```
ta_list = ["Lydia", "Joy", "Hanna", "Zaria"]

all_but_one_tas = tas[1:]
print(all_but_one_tas)  # ['Joy', 'Hanna', 'Zaria']
```

Slicing makes a new list.

# List Indexing

We use brackets to access a List's values.

```
players = ["Ronaldo", "Messi", "Drogba", "Casillas"]
print(players[0])  # prints "Ronaldo"
print(players[1])  # prints "Messi"
print(players[-1])  # prints "Casillas"
```



# List Contains

To know whether a list contains a particular item, use the `in` keyword

```
ta_list = ["Lydia", "Joy", "Hanna", "Zaria"]

zaria_is_ta = "Zaria" in ta_list  # True
manolis_is_ta = "Manolis" in ta_list  # False
```

# Rapid Fire List Functions

```
numbers = [5, 6, 12, 9, 4]

# Find largest number in list
max_num = max(numbers)  # == 12

# Find smallest number in list
min_num = min(numbers)  # == 4

# Find the sum of numbers in list
total = sum(numbers)  # == 36

fruits = ["apple", "banana", "orange"]
```

# Mutability

# Mutability

Mutate means *to change*. So **mutable** means able to change.

Mutable

Mut-able

Mutate-able

Able to Mutate

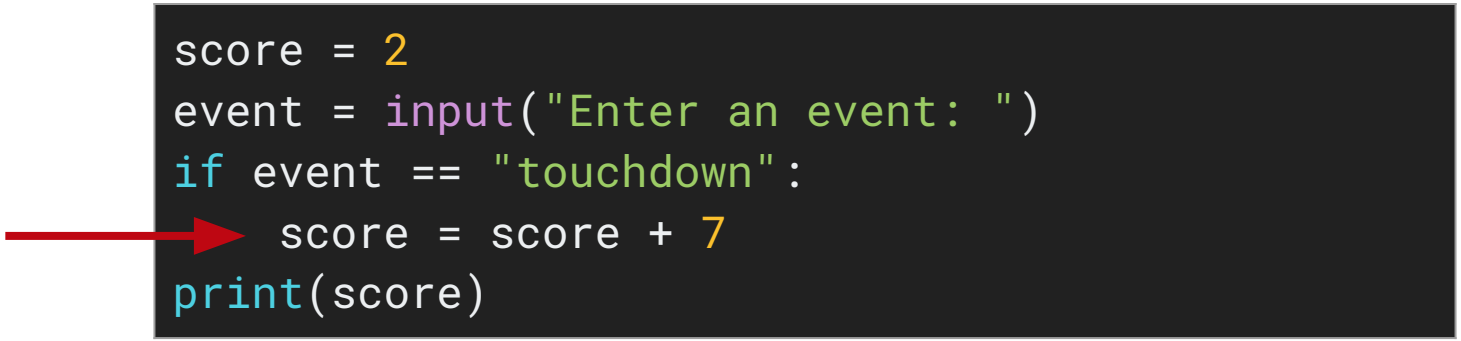
# Mutability

The first four datatypes we learned were **immutable**. You couldn't change the value, you could only replace the value…

- Integers
- Floats
- Booleans
- Strings

# Immutability

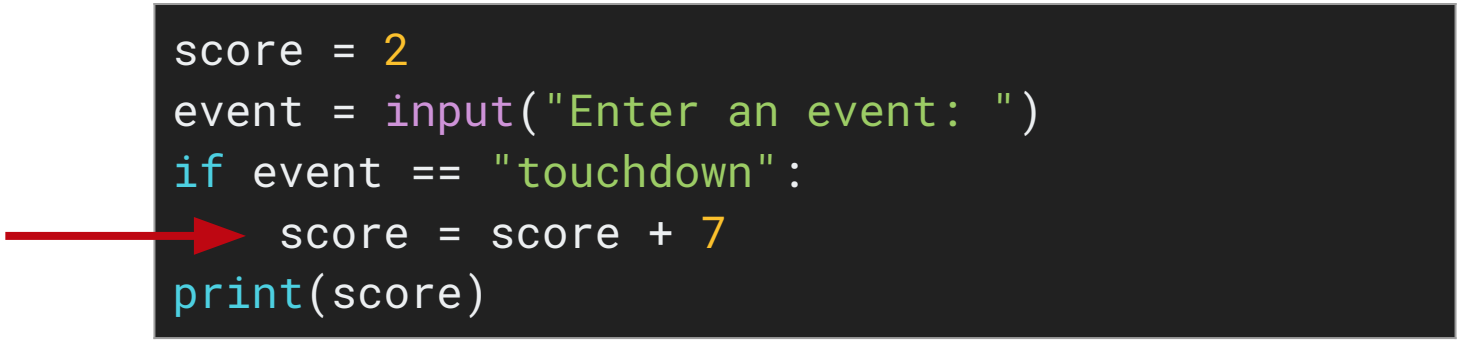
Common confusion: but wasn't this an example of *mutating* an integer?



```
score = 2
event = input("Enter an event: ")
if event == "touchdown":
    score = score + 7
print(score)
```

# Immutability

Common confusion: but wasn't this an example of *mutating* an integer?



```
score = 2
event = input("Enter an event: ")
if event == "touchdown":
    score = score + 7
print(score)
```

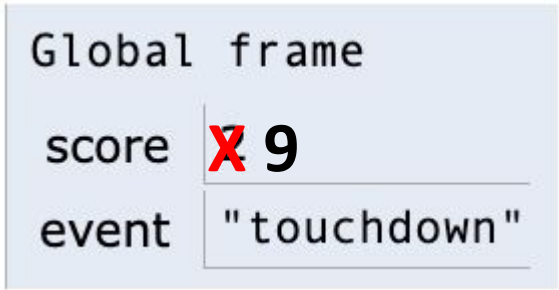
Answer: No! In score = score + 7, we first evaluate the expression `score + 7` and created a *new* integer (in this case 9), and then *replace* the previous value via assignment.

# Immutability

Common confusion: but wasn't this an example of *mutating* an integer?

```
score = 2
event = input("Enter an event: ")
if event == "touchdown":
    score = score + 7
print(score)
```

Answer: No! In score = score + 7, we first evaluate the expression `score + 7` and created a *new* integer (in this case 9), and then *replace* the previous value via assignment.



# Mutability

But some objects in Python are mutable; their value can be modified after creation *without* replacing the whole object.

**Lists** are the first mutable type you're learning!

# Mutability

To update a single value in a list, you assign to the *indexed* value.

```
grades = [95, 70, 92]
print(grades)  # prints "[95, 70, 92]"
grades[1] = 90
print(grades)  # prints "[95, 90, 92]"
```

# Mutability

To update a single value in a list, you assign to the *indexed* value.

```
grades = [95, 70, 92]
print(grades)  # prints "[95, 70, 92]"
grades[1] = 90
print(grades)  # prints "[95, 90, 92]"
```

To update multiple values at once, you can assign to the *sliced* value.

```
grades = [95, 70, 92]
print(grades)  # prints "[95, 70, 92]"
grades[1:] = [90, 100]
print(grades)  # prints "[95, 90, 100]"
```

# Coding Demo

# Lists in the Frame

# Lists in the Frame

Why am I showing you this?

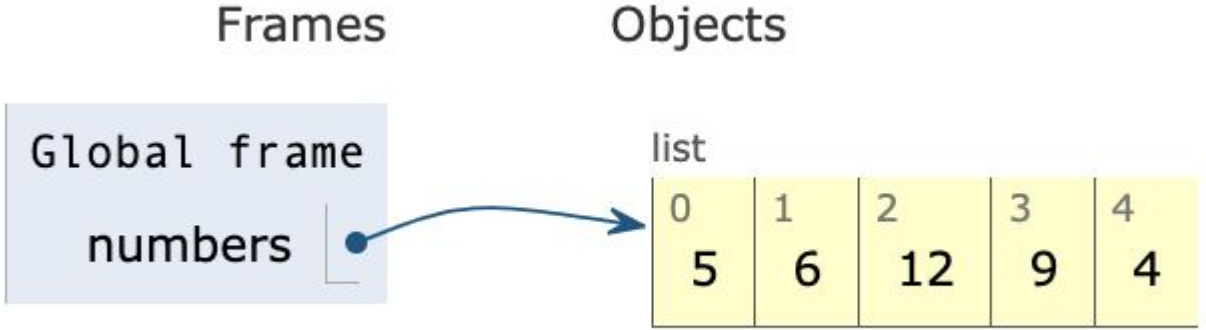
- Show you how PythonTutor represents Lists
- Hint to you what's actually going on deep down, which will help you solve problems and understand your code more easily.

Everything that PythonTutor shows is real and there for a reason. I won't focus on them heavily now, but these details will come back in later CS classes.

# Lists in the Frame

Lists are a more complicated type of object that the other types we've seen, so they're written down separately, and the variable "points" us to the list.

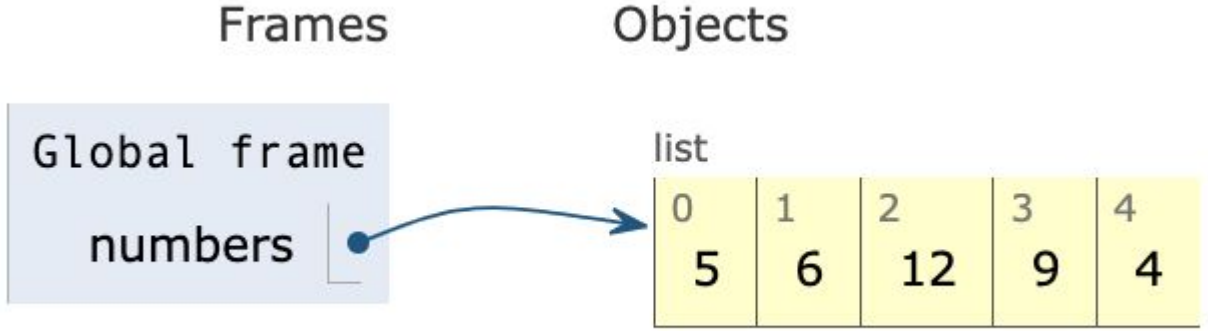
```
numbers = [5, 6, 12, 9, 4]
```



# Lists in the Frame

Lists are a more complicated type of object that the other types we've seen, so they're written down separately, and the variable "points" us to the list.

```
numbers = [5, 6, 12, 9, 4]
```

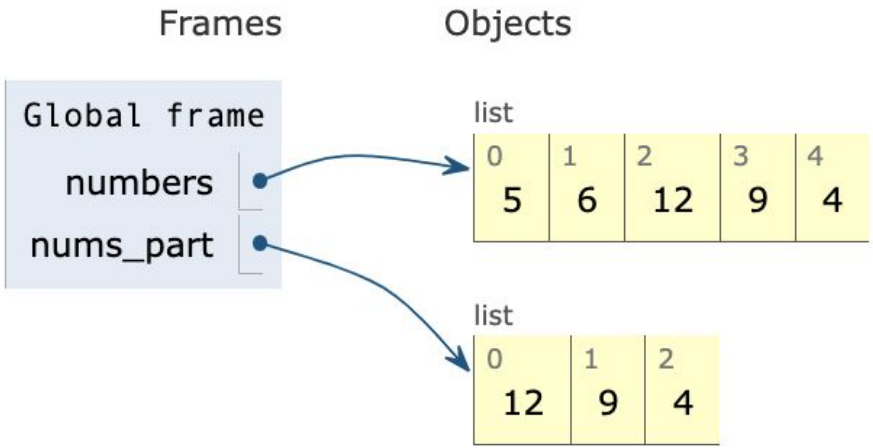



Think of this as the list itself being written down on a separate special piece of paper that is just for complicated types.

# Lists in the Frame

When you make a copy of a list (like by slicing), it makes a *new* list.

```
numbers = [5, 6, 12, 9, 4]
nums_part = numbers[2:]
```



# Lists in the Frame

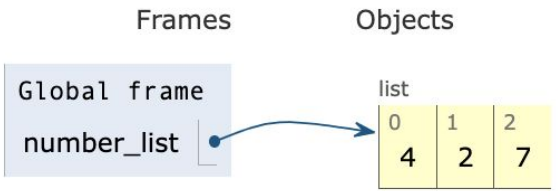
When you pass a list into a function as an argument, you point to the *same* list in the local frame as in the global frame. This means you can **mutate** the given list within a function.

```
number_list = [4, 2, 7]
def add_number(nums, x):
  nums.append(x)

add_number(number_list, 12)
print(number_list)
```

# Lists in the Frame

When you pass a list into a function as an argument, you point to the *same* list in the local frame as in the global frame. This means you can **mutate** the given list within a function.

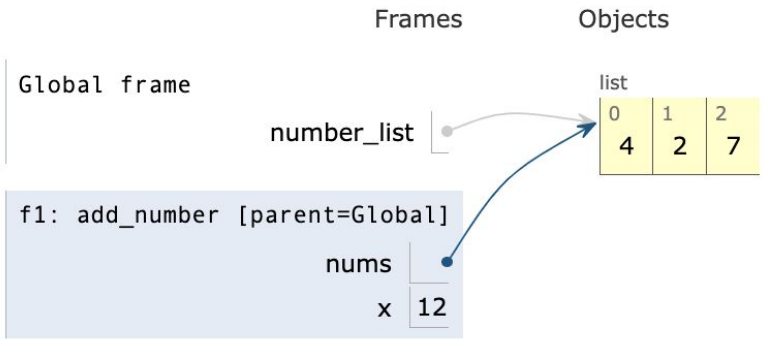
```
number_list = [4, 2, 7]
def add_number(nums, x):
  nums.append(x)

add_number(number_list, 12)
print(number_list)
```



# Lists in the Frame

When you pass a list into a function as an argument, you point to the *same* list in the local frame as in the global frame. This means you can **mutate** the given list within a function.

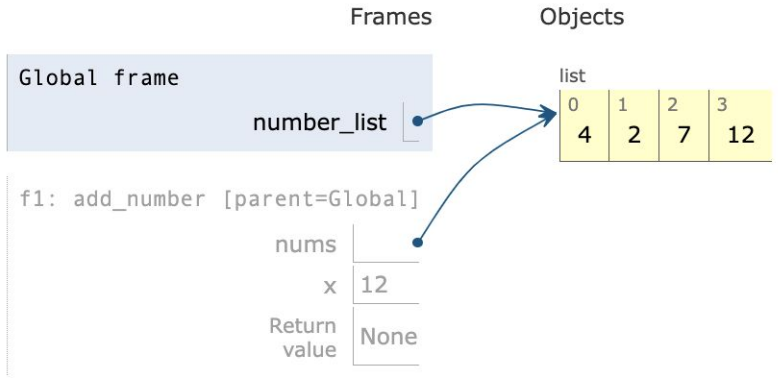
```
number_list = [4, 2, 7]
def add_number(nums, x):
  nums.append(x)

add_number(number_list, 12)
print(number_list)
```



# Lists in the Frame

When you pass a list into a function as an argument, you point to the *same* list in the local frame as in the global frame. This means you can **mutate** the given list within a function.

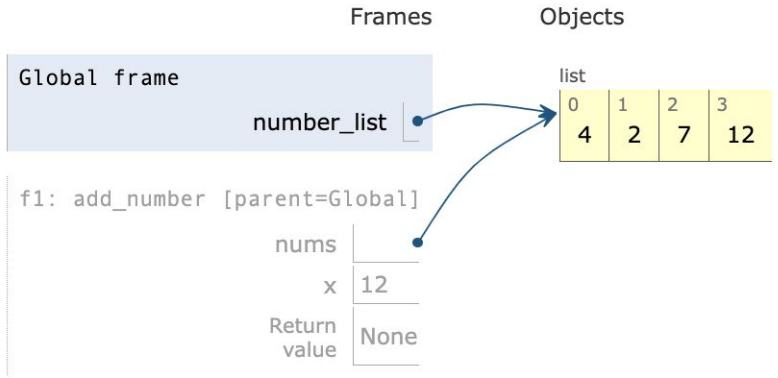
```
number_list = [4, 2, 7]
def add_number(nums, x):
    nums.append(x)

add_number(number_list, 12)
print(number_list)
```



# Lists in the Frame

When you pass a list into a function as an argument, you point to the *same* list in the local frame as in the global frame. This means you can **mutate** the given list within a function.

```
number_list = [4, 2, 7]
def add_number(nums, x):
  nums.append(x)

add_number(number_list, 12)
print(number_list)
```




```
~/CSCI100-Repl$ python3 main.py
[4, 2, 7, 12]
```