LISTS

An Introduction to Computer Science

Let's learn about Lists.

Primitives vs. Composites

Primitive

- Boolean
- Integer
- Float
- String
- None

Composite

- List

We have learned about 5 primitive types: Booleans, Integers, Floats, Strings, and the special None.
Python actually has many more types, which are called Composite types.
Composite types are named because they are composed of other types, unlike primitive types.

Lists

```
[45, 55, 32]

["Apples", "Oranges", "Bananas"]

[True, False, True]
```

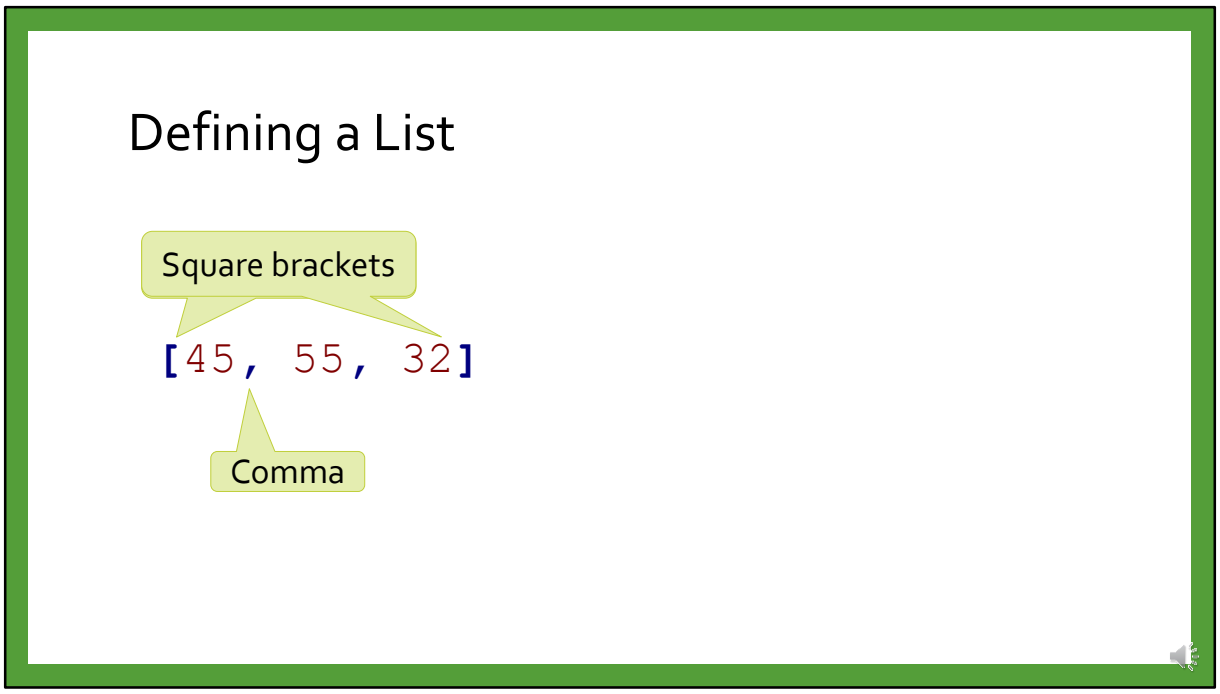
The first composite type we will learn is the List type.
A list is a sequence of values in a specific order.
For instance, you could have a list of numbers, or a list of strings.

Lists of X

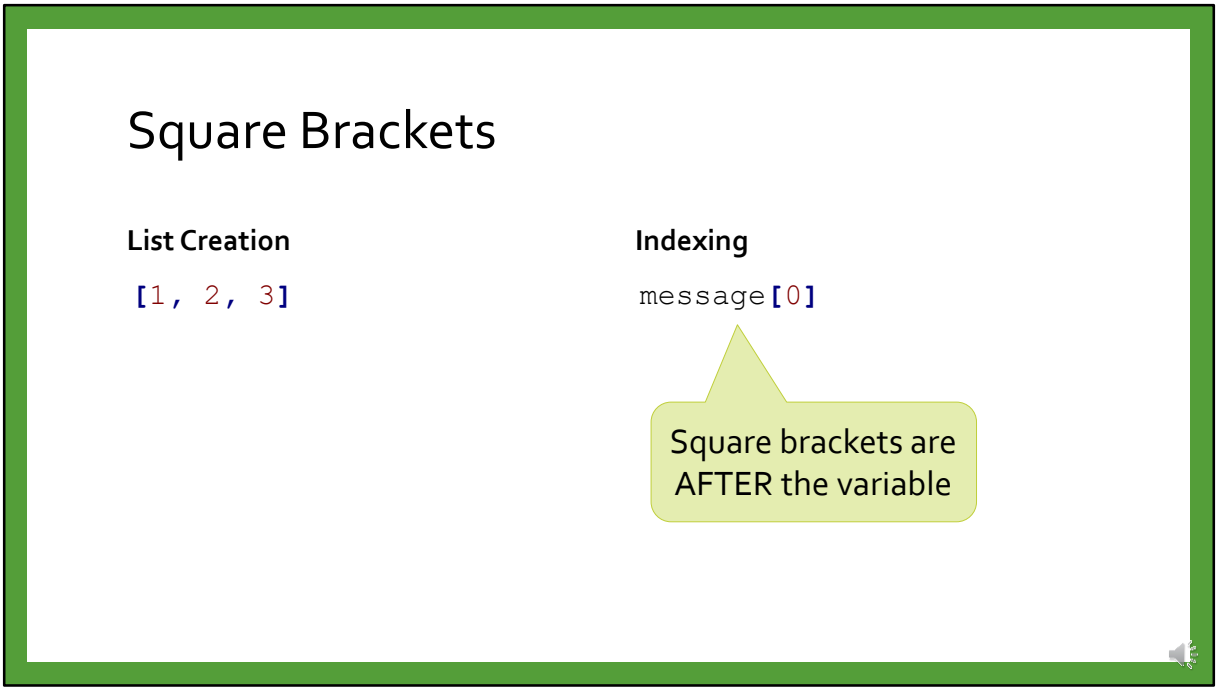
```
[45, 55, 32]
```

Type: List
Element type: Integer

A "List" is a type, much the same way that Integer is a type.
However, the List will also have an "element type", which is what it is composed of.
So if you have a list of integers, its type is "List" and its element type is "integer".



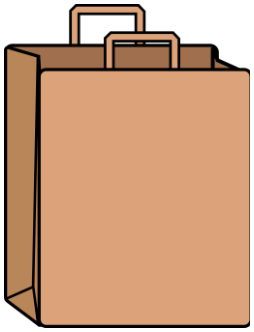
We create lists using square brackets.
Each value that we want to put into the list is separated by commas.
The square brackets are critical to making this a list; without them it is not a list.



We've seen square brackets used before, when we were subscripting and indexing a list.
Confusingly, Python ALSO uses square brackets for creating lists.
You might struggle at first figuring out when you're looking at subscripting or list creation.
The key is whether the square brackets come AFTER an expression - that means subscripting.
If the square brackets are on their own, you have a list.

Empty Lists

```
[]
```



A pair of square brackets with nothing between them creates an empty list.
An empty list is like an empty bag: you may not have anything in it, but you still have the bag itself.
Empty lists do not have an element type, until you put something in them.

Truthiness of Lists

```
my_wallet = [10, 10, 20, 1, 1]
if my_wallet:
    print("I have some money!")
```

Non-empty list evaluates to True

When used as a conditional, a list evaluates to True unless it is empty.
This can be very convenient when writing conditional expressions involving lists.

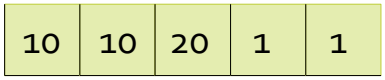
Printing Lists

```
>>> names = ["Alice", "Bob", "Carol"]

>>> print(names)

['Alice', 'Bob', 'Carol']
```

When you print a list, you will notice that the square brackets are printed too.
In fact, Python print each value of the list as if it was a literal value.
So notice that the strings have quotes around them now!

The Box Model

```
[10, 10, 20, 1, 1]
```

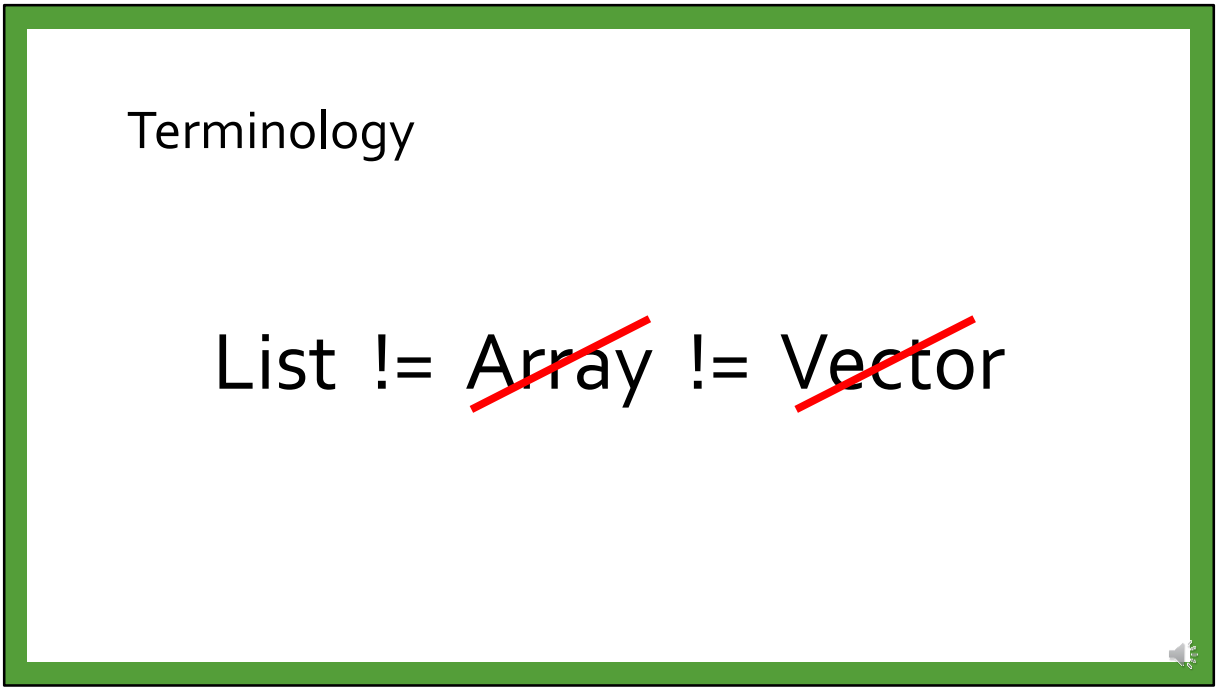


Lists are often shown in diagrams as a row of boxes, to help us visualize the data. Each box will have a value in it.

List Type

```
def transform_list(some_list: list) -> int:
    pass


def ints_to_strs(numbers: [int]) -> [str]:
    pass
```

When you define a function, you need to specify the types of the parameters and the return value.
If you don't know the element type of the list, you can just use the `list` type like we would a string or an integer.
But if we do know the element type, we wrap the type expression in square brackets.
The top function consumes a list and produces an integer.
The bottom function consumes a list of integers and produces a list of strings.
There are other ways to specify a list type, but this will be good enough for now.



Some people will use the words "Array" or "Vector" to describe lists.
These are actually slightly different concepts, which we will not cover in this course.
Don't be thrown off if someone refers to a list as an "Array", but don't use that word yourself!