



# Intro to Lists

(if you have your computer today, get VSCode ready!)

# Announcements

- **LS10 (Lists)** due tonight at 11:59pm
- **EX03 (Wordle)** released today, due Monday, February 23 at 11:59pm

# Lists

Examples of lists:

- To-do list
- Assignment due dates
- Grocery list

A list is a **data structure**—something that lets you organize and store data in a format such that they can be accessed and processed efficiently.

Lists are **mutable**, meaning *their values can be changed after initialization*.

*NOTE: Lists can be an arbitrary (but finite) length! (Not a fixed number of items.)*

# Lists are Mutable Sequences in Python

Sequences are ordered, 0-indexed collections of values

Feature	Syntax	Purpose
Type Declaration		
Constructor (function)		
List Literal		
Access Value		
Assign Item		
Length of List		

**Optional:** Complete this table as we cover each topic today!

# Lists are Mutable Sequences in Python

Sequences are ordered, 0-indexed collections of values

Feature	Syntax	Purpose
Type Declaration		
Constructor (function)		
List Literal		
Access Value		
Assign Item		
Length of List		

# Declaring the type of a list

<list name>: list[<item type>]

grocery\_list: list[str]

# Declaring the type of a list

<list name>: list[<item type>]

grocery\_list: list[str]



str, int, float, etc.

# Initializing a list

With a constructor:

- `<list name>: list[<item type>] = list()`
- `grocery_list: list[str] = list()`

The constructor `list()` is a *function* that returns the literal `[]`

With a literal:

- `<list name>: list[<item type>] = []`
- `grocery_list: list[str] = []`

declare variable

initialize list

“create a var called `grocery_list`, a list of strings, which will initially be empty”

# Initializing a list

With a constructor:

- `<list name>: list[<item type>] = list()`
- `grocery_list: list[str] = list()`

The constructor `list()` is a *function* that returns the literal `[]`

With a literal:

- `<list name>: list[<item type>] = []`
- `grocery_list: list[str] = ["apples", "bananas", "pears"]`

declare variable

initialize list

“create a var called `grocery_list`, a list of strings, which will initially contain these values”

# Initializing a list

With a constructor:

- `<list name>: list[<item type>] = list()`
- `grocery_list: list[str] = list()`

With a literal:

- `<list name>: list[<item type>] = []`
- `grocery_list: list[str] = []`

The constructor **list()** is a *function* that returns the literal `[]`

Bringing it back to something we know, you can create an empty string using the constructor **str()** or the literal `""`

# Initializing a list

With a constructor:

- `<list name>: list[<item type>] = list()`
- `grocery_list: list[str] = list()`

With a literal:

- `<list name>: list[<item type>] = []`
- `grocery_list: list[str] = []`

The constructor **list()** is a *function* that returns the literal `[]`

Bringing it back to something we know, you can create an empty string using the constructor **str()** or the literal `""`

**Let's try it!**

Create an empty list of floats with the name `my_numbers`.

# Adding an item to the end of a list


```
<list name>.append(<item>)
```

```
grocery_list.append("bananas")
```

# Adding an item to the end of a list

```
<list name>.append(<item>)
```


```
grocery_list.append("bananas")
```

- 
- Method: a function that *belongs* to the **list** class
  - Like calling `append(grocery_list, "bananas")`

# Adding an item to the end of a list

```
<list name>.append(<item>)
```

```
grocery_list.append("bananas")
```

- 
- Method: a function that *belongs* to the **list** class
  - Like calling `append(grocery_list, "bananas")`

**Let's try it!**

Add the value 1.5 to my\_numbers.

## Initializing an already populated list

<list name>: `list[<item type>]` = [`<item 0>`, `<item 1>`, ... , `<item n>`]

grocery\_list: `list[str]` = ["bananas", "milk", "bread"]

# Initializing an already populated list

<list name>: `list[<item type>]` = [`<item 0>`, `<item 1>`, ... , `<item n>`]

grocery\_list: `list[str]` = ["bananas", "milk", "bread"]

## **Let's try it!**

Create a list called `game_points` that stores the following numbers: 102, 86, 94

# Indexing

```
grocery_list: list[str] = ["bananas", "milk", "bread"]
```

```
grocery_list[0]
```

*\*\*Starts at 0, like with strings!*

# Indexing

```
grocery_list: list[str] = ["bananas", "milk", "bread"]
```

```
grocery_list[0]
```

*\*\*Starts at 0, like with strings!*

## **Let's try it!**

In `game_points`, use subscription notation to print out 94.

## Modifying by index

```
grocery_list: list[str] = ["bananas", "milk", "bread"]
```

```
grocery_list[1] = "eggs"
```

## Modifying by index

```
grocery_list: list[str] = ["bananas", "milk", "bread"]
```

```
grocery_list[1] = "eggs"
```

### **Let's try it!**

In `game_points`, use subscription notation to change 86 to 72.

## Modifying by index

```
grocery_list: list[str] = ["bananas", "milk", "bread"]
```

```
grocery_list[1] = "eggs"
```

### **Let's try it!**

In `game_points`, use subscription notation to change 86 to 72.

*Question: Could you do this type of modification with a string? Try it out!*

## Length of a list

```
grocery_list: list[str] = ["eggs", "milk", "bread"]
```

```
len(grocery_list)
```

## Length of a list

```
grocery_list: list[str] = ["eggs", "milk", "bread"]
```

```
len(grocery_list)
```

**Let's try it!**

Print the length of  
game\_points.

# Remove an item from a list – “pop off!”

```
grocery_list: list[str] = ["eggs", "milk", "bread"]
```

```
grocery_list.pop(2)
```



Index of item you want to remove

# Remove an item from a list – “pop off!”

```
grocery_list: list[str] = ["eggs", "milk", "bread"]
```

```
grocery_list.pop(2)
```

Index of item you want to remove

Before: ["eggs", "milk", ~~"bread"~~]

Index:           0           1           2

After: ["eggs", "milk"]

Index:           0           1

# Remove an item from a list – “pop off!”

```
grocery_list: list[str] = ["eggs", "milk", "bread"]
```

```
grocery_list.pop(1)
```

Index of item you want to remove

Before: ["eggs", ~~"milk"~~, "bread"]

Index:           0           1           2



After: ["eggs", "bread"]


Index:           0           1

# Remove an item from a list – “pop off!”

```
grocery_list: list[str] = ["eggs", "milk", "bread"]
```

```
grocery_list.pop(2)
```

Index of item you want to remove



**Let's try it!**  
Remove 72 from  
game\_points.

## Code-writing example

- Implement a function named `contains` with two parameters:
  - `needle: int` we are searching for
  - `haystack: list[int]` of values we are searching through
- Return `True` if the needle appears at least once in the haystack, and `False` otherwise