

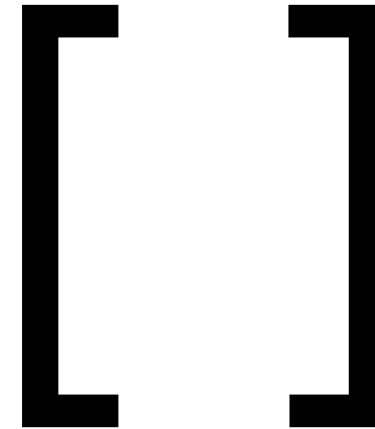
Lists

The big picture

- **Lists are one example of a bigger idea: the aggregate data structure**
 - way of grouping many values together under one name
- **Aggregate data structures you'll meet on your degree**
 - tuples – like lists, but immutable; sets – unordered, no duplicates; dictionaries (maps) – key → value pairs.
 - linked lists, trees, graphs etc
- **This is a basic introduction**

Overview

- **Lists**
 - Fridge scenario
 - Empty, populated lists, mixed populated lists, lists in lists
- **List 'slicing'**
 - Creating lists from other lists
- **2D lists**
 - Just lists (of lists)



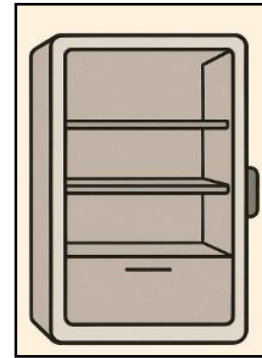
Bracket type

Lists I

- Basics of creating
- Simple fridge scenario

Fridge scenario (sparse fridge)

- **Imagine you have a fridge.**
 - At first, the fridge is completely empty.
 - You open the door and put an apple inside.
 - Now, when you look inside, you see an apple.
 - Then you add a carrot. Now the fridge has an apple and a carrot.



Empty list:

[]

state change



Fridge scenario (full fridge)

- **Imagine you have a fridge.**
 - fridge began completely empty, but soon an apple and a carrot were placed inside, followed by a shopping trip that filled it
- **Appended items:**
 - milk, butter, eggs, cheese, lettuce, grapes, cucumber, chicken, orange juice, and strawberries. What was once an empty cold box is now a fridge full of colourful and fresh food, ready for snacks, meals, and treats



Fridge scenario (full fridge)

- **Imagine you have a fridge.**
 - The fridge began completely empty, but soon an apple and a carrot were placed inside, followed by a shopping trip that filled it with milk, butter, eggs, cheese, lettuce, grapes, cucumber, chicken, orange juice, and strawberries. What was once an empty cold box is now a fridge full of colourful and fresh food, ready for snacks, meals, and treat



state change



```
1 def fridge_example():
2     """
3     ... Simple scenario of a first empty fridge which is
4     ... then filled with items
5     ... """
6
7     ... # Start with an empty fridge
8     ... fridge = []
9     ... print("Fridge (empty):", fridge)
10
11     ... # Add items to the fridge
12     ... fridge.append("apple")
13     ... fridge.append("carrot")
14     ... fridge.append("milk")
15     ... fridge.append("butter")
16     ... fridge.append("eggs")
17     ... fridge.append("cheese")
18     ... fridge.append("lettuce")
19     ... fridge.append("grapes")
20     ... fridge.append("cucumber")
21     ... fridge.append("chicken")
22     ... fridge.append("orange juice")
23     ... fridge.append("strawberries")
24
25     ... # Show final fridge contents
26     ... print("Fridge after adding items:", fridge)
```

Fridge scenario (full fridge)

- **Imagine you have a fridge with shelves.**

- The fridge began completely empty, but soon an apple and a carrot were placed inside, followed by a shopping trip that filled it with milk, butter, eggs, cheese, lettuce, grapes, cucumber, chicken, orange juice, and strawberries.
- You *place each item onto a given shelf* in the fridge.



state change



```
1 def fridge_example():
2     """
3     ... Simple scenario of a first empty fridge (empty of
4     ... food, but contains empty shelves), which is
5     ... then filled with items per shelf
6     ... """
7     ... # Fridge represented as a list of shelves (each shelf is a list)
8     ... fridge = [
9     ...     [], # Top shelf
10    ...     [], # Middle shelf
11    ...     [] # Bottom shelf
12    ... ]
13    ... print("Fridge (empty):", fridge)
14
15    ... # Add items to:
16    ... # Top shelf
17    ... fridge[0].extend(["apple", "carrot", "milk", "lettuce", "butter"])
18    ... # Middle shelf
19    ... fridge[1].extend(["eggs", "cheese", "lettuce", "cucumber", "grapes", "chicken"])
20    ... # Bottom shelf
21    ... fridge[2].extend(["orange juice", "apricot jam", "strawberries"])
22
23    ... # Show final fridge contents
24    ... print("Fridge after adding items:", fridge)
```

More methods (top 10 used)

- **LLM prompts**

- ‘Give me the top ten used list methods in Python (please).’
- Write a program that demonstrates the use of each method, and call that file `list_methods.py`.
- Include a main block and use functions for each separate example please.

Method	Purpose	Example usage	Result
<code>append(x)</code>	Add a single element <code>x</code> to the end of the list	<code>lst = [1, 2]; lst.append(3)</code>	<code>[1, 2, 3]</code>
<code>extend(iterable)</code>	Add all elements from another list (or iterable) to the end	<code>lst = [1, 2]; lst.extend([3, 4])</code>	<code>[1, 2, 3, 4]</code>
<code>insert(i, x)</code>	Insert element <code>x</code> at position <code>i</code> (shifts others right)	<code>lst = [1, 3]; lst.insert(1, 2)</code>	<code>[1, 2, 3]</code>
<code>remove(x)</code>	Remove the first occurrence of value <code>x</code>	<code>lst = [1, 2, 2, 3]; lst.remove(2)</code>	<code>[1, 2, 3]</code>
<code>pop([i])</code>	Remove and return element at index <code>i</code> (default last item)	<code>lst = [1, 2, 3]; lst.pop()</code>	returns <code>3</code> , list is <code>[1, 2]</code>
<code>clear()</code>	Remove all items (empty the list)	<code>lst = [1, 2]; lst.clear()</code>	<code>[]</code>
<code>index(x)</code>	Return the index of the first occurrence of value <code>x</code>	<code>lst = [10, 20, 30]; lst.index(20)</code>	<code>1</code>
<code>count(x)</code>	Return the number of times value <code>x</code> appears	<code>lst = [1, 2, 2, 3]; lst.count(2)</code>	<code>2</code>
<code>sort()</code>	Sort the list in ascending order (in place)	<code>lst = [3, 1, 2]; lst.sort()</code>	<code>[1, 2, 3]</code>
<code>reverse()</code>	Reverse the list in place	<code>lst = [1, 2, 3]; lst.reverse()</code>	<code>[3, 2, 1]</code>

Lists II

- Structure in data
- Generating data in loops

Structure in data

- **Fridge**

- Not too untidy for a fridge
- But not good example of well-structured data



- **Cities, temperatures**

- Cities
- Temperatures
- Cities, temperatures

cities:

['Milan' , 'Glasgow' , etc ...]

temperatures:

[25.7 , 12.3 , etc ...]

cities and temperatures:

['Milan' , 25.7 , 'Glasgow' , 12.3 , etc , etc...]

- **List of 2D coordinates**

- List of 2D lists

coordinates:

[[5 , 8] , [8 , 3] , [1 , 1] , [x , y] [etc]...]

Structure in data:

- **Cities, temperatures**
 - Cities (separate)
 - Temperatures (separate)
 - Cities, temperatures (together)
- **All 1D lists**

cities:
['London', 'Paris', etc ...]

temperatures:
[15, 22, etc ...]

cities and temperatures:
['London', 15, 'Paris', 22, etc, etc...]

```
1 def create_data_separate():
2     """
3     ... Create city and temperature data using separate lists.
4     ... Returns a tuple: (cities, temperatures).
5     ... """
6     cities = ["London", "Paris", "Berlin", "Rome", "Madrid"]
7     temperatures = [15, 22, 18, 25, 20]
8     return cities, temperatures
9
10
11 def create_data():
12     """
13     ... Create city and temperature data in a single list,
14     ... storing them at alternate positions.
15     ... Example: ["London", 15, "Paris", 22, ...]
16     ... """
17     data = [
18         "London", 15,
19         "Paris", 22,
20         "Berlin", 18,
21         "Rome", 25,
22         "Madrid", 20
23     ]
24     return data
25
26
27 def main():
28     # Method 1: Separate lists
29     cities, temperatures = create_data_separate()
30     print("Method 1 - Separate lists")
31     print("Cities:", cities)
32     print("Temperatures:", temperatures)
33
34     # Method 2: Combined list
35     data = create_data()
36     print("Method 2 - Combined list")
37     print("Data:", data)
```

- List of 2D lists

- Generate random 2D positions
- Plot positions in a 'grid'

```
ASCII plot of points:
```

Slicing

- Common list slicing operations

List slices

- **Why?**

- Sometimes you might want to grab sub-parts of data from a list.

- **Example**

- Separate temperatures and cities, from joined list, into single lists.

```
1 def create_data():
2     """
3     ... Create city and temperature data in a single list,
4     ... storing them at alternate positions.
5     ... Example: ["London", 15, "Paris", 22, ...]
6     """
7     data = [
8         "London", 15,
9         "Paris", 22,
10        "Berlin", 18,
11        "Rome", 25,
12        "Madrid", 20
13    ]
14    return data
15
16
17 def main():
18     data = create_data()
19
20     temps = data[1::2]
21     cities = data[0::2]
22
23     print("Original list:", data)
24     print("temps list:", temps)
25     print("cities members:", cities)
```

```
Original list: ['London', 15, 'Paris', 22, 'Berlin', 18, 'Rome', 25, 'Madrid', 20]
temps list: [15, 22, 18, 25, 20]
cities members: ['London', 'Paris', 'Berlin', 'Rome', 'Madrid']
```

Common operations

- **Check these**

- Create a script
- Put some data in a list
- Practice some of the operations

Slice operation	Meaning	Example (nums = [0,1,2,3,4,5,6,7,8,9])	Result
<code>nums[start:stop]</code>	Elements from <code>start</code> to <code>stop-1</code>	<code>nums[2:6]</code>	<code>[2, 3, 4, 5]</code>
<code>nums[start:]</code>	From <code>start</code> to the end	<code>nums[4:]</code>	<code>[4, 5, 6, 7, 8, 9]</code>
<code>nums[:stop]</code>	From start up to <code>stop-1</code>	<code>nums[:5]</code>	<code>[0, 1, 2, 3, 4]</code>
<code>nums[:]</code>	Copy of the whole list	<code>nums[:]</code>	<code>[0,1,2,3,4,5,6,7,8,9]</code>
<code>nums[start:stop:step]</code>	From <code>start</code> to <code>stop-1</code> with a step	<code>nums[1:8:2]</code>	<code>[1, 3, 5, 7]</code>
<code>nums[::2]</code>	Every second element (evens if starting at index 0)	<code>nums[::2]</code>	<code>[0, 2, 4, 6, 8]</code>
<code>nums[1::2]</code>	Every second element starting at index 1 (odds)	<code>nums[1::2]</code>	<code>[1, 3, 5, 7, 9]</code>
<code>nums[::-1]</code>	Reverse the list	<code>nums[::-1]</code>	<code>[9,8,7,6,5,4,3,2,1,0]</code>
<code>nums[-n:]</code>	Last <code>n</code> elements	<code>nums[-3:]</code>	<code>[7, 8, 9]</code>
<code>nums[:-n]</code>	All but the last <code>n</code> elements	<code>nums[:-3]</code>	<code>[0,1,2,3,4,5,6]</code>

Index practice

- **Elvis**

- ['E', 'l', 'v', 'i', 's']

- **Levi's**

- Start with []

- How do you get this:

- ['L', 'e', 'v', 'i', 's']

- From:

- ['E', 'l', 'v', 'i', 's']

- **Code it**



Conclusions

- **Lists are Python's flexible, ordered, mutable container**
 - mix types, nest lists, grow and shrink freely
 - you can build well-structured data with them: parallel lists, 2D lists, lists of records
 - slicing lets you pull out exactly the sub-part of a list you need
- **Lists sit inside a bigger family of aggregate data structures**
 - tuples, sets, dictionaries and more each trade off differently – you'll meet these soon
- **Next steps**
 - practice the common list methods and operations until they're second nature
 - start noticing when a list is the right tool – and when another structure might fit better

Thanks!