

LECTURE 3.2

Lists and tuples

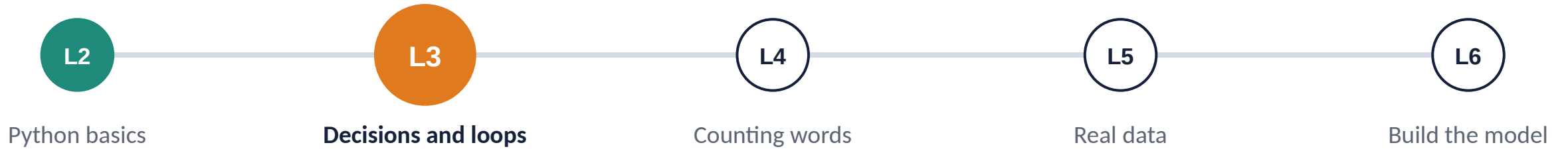
Storing many values: a whole inbox in one variable

Today's key line of code

```
inbox = ["Hi", "Win cash", "Class at 10"]  
print(inbox[1])
```

Where we are: Lecture 3, part 2 of 3

Part 2 of 3



Today you will

- 1 Create lists and read items by position
- 2 Slice out parts of a list
- 3 Add, change and remove items
- 4 Use tuples for fixed records, and unpack them
- 5 Avoid the two-names-one-list trap

Why lists?

- One variable per message does not scale to 5,000 messages
- A list keeps many values, in order, under one name
- `len()` counts the items
- Lists are written with square brackets

Code

```
msg1 = "Win a free phone"
msg2 = "Class moved to 305"
msg3 = "Claim your prize"
inbox = [msg1, msg2, msg3]
print(inbox)
print(len(inbox))
```

Output

```
['Win a free phone', 'Class moved to 305', 'Claim your prize']
3
```

Anatomy of a list

Code

```
scores = [72, 85, 90]  
print(scores)
```

Output

```
[72, 85, 90]
```

The name of the list

[opens the list

Items: any values, here numbers

Commas separate the items

] closes the list

Positions start at 0

index from the start

0	1	2	3
"Win cash"	"Class 305"	"Claim prize"	"See you"
-4	-3	-2	-1

index from the end

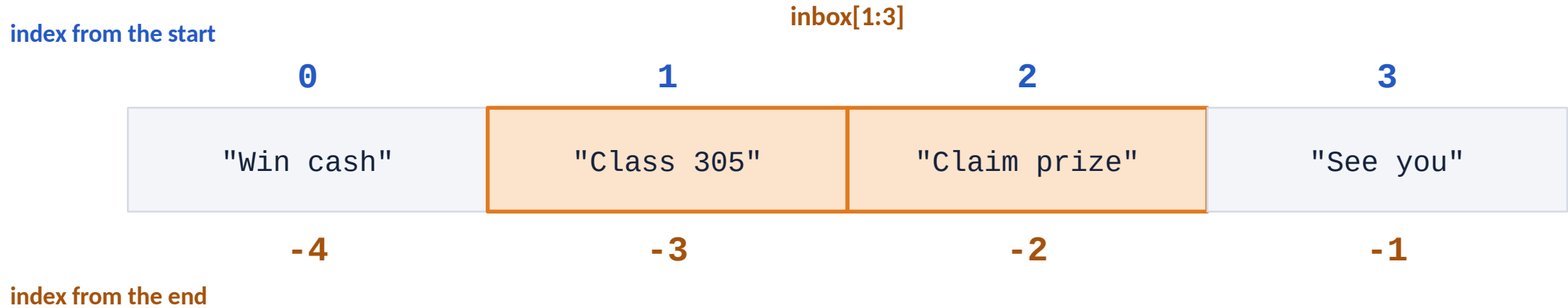
Code

```
inbox = ["Win cash", "Class 305",  
         "Claim prize", "See you"]  
print(inbox[0])  
print(inbox[2])  
print(inbox[-1])  
print(inbox[-2])
```

Output

```
Win cash  
Claim prize  
See you  
Claim prize
```

Slicing: take a part of a list



Code

```
print(inbox[1:3])
print(inbox[:2])
print(inbox[2:])
```

Output

```
['Class 305', 'Claim prize']
['Win cash', 'Class 305']
['Claim prize', 'See you']
```

Predict the slices

Predict first

Code

```
nums = [10, 20, 30, 40, 50]
print(nums[1:4])
print(nums[-3:])
print(len(nums[2:]))
```



What will this print? Write your prediction as a comment before you run the cell.

Answer: Predict the slices

Code

```
nums = [10, 20, 30, 40, 50]
print(nums[1:4])
print(nums[-3:])
print(len(nums[2:]))
```

Output

```
[20, 30, 40]
[30, 40, 50]
3
```

- `[1:4]` takes positions 1, 2, 3: stop before 4
- `[-3:]` means the last three items
- A slice is itself a list, so `len()` works on it

Changing a list

- `append(x)` adds to the end
- `insert(i, x)` adds at position `i`
- `list[i] = x` replaces an item
- `remove(x)` deletes the first `x`
- `pop()` removes and returns the last item

Code

```
tasks = ["read", "code"]
tasks.append("sleep")
tasks.insert(0, "eat")
tasks[1] = "study"
tasks.remove("code")
last = tasks.pop()
print(tasks)
print(last)
```

Output

```
['eat', 'study']
sleep
```

Watching the list change, step by step

Each row runs one line. Follow how the list looks after it.

Code

```
tasks = ["read", "code"]
tasks.append("sleep")
tasks.insert(0, "eat")
tasks[1] = "study"
tasks.remove("code")
last = tasks.pop()
```

step	operation	tasks after this line
1	create	['read', 'code']
2	append	['read', 'code', 'sleep']
3	insert	['eat', 'read', 'code', 'sleep']
4	replace [1]	['eat', 'study', 'code', 'sleep']
5	remove	['eat', 'study', 'sleep']
6	pop -> 'sleep'	['eat', 'study']

Click to reveal the next row. Predict it first!

Asking questions about a list

- `len`, `max`, `min`, `sum` work on whole lists
- `x in list` checks membership
- `count(x)` and `index(x)` find items
- `sorted(list)` gives a new sorted list; the original stays

Code

```
nums = [5, 3, 8, 3, 1]
print(len(nums), max(nums), min(nums), sum(nums))
print(8 in nums)
print(nums.count(3))
print(nums.index(8))
print(sorted(nums))
print(nums)
```

Output

```
5 8 1 20
True
2
2
[1, 3, 3, 5, 8]
[5, 3, 8, 3, 1]
```

sort() versus sorted()

Predict first

Code

```
nums = [5, 3, 8]
result = nums.sort()
print(nums)
print(result)
```



What will this print? Write your prediction as a comment before you run the cell.

Answer: `sort()` versus `sorted()`

Code

```
nums = [5, 3, 8]
result = nums.sort()
print(nums)
print(result)
```

Output

```
[3, 5, 8]
None
```

- `.sort()` sorts the list itself, in place
- It returns nothing, which Python shows as `None`
- `sorted(nums)` returns a new sorted list instead
- Never write `nums = nums.sort()`: you lose the list

Strings behave like lists, almost

- Strings have positions and slices too
- `len()` counts characters
- But strings cannot be changed in place
- `word[0] = "J"` gives an error (see the mistakes slide)

Code

```
word = "PYTHON"  
print(word[0], word[-1])  
print(word[1:4])  
print(len(word))
```

Output

```
P N  
YTH  
6
```

Anatomy of a tuple

Code

```
student = ("Nadia", 21, "CSE")  
print(student[0])  
print(len(student))
```

Output

```
Nadia  
3
```

Round brackets make a tuple

Items, read with positions like a list

Commas separate items

Indexing works the same as lists

Tuples cannot be changed

- Reading works; changing gives a `TypeError`
- Use tuples for fixed records: a student, a date, a coordinate
- Fixed means safe: no code can change it by accident
- Functions use tuples to return several values (Lecture 4.1)

Code

```
student = ("Nadia", 21, "CSE")  
print(student[1])  
student[1] = 22
```

Output

```
21  
TypeError: 'tuple' object does not support item assignment
```

Unpacking: one tuple, several names

- `name, age, dept = student` gives each item a name
- The number of names must match the number of items
- `a, b = b, a` swaps two values in one line
- You will use this constantly with functions

Code

```
name, age, dept = student
print(name)
print(age)
a, b = 1, 2
a, b = b, a
print(a, b)
```

Output

```
Nadia
21
2 1
```

List or tuple?

List []

- Square brackets
- Can change: append, remove, replace
- For collections that grow: messages, scores
- Example: an inbox

Tuple ()

- Round brackets
- Fixed once created
- For records: a student, a coordinate
- Example: several values returned by a function

Careful: two names, one list

- `b = a` does not copy the list
- Both names point to the same list
- Change it through `b`, and `a` sees the change
- Use `a.copy()` for a real copy

Code

```
a = [1, 2, 3]
b = a
b.append(4)
print(a)
c = a.copy()
c.append(5)
print(a)
print(c)
```

Output

```
[1, 2, 3, 4]
[1, 2, 3, 4]
[1, 2, 3, 4, 5]
```

Watch it run: changing a list, and two names

WATCH FOR

- append, replace and pop change the list
- pop hands the removed item back
- b = tasks: the note says 'same object'
- b.append changes what tasks shows too

WATCH THE CODE RUN, STEP BY STEP

A list changing, and two names for one list

Watch the list change after each line.

Then watch the green note: two names, one list.

Click the video to play, click again to pause. Silent: talk the class through it.

Common mistakes with lists and tuples

Read the last line of an error first. It names the problem.

What you typed	What Python says	What it means
<pre>print(inbox[10])</pre>	<code>IndexError: list index out of range</code>	No position 10. Valid positions: 0 to <code>len(inbox) - 1</code> .
<pre>student[1] = 22</pre>	<code>TypeError: 'tuple' object does not support item assignment</code>	Tuples cannot be changed. Use a list, or build a new tuple.
<pre>nums.remove(99)</pre>	<code>ValueError: list.remove(x): x not in list</code>	99 is not in the list. Check with 99 in <code>nums</code> first.
<pre>word[0] = "J"</pre>	<code>TypeError: 'str' object does not support item assignment</code>	Strings cannot be changed. Build a new one: <code>"J" + word[1:]</code> .

In-class exercises

Work in pairs. 15 minutes. No loops yet: use indexing, slicing and list functions.

1 Second largest

Use `sorted()` to print the second largest of `[4, 9, 2, 7]`.
Expected: 7. Bonus: what goes wrong with `[4, 9, 9, 2]`?

2 Ends of the inbox

Print the first two and the last two messages of a 6-message list, using slices.

3 Swap ends

Swap the first and last items of a list. Test: `[1, 2, 3, 4]` becomes `[4, 2, 3, 1]`.

4 Student record

Store (name, roll, cgpa) as a tuple, unpack it, and print one f-string sentence.

Solutions

Show these after students have tried the exercises.

Solutions 1 and 2

The bonus shows a real limitation: with a repeated maximum, `[-2]` returns the maximum again.

Code

```
nums = [4, 9, 2, 7]
print(sorted(nums)[-2])
print(sorted([4, 9, 9, 2])[-2])

inbox = ["m1", "m2", "m3", "m4", "m5", "m6"]
print(inbox[:2])
print(inbox[-2:])
```

Output

```
7
9
['m1', 'm2']
['m5', 'm6']
```

Solutions 3 and 4

Swapping uses tuple unpacking. Unpacking gives each field of a record its own name.

Code

```
items = [1, 2, 3, 4]
items[0], items[-1] = items[-1], items[0]
print(items)

record = ("Nadia", 1042, 3.71)
name, roll, cgpa = record
print(f"{name} (roll {roll}) has CGPA {cgpa}")
```

Output

```
[4, 2, 3, 1]
Nadia (roll 1042) has CGPA 3.71
```

Interactive lab: Lists Lab

In the lab you can

1 List playground

Append, insert, remove and pop: see each box change.

2 Slicing

Drag start, stop and step and watch which items are taken.

3 Ask the list

Edit marks: len, max, sum, sorted and a live chart.

4 Two names, one list

Arrows show when b = a shares one list.

LECTURE 3.2 · INTERACTIVE LAB

Lists Lab

Change a list item by item, slice it any way you like, and see when two names share one list

Introduction to Data Science and Machine Learning
Md Mahbub E Noor · University of Barishal
[All Labs](#)

1 List playground2 Slicing3 Ask the list4 Two names, one list

List playground: change a list one operation at a time

Each button runs one line of real Python on the list. Watch which box changes, what the method gives back, and the error you get when an operation is impossible.

Start with Slide example: ["read", "code"]

Undo↻ Reset

0123

tasks = ['eat', 'read', 'code', 'sleep']

-4-3-2-1

x = "eat"i = 0

x is any Python value: text needs quotes.

CHANGE THE LIST

append(x)insert(i, x)tasks[i] = xremove(x)

pop()pop(i)sort()reverse()

ASK A QUESTION (THE LIST STAYS THE SAME)

len(tasks)x in tasksex.index(x)count(x)

tasks[i]sorted(tasks)

YOUR CODE SO FAR (READY TO PASTE INTO JUPYTER)

```
1 tasks = ["read", "code"]
2 tasks.append("sleep") # returns None
3 tasks.insert(0, "eat") # returns None
```

TRACE: THE LIST AFTER EACH LINE

Step	Line	tasks after this line	Gives back
1	tasks = ["read", "code"]	['read', 'code']	
2	tasks.append("sleep")	['read', 'code', 'sleep']	None
3	tasks.insert(0, "eat")	['eat', 'read', 'code', 'sleep']	None

Open the lab ►

Works offline in any browser

Labs_open_in_browser/
L3.2_Lists_Lab.html

Today's toolkit

<code>[a, b, c]</code>	Make a list
<code>x[0], x[-1]</code>	First item, last item
<code>x[1:3]</code>	Items 1 and 2 (stop before 3)
<code>append / insert</code>	Add items
<code>remove / pop</code>	Delete items

<code>sorted(x) / x.sort()</code>	New sorted list / sort in place
<code>(a, b, c)</code>	Make a tuple (fixed)
<code>n, a, d = t</code>	Unpack a tuple
<code>x.copy()</code>	A real, separate copy

Homework 3.2

Due before Lecture 3.3 • [date]

1 Predict, then run

Predict first: `x = [3, 1, 2]; print(x[::-1]); print(x[5:]); print(sorted(x) == x)`.

2 Grade book

Make a list of 5 marks. Print the highest, lowest, average (sum / len) and the marks sorted.

3 Inbox editor

Start with 3 messages. Add 2, replace the second, remove one by value, and print the list after each step.

4 Trace it

Draw a trace table (step, list after) for the Inbox editor code by hand, then check it by printing.

5 Explain

In 2 sentences: why does `b = a` followed by `b.append(4)` change `a`?

Submit on [LMS / Google Classroom]: your notebook file `HW3_2_<your ID>.ipynb`, or a Colab link set to “Anyone with the link can view”.

Recap: Lecture 3.2 in 90 seconds

RECAP

Lecture 3.2 in 90 seconds

A short recap of today's lecture. Pause any time.

Click to play or pause. Watch it again at home to revise.