1.2

Lists & Tuples

[10] Introductions (icebreaker)

Lesson Plan

- [10] Icebreaker
- [25] Practice: Which Loop?
- [10] Mutation
- [20] Practice: reverse a list
- [10] Passing lists to functions
- [5] Shallow vs deep copy
- [10] Accumulator Pattern & Built-in Functions
- [10] Tuples & Unpacking

Recap

Range For-Loops

```
for i in range(3):
  print(i)

for i in range(0, 3):
  print(i)

for i in range(0, 3, 1):
  print(i)
```

While Loops

```
answer = ''
while answer != 'y':
  answer = input('quit? (y/n): ')
print('goodbye!')
```

For-Each Loops

```
word = 'hello'
for letter in word:
  print(letter)

lst = ['hey', 'sup', 'hi']
for word in lst:
  print(word)

sentence = 'Welcome to tech exchange!'
for word in sentence.split():
  print(word)
```

Enumerate

```
word = 'hello'
for i, letter in enumerate(word):
  print(f'The {i}th letter is {letter}')
```

Practice: Which Loop?

Practice Problem: Loops

You will be working in teams of 2 or 3. The goal is to collaboratively find a solution and be able to explain it to the class. Use the table below to figure out what your role is.

Role	**Responsibilities**	**Assignment Criteria**
Driver	Copy and share the repl.it, write the code, make sure you're listening to ideas from your teammates	Whoever is closest to Google's Mountain View campus
Tester	Play devil's advocate, thinks of edge cases, write unit tests for the driver's code	Second closest to Google's Mountain View campus
Presenter	Document the code, be prepared to present the team's design decisions, and share one thing the team learned from the problem	Furthest from Google's Mountain View campus

If there are only 2 members in your team, the tester will also take on the presenter role.

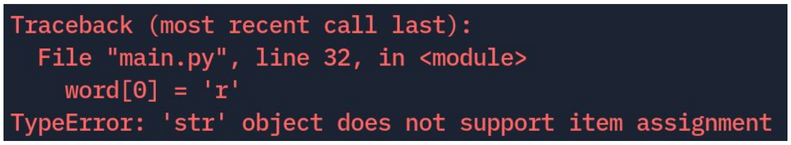
Mutation

Strings are Immutable!

In Python:

- Lists are **mutable** - you can change a list’s elements
- Strings are **immutable** - you can’t change the letters in a string

```
word = 'facecar'
word[0] = 'r'
```

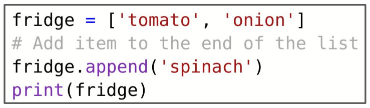


```
Traceback (most recent call last):
  File "main.py", line 32, in <module>
    word[0] = 'r'
TypeError: 'str' object does not support item assignment
```

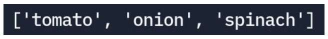
Python lists can change length!

Similar to:

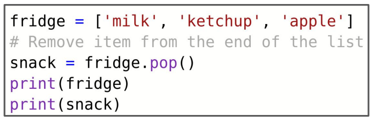
- an **ArrayList** in Java
- a **vector** in C++



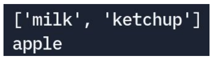
```
fridge = ['tomato', 'onion']
# Add item to the end of the list
fridge.append('spinach')
print(fridge)
```



```
['tomato', 'onion', 'spinach']
```



```
fridge = ['milk', 'ketchup', 'apple']
# Remove item from the end of the list
snack = fridge.pop()
print(fridge)
print(snack)
```



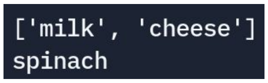
```
['milk', 'ketchup']
apple
```

Deleting an item from the middle shifts the rest!

```
fridge = ['milk', 'spinach', 'cheese']
gross_food = fridge.pop(1)
print(fridge)
print(gross_food)
```

What is the original index of ‘cheese’?

Which gross food was removed?



```
['milk', 'cheese']
spinach
```

What is the new index of ‘cheese’?

Can use e.g. **del fridge[1]** to delete the food at position 1 if you don’t care about checking which food was deleted

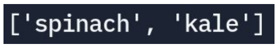
```
def remove_spinach(fridge):
  for i, food in enumerate(fridge):
    if food == 'spinach':
      fridge.pop(i)
```

What is wrong with this function?

Another example

```
def remove_spinach(fridge):
  for i, food in enumerate(fridge):
    if food == 'spinach':
      fridge.pop(i)

fridge = ['spinach', 'spinach', 'kale']
remove_spinach(fridge)
print(fridge)
```

What do you think this prints?



```
['spinach', 'kale']
```

A better approach

```
def remove_spinach(fridge):
    tasty_foods = []
    for food in fridge:
        if food != 'spinach':
            tasty_foods.append(food)
    fridge.clear()
    fridge.extend(tasty_foods)
```

1. Create a new list to store the items you're **keeping**

2. Add keepers to the list

3. Clear the original list

4. Add all the keepers to the original list

“Mutating” function

```
def remove_spinach(fridge):
    tasty_foods = []
    for food in fridge:
        if food != 'spinach':
            tasty_foods.append(food)
    fridge.clear()
    fridge.extend(tasty_foods)
```

This function **modifies the list “in-place”**, meaning it actually mutates the input list.

In an interview, you'll want to clarify **whether you should modify your input**.

“Non-mutating Function”

```
def remove_spinach(fridge):
    tasty_foods = []
    for food in fridge:
        if food != 'spinach':
            tasty_foods.append(food)
    return tasty_foods
```

This function **does not modify** the input list

Instead, it **returns a new list** that is identical to the old one, but without any spinach

What does this mean practically?

If you pass a mutable object (e.g. a list) to a function, **you can modify the object**

- lst.clear()
- lst[0] = 3
- lst.append(5)

However, you **can't modify what variables outside the function point to**

- lst = []
- swapping two variables

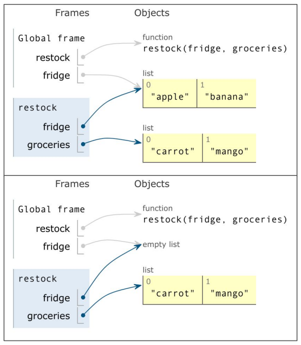
You also can't modify the values of any **immutable objects**

- Strings, ints, Booleans, etc.

Key Point: Use Python Tutor to visualize code execution!

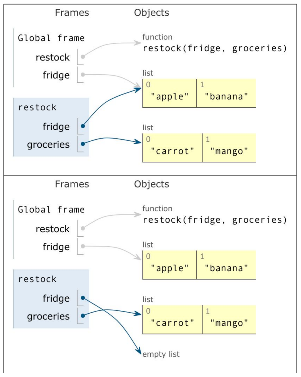
Example A

```
def restock(fridge, groceries):
  fridge.clear()
  fridge.extend(groceries)

fridge = ['apple', 'banana']
restock(fridge, ['carrot', 'mango'])
```



Example B

```
def restock(fridge, groceries):
  fridge = []
  fridge.extend(groceries)

fridge = ['apple', 'banana']
restock(fridge, ['carrot', 'mango'])
```



Python uses “Call by Object Reference” (Docs)

Also known as “call by sharing”

- Call by value where the value is a reference

Each function call creates a new local symbol table

- Each variable in the symbol table stores a reference (pointer) to an object.
- You can visualize the symbol table using Python Tutor

Practice: reverse a list

Reverse a list “mutating” vs “non-mutating”

In your Breakout Rooms:

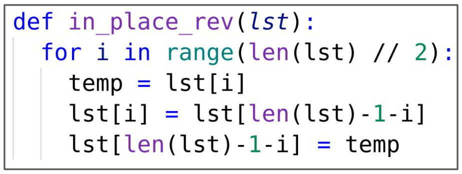
Reverse a list with a mutating function

- Don’t create a new list
- Hint: which element pairs will you be swapping?

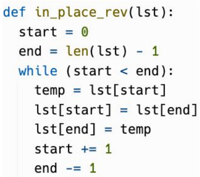
Reverse a list with a non-mutating function

- Create a new list
- Hint: to iterate backwards, what should the 3 arguments to range() be?

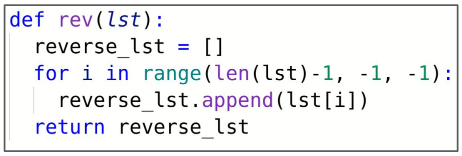
Example Solutions



```
def in_place_rev(lst):
  for i in range(len(lst) // 2):
    temp = lst[i]
    lst[i] = lst[len(lst)-1-i]
    lst[len(lst)-1-i] = temp
```



```
def in_place_rev(lst):
  start = 0
  end = len(lst) - 1
  while (start < end):
    temp = lst[start]
    lst[start] = lst[end]
    lst[end] = temp
    start += 1
    end -= 1
```



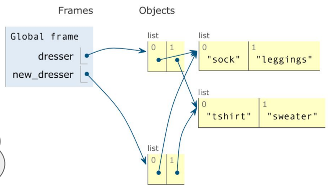
```
def rev(lst):
  reverse_lst = []
  for i in range(len(lst)-1, -1, -1):
    reverse_lst.append(lst[i])
  return reverse_lst
```

Shallow vs Deep Copy

Shallow Copy

```
dresser = [['sock', 'leggings'], ['tshirt', 'sweater']]
new_dresser = dresser.copy()
```

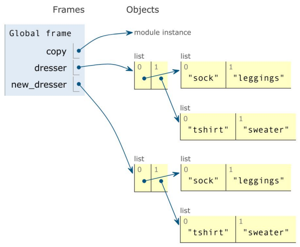
If I change dresser[0], does new_dresser[0] change?



When should you use shallow copy?

Deep Copy

```python
import copy

dresser = [['sock', 'leggings'], ['tshirt', 'sweater']]
new_dresser = copy.deepcopy(dresser)
```



If I change dresser[0], does new_dresser[0] change?

When should you use deep copy?

Accumulator Pattern

- Counting
- Totalling
- Min/Max
- Building a List
- Building a String

Accumulator Pattern

This is the pattern by which you:

- Set a variable equal to an initial value
- For each item in the list
    - Possibly update the variable based on the item

Chat Waterfall:
What are some examples of the accumulator pattern that you've seen before?

Examples:

- Counting the number of times something occurs
- Keeping track of a total
- Finding the min or max
- Building a new list of items that match a certain filter
- Building a string

Counting

```python
def count_spinach(foods):
  spinach_count = 0
  for food in foods:
    if food == 'spinach':
      spinach_count += 1
  return spinach_count
```

```python
def count_spinach(foods):
  return foods.count('spinach')
```

Lists have a count() function!

Totalling

```python
def total_cost(prices):
  total = 0
  for price in prices:
    total += price
  return total
```

```python
def total_cost(prices):
  return sum(prices)
```

Python has a built-in sum() function!

Min and Max

```python
import math

def minimum(nums):
  smallest = ?
  for num in nums:
    if num ? smallest:
      smallest = num
  return smallest
```

```python
import math

def maximum(nums):
  biggest = ?
  for num in nums:
    if num ? biggest:
      biggest = num
  return biggest
```

Min and Max

Python has built-in min() and max() functions!

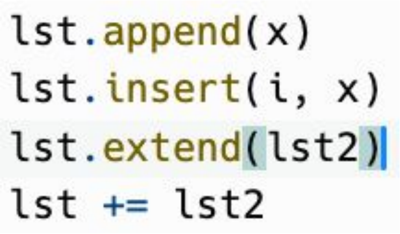
```
def minimum(nums):
  return min(nums)

def maximum(nums):
  return max(nums)
```

Building a list

```
def square(nums):
  squares = []
  for num in nums:
    squares.append(num**2)
  return squares
```

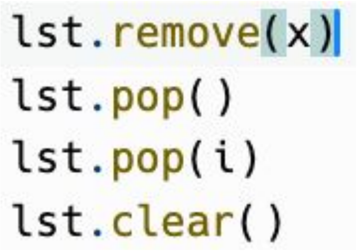
Other Helpful List Methods

Adding Elements to a List



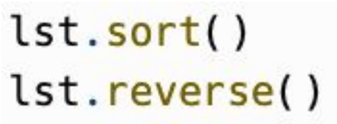
```
lst.append(x)
lst.insert(i, x)
lst.extend(lst2)
lst += lst2
```

Removing Elements from a List



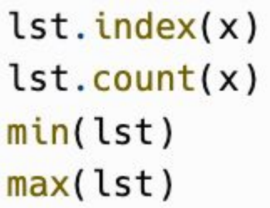
```
lst.remove(x)
lst.pop()
lst.pop(i)
lst.clear()
```

Modifying a List



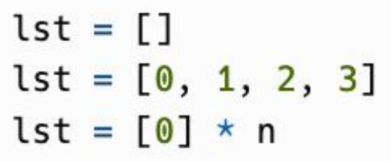
```
lst.sort()
lst.reverse()
```

Checking Elements of a List



```
lst.index(x)
lst.count(x)
min(lst)
max(lst)
```

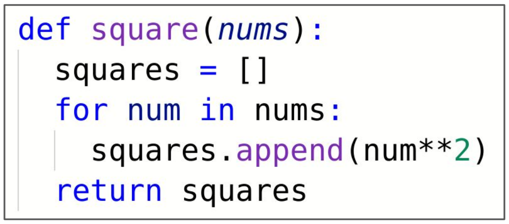
Initializing a List



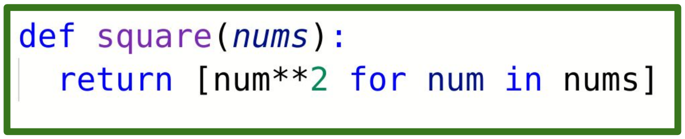
```
lst = []
lst = [0, 1, 2, 3]
lst = [0] * n
```

List Comprehensions

List Comprehensions - great for building lists!

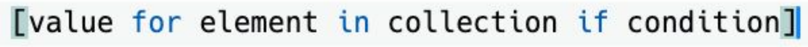


```
def square(nums):
  squares = []
  for num in nums:
    squares.append(num**2)
  return squares
```



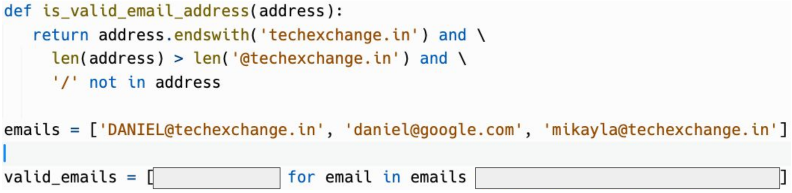
```
def square(nums):
  return [num**2 for num in nums]
```

Generalized List Comprehension



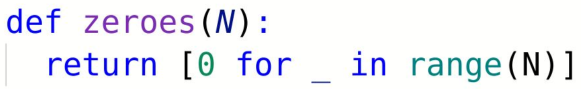
```
[value for element in collection if condition]
```

Generalized List Comprehension Example



```
def is_valid_email_address(address):
    return address.endswith('techexchange.in') and \
      len(address) > len('@techexchange.in') and \
      '/' not in address

emails = ['DANIEL@techexchange.in', 'daniel@google.com', 'mikayla@techexchange.in']

valid_emails = [              for email in emails                                ]
```

Build a list containing N zeroes



```
def zeroes(N):
  return [0 for _ in range(N)]
```

Use an underscore when you don't care about the value!

Tuples & Unpacking

Unpacking Tuples

```
circle = (2, 3, 4)
```

```
x = circle[0]
y = circle[1]
r = circle[2]
```

You can access elements in a tuple like you would in a list…

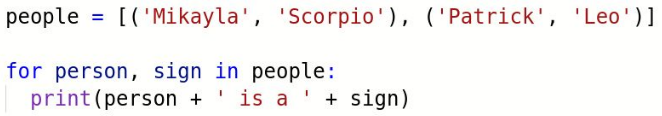
```
x, y, r = circle
```

… but it's often clearer to unpack a tuple this way

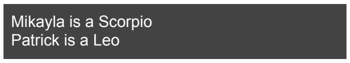
Use an underscore when you don't care about the value!

```
_, _, r = circle
area = math.pi * r**2
```

Iterating through a list of Tuples

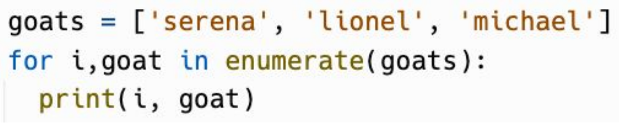


```
people = [('Mikayla', 'Scorpio'), ('Patrick', 'Leo')]

for person, sign in people:
  print(person + ' is a ' + sign)
```



```
Mikayla is a Scorpio
Patrick is a Leo
```

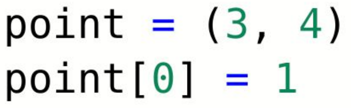
We've seen tuples before…



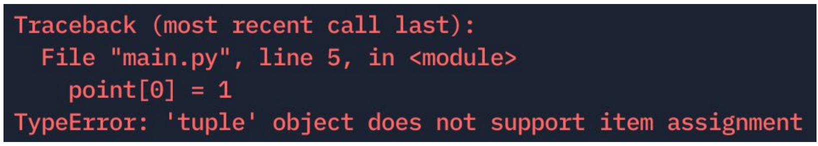
```
goats = ['serena', 'lionel', 'michael']
for i,goat in enumerate(goats):
  print(i, goat)
```

(0, 'serena')
(1, 'lionel')
(2, 'michael')

Tuples are Immutable!



```
point = (3, 4)
point[0] = 1
```



```
Traceback (most recent call last):
  File "main.py", line 5, in <module>
    point[0] = 1
TypeError: 'tuple' object does not support item assignment
```

Feedback / Attendance

Bonus Material: filter, map, zip, & lambda functions

https://docs.python.org/3/library/functions.html

- filter
- map
- zip
- lambda functions
- ternary: x if condition else y
- Practice using min, max, sum with the above (possibly also count)