Reinforcement

Andey Robins & Dr Borowczak

March 7, 2023



Outline

- Creating Lists
- ► Reading Lists
- Modifying Lists
- Slices
- ▶ for Loops
- ► List Methods

Creating Lists

Pre-filled Lists

We've seen this a lot, how do we construct a list of the numbers from 2 to 4?

```
two_to_four = [2, 3, 4]
```

List Syntax

```
<Variable name> = [ <value> , <value> ]
```

Empty Lists

```
<Variable name> = [ ]
```

Why Empty Lists?

We make empty lists for a few reasons:

- 1. To later update
- 2. As a placeholder
- 3. To check if a list is empty

Reading Lists

Elements

```
ex_list = ['a', 'b', 'c', 'd', 'e']
```

Positive Index	0	1	2	3	4
Negative Index	-5	-4	-3	-2	-1
List Item	'a'	'b'	'c'	'd'	'e'

Slices

In python notation:

my_list[start:end]

In interval notation:

[start, end)

In English:

All elements in my_list from start up to but not including end.

Why?

What does this give us?

my_list[0:len(my_list)]

```
my_list[0:len(my_list)] == my_list
```

Helpful Slices

```
my_list = [1, 2, 3, 4, 5, 6]
without_last = my_list[:-1]
without_first = my_list[1:]
without_ends = my_list[1:-1]
only_first = my_list[:1]
only_last = my_list[-1:]
```

What's the Difference?

```
only_first = my_list[:1]
# print(only_first) -> ???
only_first = my_list[0]
# print(only_first) -> ???
```

```
only_first = my_list[:1]
# print(only_first) -> [1]
only_first = my_list[0]
# print(only_first) -> 1
```



Update an Element

```
my_list = [0, 1, 2, 3, 4]
my_list[3] = 'a'
print(my_list) # ???
```

```
my_list = [0, 1, 2, 3, 4]
my_list[3] = 'a'
print(my_list) # [0, 1, 2, 'a', 4]
```

Updating Slices

```
my_list = [0, 1, 2, 'a', 4]

my_list[3:5] = ['b', 'c']

print(my_list) # What does this print?
```

```
my_list = [0, 1, 2, 'a', 4]
my_list[3:5] = ['b', 'c']
print(my_list) # [0, 1, 2, 'b', 'c']
```

Updating Strings

Indices work the same in strings and lists. That's why we sometimes say strings are like lists of characters. We'll now examine how they aren't exactly alike.

```
my_string = "hiya"
```

print(my_string[3]) # ???

```
my_string = "hiya"
print(my_string[3]) # "a"
```

```
my_string = "hiya"
```

print(my_string[-2]) # ???

```
my_string = "hiya"
print(my_string[-2]) # "y"
```

```
my_string = "hiya"
print(my_string[1:4]) # ???
```

```
my_string = "hiya"
print(my_string[1:4]) # "iya"
```

```
my_string[1] = "o"
```

my_string = "hiya"

print(my_string) # ???

```
my_string = "hiya"
my_string[1] = "o" # ERROR
```

print(my_string)

Traceback (most recent call last):
 File "<stdin>", line 1, in <module>

TypeError: 'str' object does not support item assignment

Slices

What Does This Do?

```
my_string = "this is a string."

new_string = my_string[0].upper() + my_string[1:]

print(new_string) # ???
```

```
my_string = "this is a string."

new_string = my_string[0].upper() + my_string[1:]

print(new_string) # "This is a string."
```

```
my_string = "this is a string."

new_string = my_string[:-1] + "!"

print(new_string) # ???
```

```
my_string = "this is a string."
new_string = my_string[:-1] + "!"
```

print(new_string) # "this is a string!"

print(new_string) # ???

print(new_string) # "THIS IS My string."



Syntax

for variable in list:
 # loop code here

```
for num in [1, 2, 3, 4, 5]:
    print(num)
```

```
sum_val = 0
for num in my_list:
    sum_val += num
```

print(sum_val)

A Simpler Approach to Last Lab

```
translation = ""
for word in sentence.split():
   translation += en2tp[word] + " "
...
```

Loop Equivalence

The loops presented on the next two slides are semantically equivalent.

While Loop

```
my_list = [1, 2, 3, 4, 5, 6, 7]

idx = 0
while idx < len(my_list):
    print(my_list[idx])
    idx += 1</pre>
```

For Loop

```
my_list = [1, 2, 3, 4, 5, 6, 7]
for num in my_list:
    print(num)
```

List Methods



append(val) takes one argument, an item, and adds it to the list

```
fruit = ["apples", "bananas", "cherries"]
fruit.append("dates")
print(fruit)
# ["apples", "bananas", "cherries", "dates"]
```

pop

 $\ensuremath{\mathsf{pop}}$ () removes the last item from the list.

```
fruit = ["apples", "bananas", "cherries"]
fruit.pop()
print(fruit) # ["apples", "bananas"]
```

insert

insert(idx, val) takes two arguments: the index to insert the
value at and the value to insert.

```
fruit = ["apples", "cherries"]
fruit.insert(1, "bananas")
print(fruit) # ["apples", "bananas", "cherries"]
```

What Does This Do?

```
fruit = ["apples", "cherries"]
fruit.inesert(100, "bananas")
print(fruit) # ???
```

```
fruit = ["apples", "cherries"]
fruit.inesert(100, "bananas")
print(fruit) # ["apples", "cherries", "bananas"]
```

clear

clear() removes all items from a list.

```
fruit = ["apples", "bananas", "cherries"]
fruit.clear()
print(fruit) # []
```



index(val) returns the index of the item given as an argument.

```
fruit = ["apples", "bananas", "cherries"]
print(fruit.index("bananas")) # ???
```

print(fruit.index("dates")) # ???

```
fruit = ["apples", "bananas", "cherries"]
print(fruit.index("bananas")) # 1
```

print(fruit.index("dates")) # ERROR

Traceback (most recent call last):
 File "<stdin>", line 1, in <module>
ValueError: 'dates' is not in list

reverse

reverse() reverses the list.

```
fruit = ["apples", "bananas", "cherries"]
fruit.reverse()
print(fruit) # ["cherries", "bananas", "apples"]
```

sort

 ${\tt sort}$ () sorts the list in lexicographic order.

```
fruit = ["apples", "bananas", "cherries"]
fruit.sort()
print(fruit) # ["apples", "bananas", "cherries"]
```

```
fruit = ["dates", "bananas", "cherries", "apples"]
fruit.sort()
print(fruit)
# ["apples", "bananas", "cherries", "dates"]
```

An Example

Imagine we are a fruit merchant wanting to standardize our item list. Currently it's random and mixed case. We want to output everything as an all capitals, alphabetical list.

```
mixed_fruit = ["Bananas", "APPLES", "cherries", "dAtEs"]
output_fruit = []
for fruit in mixed_fruit:
  output_fruit.append(fruit.upper())
```

output_fruit.sort()
print(output_fruit)



Reinforcement

Outline

- ▶ for loop setups
- ► Example List Problem
- ► Text Adventure Updates



Building Structures

```
sentence = "someone is good"
translation = ""
for word in sentence.split():
   translation += en2tp[word]
   translation += " "
print(translation)
```

```
# first loop
translation == "jan "
# second loop
translation == "jan li "
# third loop
translation == "jan li pona "
```

Takeaway

When we move through one collection, we can build another collection simultaneously.

Another Example

Given a piece of text, count the frequency of each letter.

Setup

We will use an array with 26 spots to count the Latin letter frequency.

```
frequency[0] # count of 'a'
frequency[3] # count of 'd'
frequency[25] # count of 'z'
```

```
letter_freq = []
idx = 0
while idx < 26:
  letter_freq.append(0)
  idx += 1</pre>
```

letter_freq == [0, 0, 0, 0, ..., 0]

Iteration

This prints out each individual word.

```
for word in text.split():
   print(word)
```

This prints out each i	ndividual character.

for word in text.split():

for char in word:

print(char)

Nested for Loops

Nested for loops allow us to look through a collection that is inside a collection.

- We can look at characters in a string, which are in a list.
- ▶ We can look at numbers in a list, which are in a list.
- ▶ We can look at strings in a list, which is in a list, which is in a list,

for word in text.split():
 for char in word:

print(ord(char)-97)

```
>>> ord('a')
97
>>> ord('a')-97
0
>>> ord('b')
```

>>> ord('b')-97

98

1

```
for word in text.split():
  for char in word:
    letter_freq[ord(char)-97] += 1
```

Iterative Output

```
idx = 0
while idx < 26:
    print(chr(idx+97), "-", letter_freq[idx])
    idx += 1</pre>
```

```
a - 12
```

b - 6

y - 2

z - 0

Example Problem

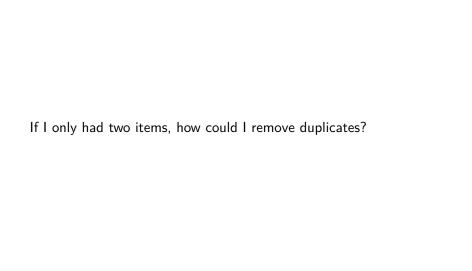


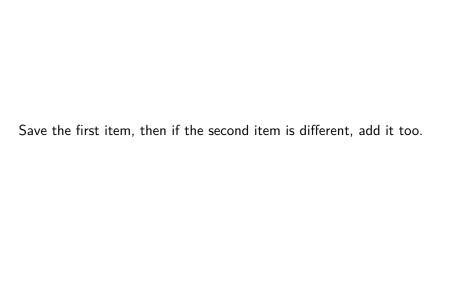
Given a list, write a function which returns a list without duplicates.

Possible Approaches?

Hint

Think in terms of building a collection step by step.





Now, extend this to three items. Assuming I process the first two properly, how do we know if a list of three has duplicates?

f there's only one item, compare the third item to it. If there are wo, compare the third item to the first item and then the second.

t

Therefore, if we have a list of all the unique elements, and we have
a new element, we can compare the new element to each other

element. If we don't see a duplicate, we can add the new element to the list. Repeat this process until we have no more new elements.

```
Outline our function
def remove_dupes(input_list):
   output_list = []
```

return output_list

```
Create the nested loops
def remove_dupes(input_list):
```

return output_list

output_list = []
for potential_item in input_list:
 for item in output_list:
 ...

```
Track if we found the item
def remove_dupes(input_list):
```

output_list = [] for potential_item in input_list:

item_found = False for item in output_list:

. . . return output_list

```
Check if the item is unique
def remove_dupes(input_list):
  output_list = []
  for potential_item in input_list:
    item_found = False
```

return output_list

for item in output_list:

if item == potential_item:
 item_found = True

```
After checking all the saved items, if it hasn't been found, it's
unique! So save it!
def remove dupes(input list):
  output_list = []
  for potential_item in input_list:
    item found = False
    for item in output_list:
      if item == potential item:
        item_found = True
    if not item found:
      output list.append(potential item)
  return output list
```



Goal

We're going to put together everything we've done so far and write a fun little text adventure game. Let's begin by breaking down what we actually need to do.

