Lists

Ruth Anderson
UW CSE 160
Autumn 2022

Lists

- What do we already know about Lists?
- List Operations
  - Creation
  - Querying
  - Modification

Loop Examples: Where's the list?

See in python tutor

```
for num in [2, 4, 6]:
    print(num)

for i in [1, 2, 3]:
    print("Hi there!")

for char in "happy":
    print(char)
```

sequence is a string, NOT a list

Prints the values of sequence

The range function

A typical for loop does not use an explicit list:

```
for i in range(5):
    … body …
```

Produces the list [0, 1, 2, 3, 4]

Upper limit (*exclusive*)

`range(5)`: cycles through [0, 1, 2, 3, 4]

Lower limit (*inclusive*)

`range(1, 5)`: cycles through [1, 2, 3, 4]

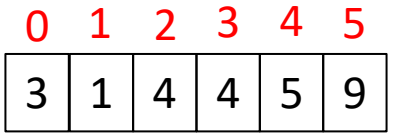
step (distance between elements)

`range(1, 10, 2)`: cycles through [1, 3, 5, 7, 9]

What is a list?

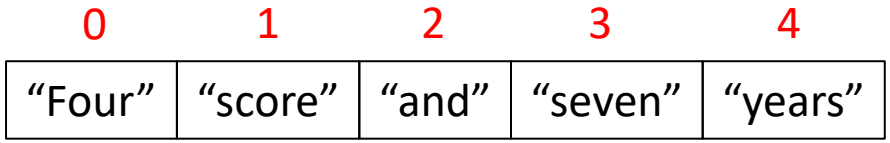
- A list is an ordered sequence of values
  - A list of integers:

```
[3, 1, 4, 4, 5, 9]
```



  - A list of strings:

```
["Four", "score", "and", "seven", "years"]
```



- Each value has an index
  - Indexing is zero-based (counting starts with zero)
- `len([3, 1, 4, 4, 5, 9])` returns `6`

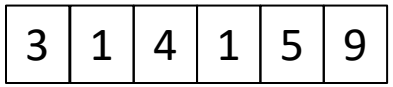
List Operations

- What operations should a list support efficiently and conveniently?
  - Creation
  - Querying
  - Modification

See in python tutor

List Creation

```
a = [3, 1, 2 * 2, 1, 10 / 2, 10 - 1]
```



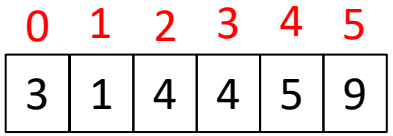

```
b = [5, 3, 'hi']
```

```
c = [4, 'a', a]
```

```
d = [[1, 2], [3, 4], [5, 6]]
```

```
e = [ ]    # An empty list
```

List Querying



Expressions that return parts of lists:

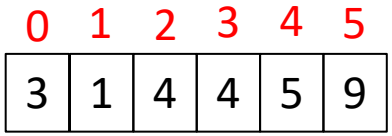
- Single element: **`my_list[index]`**
  - The single element stored at that location

- Sublist ("slicing"): **`my_list[start:end]`**
  - the sublist that starts at index **`start`** and ends at index **`end - 1`**
  - If **`start`** is omitted: defaults to **`0`**
  - If **`end`** is omitted: defaults to **`len(my_list)`**
  - **`my_list[:]`** evaluates to the whole list
  - **`my_list[0:len(my_list)]`** also does

See in python tutor

Indexing and Slicing Examples

```
a = [3, 1, 4, 4, 5, 9]
print(a[0])
print(a[5])
print(a[6])
print(a[-1]) # last element in list
print(a[-2]) # next to last element

print(a[0:2])
print(a[0:-1])
```



```
a = [3, 1, 4, 4, 5, 9]
```

What is printed by: `print(a[1:3])`

A. `[3, 1]`

B. `[3, 1, 4]`

C. `[1, 4]`

D. `[1, 4, 4]`

E. `[1, 2, 3]`

See in python tutor

What python code will print: 9 4 7

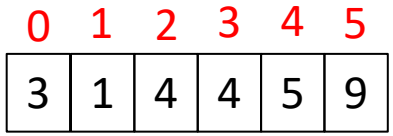
```
a = [2, 7, 3, 9, 4]
```

```
A.print(a[4], a[5], a[2])

B.print(a[3], a[-1], a[1])

C.print(a[4:6], a[2])

D.print(a[9], a[4], a[7])

E.print(a[3], a[5], a[1])
```

See in python tutor

More List Querying

- Find/lookup in a list



  `x in my_list`
    - Returns True if **`x`** is found in **`my_list`**

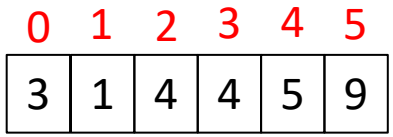
  `my_list.index(x)`
    - Return the integer index in the list of the *first item* whose value is **`x`**.
    - It is an error if there is no such item.

  `my_list.count(x)`
    - Return the number of times **`x`** appears in the list.

See in python tutor

List Querying Examples

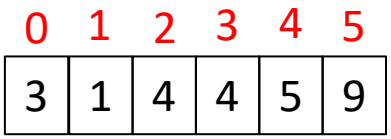
```
a = [3, 1, 4, 4, 5, 9]
print(5 in a)
print(16 in a)
print(a.index(4))
print(a.index(16))
print(a.count(4))
print(a.count(16))
```



List Modification

- Insertion
- Removal
- Replacement
- Rearrangement

List Insertion



- **`my_list.append(x)`**
  - Adds item **`x`** at the end of **`my_list`**
- **`my_list.extend(L)`**
  - Extend **`my_list`** by appending all the items in the argument list **`L`** to the end of **`my_list`**
- **`my_list.insert(i, x)`**
  - Insert item **`x`** *before* position **`i`**.
  - **`a.insert(0, x)`** inserts at the front of the list
  - **`a.insert(len(a), x)`** is equivalent to **`a.append(x)`**

Note: **`append`**, **`extend`** and **`insert`** all return **`None`**

See in python tutor

List Insertion Examples

```
lst = [1, 2, 3, 4]
lst.append(5)
lst.extend([6, 7, 8])
lst.insert(3, 3.5)
```

See in python tutor

What is printed by: `print(lst[2])`

```
lst = [1, 3, 5]
lst.insert(2, [4, 6])
print(lst[2])
```

A. 4

B. 5

C. 3

D. [4, 6]

E. IndexError: list index out of range

List Removal

- **`my_list.remove(x)`**
  - Remove the first item from the list whose value is **`x`**
  - It is an error if there is no such item
  - Returns **`None`**

Notation from the Python Library Reference:
The square brackets around the parameter, "[i]", means the argument is *optional*.
It does *not* mean you should type square brackets at that position.

- **`my_list.pop([i])`**
  - Remove the item at the given position in the list, and return it.
  - If no index is specified, **`a.pop()`** removes and returns the last item in the list.

Note: **`remove`** returns **`None`**

List Replacement

- **`my_list[index] = new_value`**
- **`my_list[start:end] = new_sublist`**
  - Replaces **`my_list[start]`**… **`my_list[end - 1]`** with **`new_sublist`**
  - Can change the length of the list

Examples:

- **`my_list[start:end] = []`**
  - removes **`my_list[start]`**… **`my_list[end - 1]`**
- **`my_list[len(my_list):] = L`**
  - is equivalent to **`a.extend(L)`**

See in python tutor

List Removal & Replacement Examples

```
lst = [1, 2, 3, 4, 5, 6, 7]
print(lst.pop())
print(lst.pop(1))
lst.remove(3)
lst[3] = 'blue'
lst[1:3] = [10, 11, 12]
```

List Rearrangement

- **`my_list.sort()`**
  - Sort the items of the list, **in place**.
  - "in place" means by *modifying the original list*, not by creating a new list.
- **`my_list.reverse()`**
  - Reverse the elements of the list, **in place**.

Note: **`sort`** and **`reverse`** return **`None`**

List Modification Examples

See in python tutor

```
lst = [10, 12, 23, 54, 15]
lst.append(7)
lst.extend([8, 9, 3])
lst.insert(2, 2.75)
lst.remove(3)
print(lst.pop())
print(lst.pop(4))
lst[1:5] = [20, 21, 22]
lst2 = [4, 6, 8, 2, 0]
lst2.sort()
lst2.reverse()
lst3 = lst2
lst4 = lst2[:]
lst2[-1]= 17
```

What will convert a into [1, 2, 3, 4, 5]?

```
a = [1, 3, 5]
```

A.
```
a.insert(1, 2)
a.insert(2, 4)
```

B.
```
a[1:2] = [2, 3, 4]
```

C.
```
a.extend([2, 4])
```

D.
```
a[1] = 2
a[3] = 4
```

See in python tutor

Exercise: list lookup

```
def my_index(lst, value):
    """Return the position of the first occurrence
    of value in the list lst. Return None if value
    does not appear in lst."""
```

Examples:

```
gettysburg = ["four", "score", "and", "seven", "years", "ago"]
print(my_index(gettysburg, "and")) # Should return 2
print(my_index(gettysburg, "years")) # Should return 4
```

Fact: `my_list[my_index(my_list, x)] == x`

See in python tutor

Exercise: list lookup (Answer #1)

```
def my_index(lst, value):
    """Return the position of the first
    occurrence of value in the list lst.
    Return None if value does not appear
    in lst."""
    i = 0
    for element in lst:
        if element == value:
            return i
        i = i + 1
    return None
```

See in python tutor

Exercise: list lookup (Answer #2)

```
def my_index(lst, value):
    """Return the position of the first
    occurrence of value in the list lst.
    Return None if value does not appear
    in lst."""
    for i in range(len(lst)):
        if lst[i] == value:
            return i
    return None
```

See in python tutor

Exercise: Convert Units

```
def cent_to_fahr(cent):
  return cent / 5.0 * 9 + 32

c_temps = [-40, 0, 20, 37, 100]
# Goal:  set f_temps to [-40, 32, 68, 98.6, 212]

f_temps = []
```

See in python tutor

Exercise: Convert Units (Answer)

```
def cent_to_fahr(cent):
  return cent / 5.0 * 9 + 32

c_temps = [-40, 0, 20, 37, 100]
# Goal:  set f_temps to [-40, 32, 68, 98.6, 212]

f_temps = []
for c in c_temps:
    f = cent_to_fahr(c)
    f_temps.append(f)
```

More on List Slicing

`my_list[startindex:endindex]` evaluates to a sublist of the original list

- **`my_list[index]`** evaluates to an element of the original list

- Arguments are like those to the **`range`** function
  - **`my_list[start:end:step]`**
  - start index is inclusive, end index is exclusive
  - *All* 3 indices are *optional*
- Can assign to a slice: **`my_list[s:e] = yourlist`**

See in python tutor

List Slicing Examples

```
test_list = ['e0', 'e1', 'e2', 'e3', 'e4', 'e5', 'e6']
```

```
test_list[2:]

test_list[:5]

test_list[-1]

test_list[-4:]

test_list[:-3]

test_list[:]

test_list[::-1]
```

See in python tutor

Answer: List Slicing Examples

```
test_list = ['e0', 'e1', 'e2', 'e3', 'e4', 'e5', 'e6']
```

`test_list[2:]`	From e2 to the end of the list
`test_list[:5]`	From beginning up to (but not including) e5
`test_list[-1]`	Last element
`test_list[-4:]`	Last four elements
`test_list[:-3]`	Everything except last three elements
`test_list[:]`	Get a copy of the whole list
`test_list[::-1]`	Reverse the list

How to evaluate a list expression

There are two new forms of expression:

- `[a, b, c, d]` list creation
  - To evaluate:
    - evaluate each element to a value, from left to right
    - make a list of the values
  - The elements can be arbitrary values, including lists:
    - `["a", 3, fahr_to_cent(-40), [3 + 4, 5 * 6]]`

Same tokens "`[]`" with two *distinct* meanings

List expression

- `a[b]` list indexing or dereferencing
  - To evaluate:
    - evaluate the **list expression** to a value
    - evaluate the **index expression** to a value
    - if the list value is not a list, execution terminates with an error
    - if the element is not in range (not a valid index), execution terminates with an error
    - the value is the given element of the list value (counting from zero)

Index expression

See in python tutor

List expression examples

What does this mean (or is it an error)?

```
["four", "score", "and", "seven", "years"][2]

["four", "score", "and", "seven", "years"][0,2,3]

["four", "score", "and", "seven", "years"][[0,2,3]]

["four", "score", "and", "seven", "years"][[0,2,3][1]]
```