CptS 355- Programming Language Design

Python Lists

Instructor: Sakire Arslan Ay

Fall 2023

WASHINGTON STATE
UNIVERSITY

World Class. Face to Face.

Lecture material

- Please watch the Python part-1 and part-2 videos on Canvas.
- No lecture notes on Python basics
- Lecture notes on:
  - Python lists
  - Python dictionaries
  - Higher order functions, recursion
  - Classes, iterators, streams

Lists

- List Operations
  - Creation
  - Querying
  - Modification

- Examples adopted from :
  - *“Introduction to Computation and Programming Using Python, Second Edition”*

What is a list?

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

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

0	1	2	3	4	5
3	1	4	4	5	9

  - A list of strings:

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

0	1	2	3	4
“Four”	“score”	“and”	“seven”	“years”

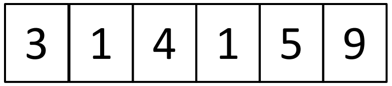
- Each value has an index
  - Indexing is zero-based (counting starts with zero)
- `len([3, 1, 4, 4, 5, 9])` returns `6`

List Operations

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

List Creation

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



3	1	4	1	5	9

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

c = [4, 'a', a]

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

List Querying

0	1	2	3	4	5
3	1	4	4	5	9

Expressions that return parts of lists:

- Single element: `mylist[index]`
  - The single element stored at that location
- Sublist ("slicing"): `mylist[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(mylist)`
  - `mylist[:]` evaluates to the whole list
  - `mylist[0:len(mylist)]` also does

Indexing and Slicing Examples

0	1	2	3	4	5
3	1	4	4	5	9

- `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]`

More List Querying

- Find/lookup in a list

0	1	2	3	4	5
3	1	4	4	5	9

  `x in mylist`

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

  `mylist.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.

  `mylist.count(x)`

  - Return the number of times **`x`** appears in the list.

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))
```

0	1	2	3	4	5
3	1	4	4	5	9

List Modification

- Insertion
- Removal
- Replacement
- Rearrangement

List Insertion

0	1	2	3	4	5
3	1	4	4	5	9

- **`mylist.append(x)`**
  - Extend **`mylist`** by inserting **`x`** at the end
- **`mylist.extend(L)`**
  - Extend **`mylist`** by appending all the items in the argument list **`L`** to the end of **`mylist`**
- **`mylist.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`**

List Insertion Examples

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

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

- **`mylist.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.

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

<u>**Note**</u>: **`remove`** returns **`None`**

List Replacement

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

Examples:

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

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

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

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

sorted function

- Sorted is a built-in function that you can sort lists without changing the list value.
- It returns a new list where the items in the list are sorted.

```
# Sorted doesn't change the original list.
L = [3,7,6,2,1]
sorted (L)
T = (3,7,6,2,1)
sorted (T)

# You can sort it backwards.
sorted (L,reverse = True )
```

Note: **`sorted`** **returns the sorted list**

List Modification Examples

```python
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
```

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"]
my_index(gettysburg, "and") => 2
my_index(gettysburg,"years") => 4
```

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

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
```

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
```

More on List Slicing

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

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

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

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]
```

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

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]]
```

Loop Examples:

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

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

myL =[1,2,"three",(4,5),[6,7,8],True]
for i in range(0,len(myL)):
    print(myL[i])

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]

Functions

- You can create functions with the def keyword.

```
def nothing (n):
    pass

def sum (x,y):
    return x + y

sum (3, 4)

def get_item (t):
    return t[1]

get_item ( (1,'a') )


# In Python, types are checked at run-time
get_item ( (1,'a') )
get_item ( [2,'b'] )
get_item ( "3c" )
```

Anonymous (lambda) functions:

```
(lambda x : x+1)

(lambda x : x+1) (2) #calling the anonymous function  with input 2

(lambda item: item[1])
```

Sorting revisited

Sorting list of tuples

```
myL = [('a',3), ('b',2), ('f',1),('d',1), ('e',1), ('c',1)]

# sorting according to first values of the tuples
sorted(myL)
# sorting according to second values of the tuples
sorted(myL, key = lambda item: item[1])

# sort first on the second value then the first
sorted(sorted(myL), key = lambda item: item[1])
```