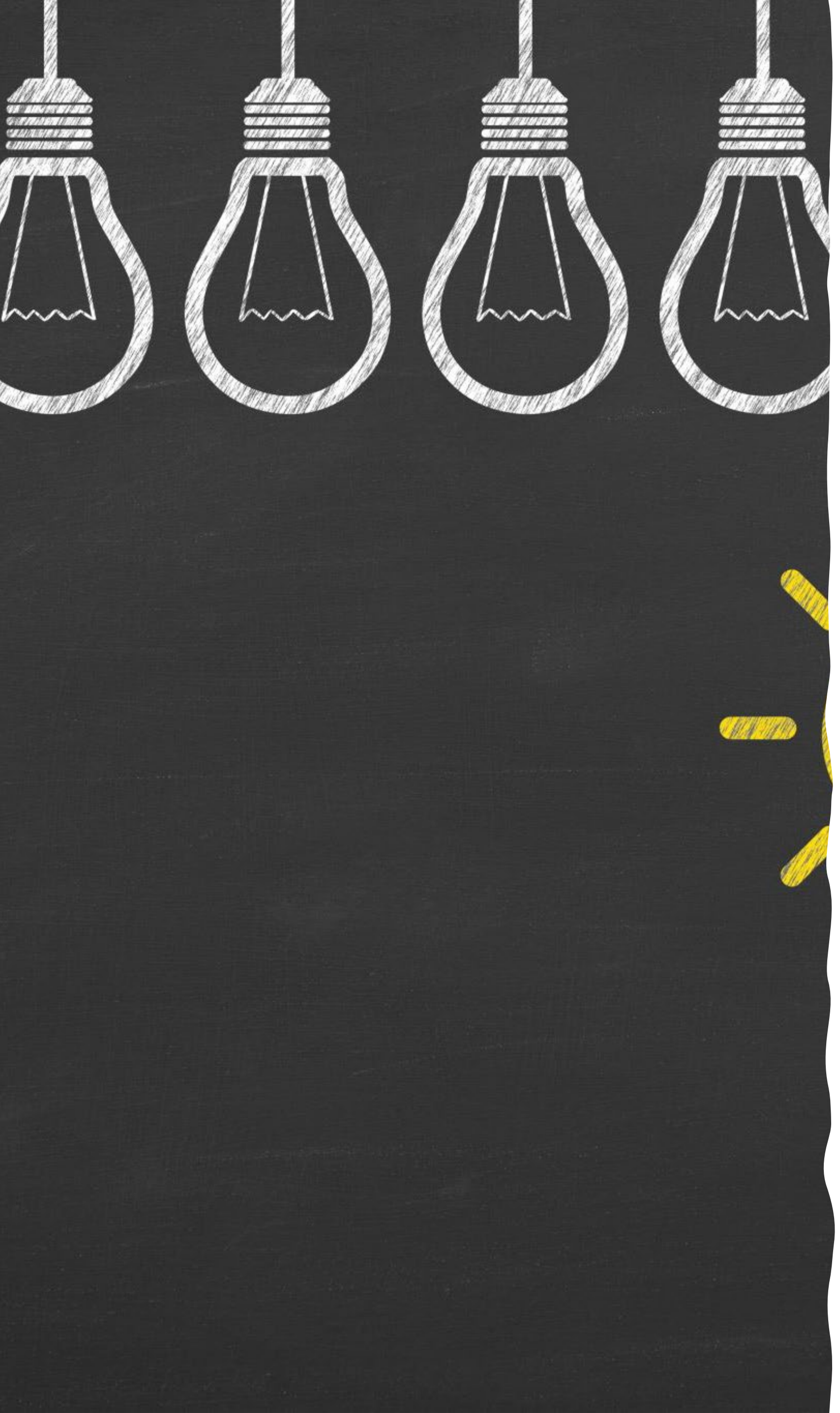


CMSC 105
Elementary Programming

Acknowledgement: These slides are adapted from slides provided with “Introduction to Programming Using Python, Liang (Pearson 2013)” and slides shared by Dr. Jory Denny

Outline

Lists

Practice Exercises

Last Time: Strings & String Operations

- Strings are an **ordered collection** of characters
- Strings can be **indexed**, and **sliced**

0	1	2	3	4	5	6
T	u	e	s	d	a	y

```
>>> word_1 = "Tuesday"

>>> word_1[0]

'T’

>>> word_1[5]

'a'
```

Today – Lists

They are ordered collections of well………., ANYTHING!!

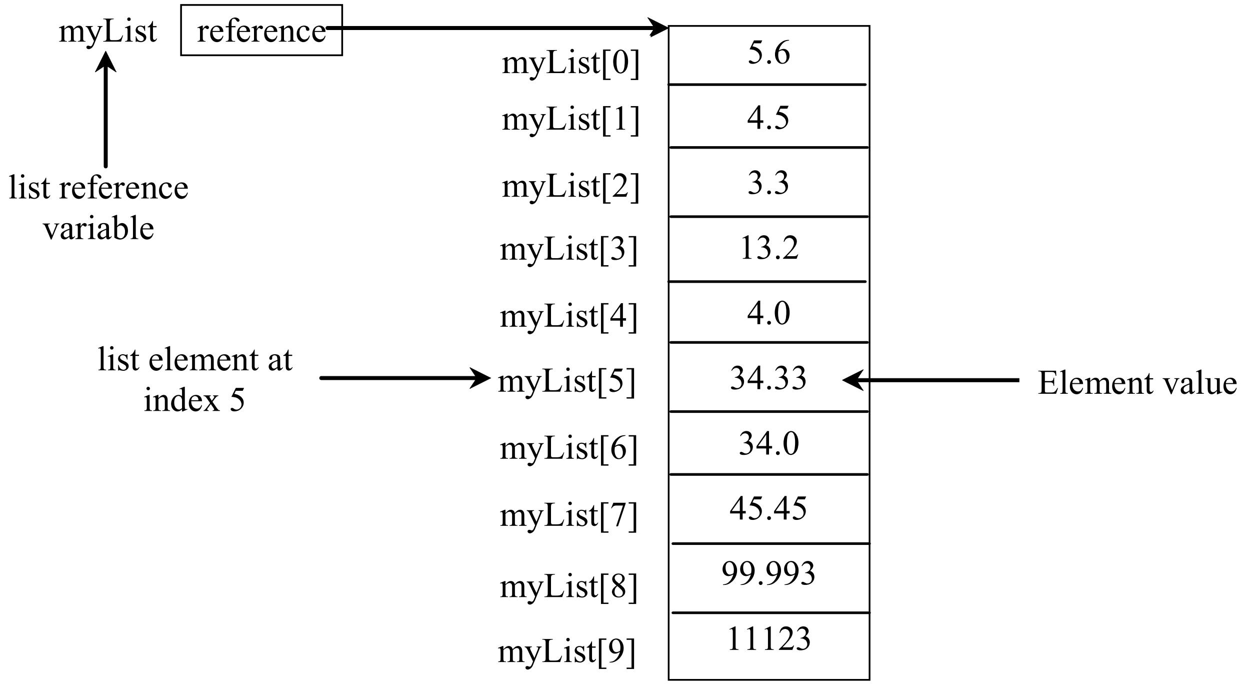
Motivation

- Read one hundred numbers, compute their average, and find out how many numbers are above the average.
- Store and manipulate large amounts of data
  - 52 playing cards in a deck
  - 3 thousand undergrads at UR
  - 140 characters per Tweet
  - 4 billion nucleotides in a DNA strand
  - 50 trillion cells in the human body
  - $6.022x10^{23}$ particles in a mole

Introducing lists

- **List** is a data structure that represents a collection of data of any size.
- List objects are **references**.

```
myList = [5.6, 4.5, 3.3, 13.2, 4.0, 34.33, 34.0, 45.45, 99.993, 11123]
```



Like strings, lists can be indexed and sliced

```
>>> list_sample=[1,2,3,4,5,6]
>>> list_sample[0]
1
>>> list_sample[4]
5
>>> list_sample[0:3]
[1, 2, 3]
```

0	1	2	3	4	5
1	2	3	4	5	6

Trace Problem with Lists

```
values = [0,0,0,0,0]
for i in range(1, 5):
    values[i] = i + values[i-1]
values[0] = values[1] + values[4]
```

Create a list with 5 zeroes in it named values

Memory

values   0xA

0xA

Index	Value
0	0
1	0
2	0
3	0
4	0

Trace Problem with Lists

```
values = [0,0,0,0,0]
for i in range(1, 5):
    values[i] = i + values[i-1]
values[0] = values[1] + values[4]
```

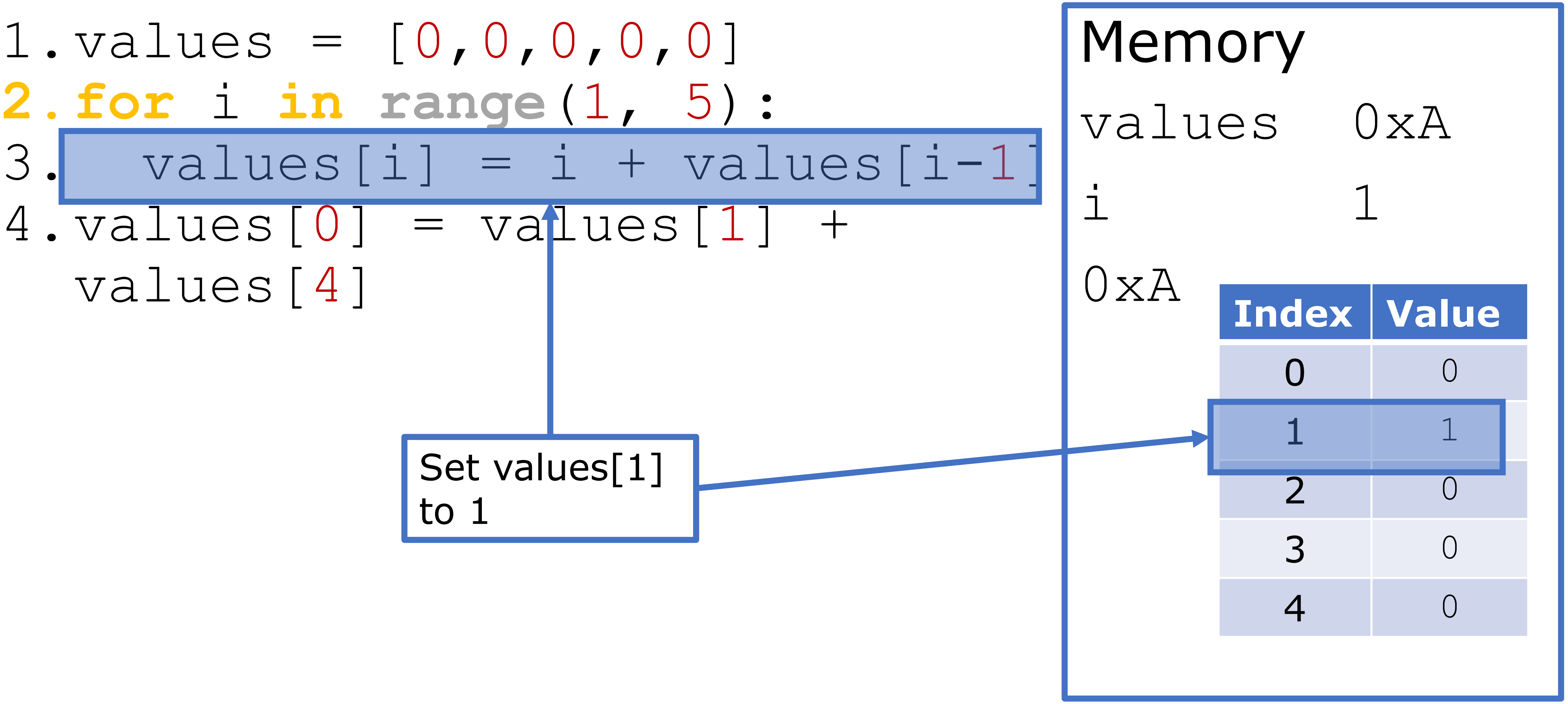
Memory

values	0xA
i	1

0xA

Index	Value
0	0
1	0
2	0
3	0
4	0

Trace Problem with Lists



```
values = [0,0,0,0,0]
for i in range(1, 5):
    values[i] = i + values[i-1]
values[0] = values[1] + values[4]
```

Set values[1] to 1

Memory

```
values   0xA
i        1
0xA
```

Index	Value
0	0
1	1
2	0
3	0
4	0

Trace Problem with Lists

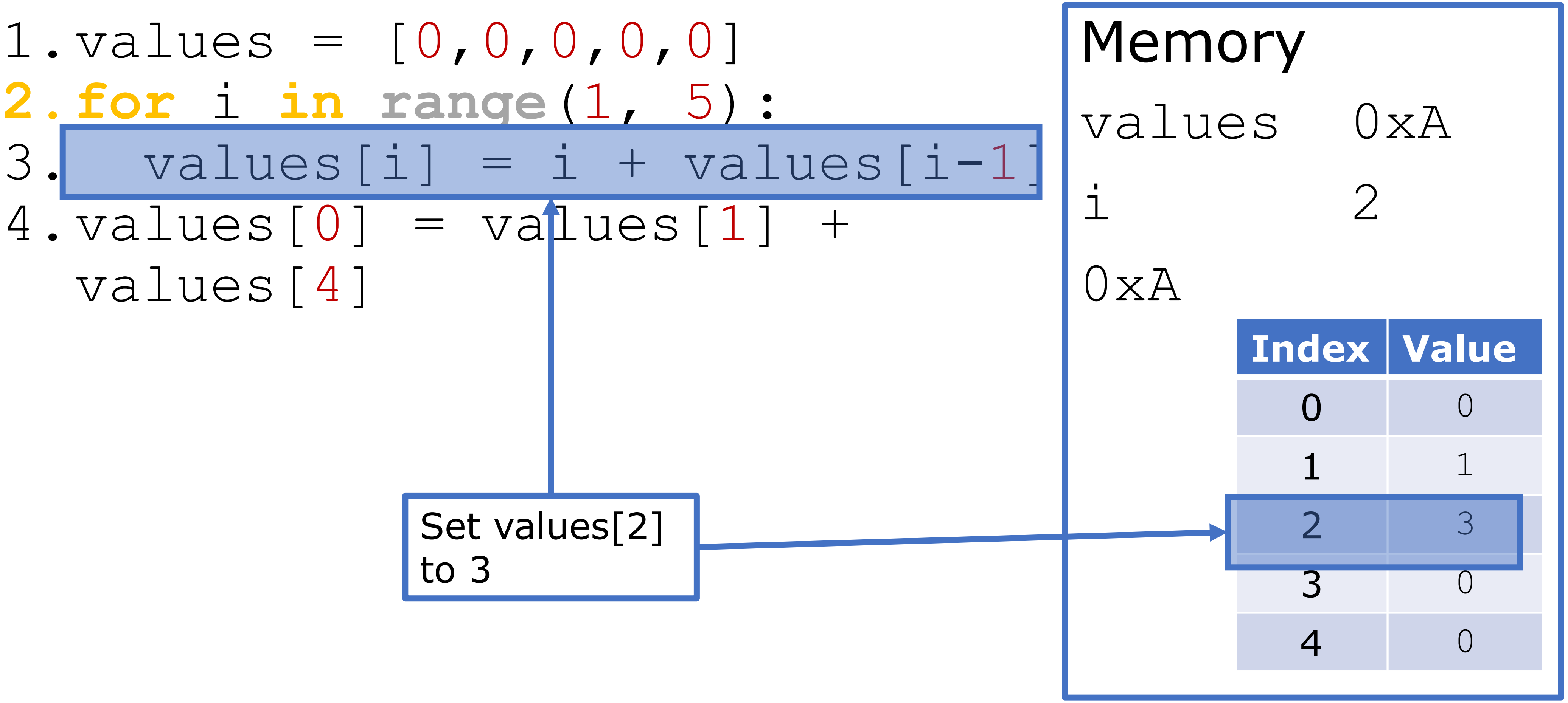
```
values = [0,0,0,0,0]
for i in range(1, 5):
  values[i] = i + values[i-1]
values[0] = values[1] +
values[4]
```

Memory

```
values    0xA
i         2
0xA
```

Index	Value
0	0
1	1
2	0
3	0
4	0

Trace Problem with Lists



```
values = [0,0,0,0,0]
for i in range(1, 5):
    values[i] = i + values[i-1]
values[0] = values[1] + values[4]
```

Memory

values	0xA
i	2

0xA

Index	Value
0	0
1	1
2	3
3	0
4	0

Set values[2] to 3

Trace Problem with Lists

```
values = [0,0,0,0,0]
for i in range(1, 5):
    values[i] = i + values[i-1]
values[0] = values[1] + values[4]
```

Memory

```
values    0xA
i         3
```

0xA

Index	Value
0	0
1	1
2	3
3	0
4	0

Trace Problem with Lists

```
values = [0,0,0,0,0]
for i in range(1, 5):
    values[i] = i + values[i-1]
values[0] = values[1] +
values[4]
```

Set values[3] to 6

Memory

values	0xA
i	3

0xA

Index	Value
0	0
1	1
2	3
3	6
4	0

Trace Problem with Lists

```
values = [0,0,0,0,0]
for i in range(1, 5):
    values[i] = i + values[i-1]
values[0] = values[1] + values[4]
```

Memory

values	0xA
i	4

0xA

Index	Value
0	0
1	1
2	3
3	6
4	0

Trace Problem with Lists

```
values = [0,0,0,0,0]
for i in range(1, 5):
    values[i] = i + values[i-1]
values[0] = values[1] +
values[4]
```

Memory

```
values    0xA
i         4
0xA
```

Index	Value
0	0
1	1
2	3
3	6
4	10

Set values[4] to 10

Trace Problem with Lists

```
values = [0,0,0,0,0]
for i in range(1, 5):
    values[i] = i + values[i-1]
values[0] = values[1] + values[4]
```

The end has been reached.

Memory

values	0xA
i	4

0xA

Index	Value
0	0
1	1
2	3
3	6
4	10

Trace Problem with Lists

```
values = [0,0,0,0,0]
for i in range(1, 5):
  values[i] = i + values[i-1]
values[0] = values[1] + values[4]
```

Set values[0] to 11

Memory

```
values   0xA
i        4
```

0xA

Index	Value
0	11
1	1
2	3
3	6
4	10

List Syntax and Operators

Creating Lists

- You can create lists using the list class constructor:

```
list1 = list()              # Create an empty list
list2 = list([2, 3, 4])     # Create a list with elements 2,
3, 4
list3 = list(["red", "green", "blue"]) # Create a list with
strings
list4 = list(range(3, 6)) # Create a list with elements 3,
4, 5
list5 = list("abcd")        # Create a list with characters a,
b, c,d
```

- For convenience, you may create a list using the following syntax:

```
list1 = []                 # Same as list()
list2 = [2, 3, 4]          # Same as list([2, 3, 4])
list3 = ["red", "green"] # Same as list(["red", "green"])
```

list Methods

list	
append(x: object): None	Add an item x to the end of the list.
insert(index: int, x: object): None	Insert an item x at a given index. Note that the first element in the list has index 0.
remove(x: object): None	Remove the first occurrence of the item x from the list.
index(x: object): int	Return the index of the item x in the list.
count(x: object): int	Return the number of times item x appears in the list.
sort(): None	Sort the items in the list.
reverse(): None	Reverse the items in the list.
extend(l: list): None	Append all the items in L to the list.
pop([i]): object	Remove the item at the given position and return it. The square bracket denotes that parameter is optional. If no index is specified, list.pop() removes and returns the last item in the list.

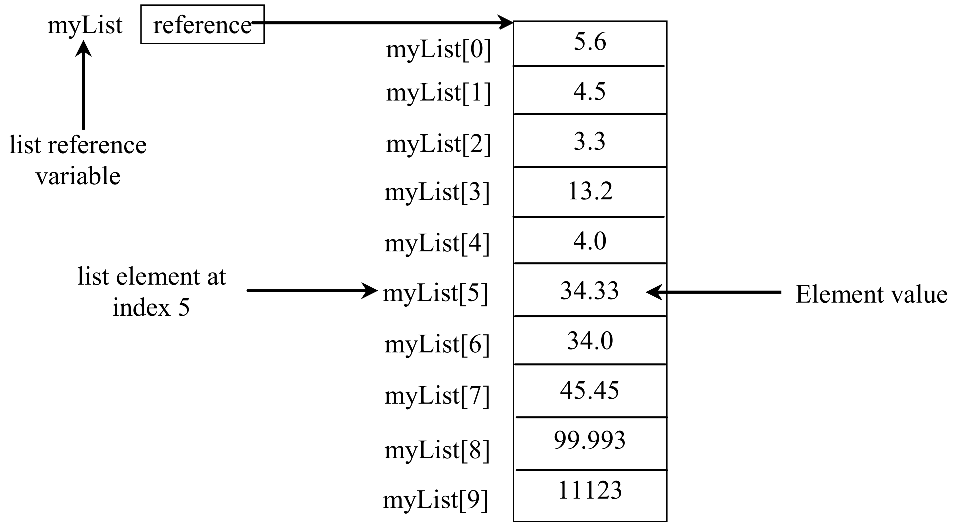
Built-in Functions for lists

- Let:
  ```
  l = [2, 3, 4, 1, 32]
  ```
- `len(l)` – computes the number of entries in the list (in this case 5)
- `max(l)` – computes the maximum element of the list (in this case 32)
- `min(l)` – computes the minimum element of the list (in this case 1)
- `sum(l)` – computes the summation of the elements in the list (in this case 42)
- Other libraries contain more functionality. Example of shuffling a list:
  ```
  import random
  random.shuffle(l) # Shuffle the items in the list
  print(l)          # Shows: [4, 1, 2, 32, 3]
  ```

Index Operator []

- The index operator `[]` selects the object at a specific location (**index**) in the data

```
myList = [5.6, 4.5, 3.3, 13.2, 4.0, 34.33, 34.0, 45.45, 99.993, 11123]
```



The +, *, [:], and in Operators

- Lists are similar to strings. Consider:

```
l1 = [2, 3]
l2 = [1, 9]
l3 = l1 + l2 # l3 contains [2, 3, 1, 9]
l4 = 3*l1    # l4 contains [2, 3, 2, 3, 2, 3]
l5 = l4[2:4] # l5 contains [2, 3]

contains4 = 4 in l5          # contains4 stores False
doesntcontain5 = 5 not in l5 # doesntcontains5 stores True
```

The +, *, [:], and in Operators

- `+` – is an operator that **concatenates** (joins/appends) two lists and returns the result
- `*` – is an operator that repeats a list some amount of times and returns the result (called the **repetition operator**)
- `[:]` – is an operator that returns a sublist of the list, called the **slicing operator**. The slice returned begins at the first index and ends at the second index -1.
- `in` and `not in` – are **containment operators** returning Boolean values whether an object/sublist is contained/not contained within a list.

Negative Index in a slicing operator

- Consider:

```
l1 = [2, 3, 5, 2, 33, 21]
print(l1[-1]) # 21
print(l1[-3]) # 2
```

- A negative index counts from the end of the list

Common pitfall

off-by-one errors

- Be careful of indexing and slicing operators, it is easy to get an index that is not valid.
- Consider:

```
lst = [0, 1, 2, 3]
i = 0
while i <= len(lst):
  print(lst[i])
  i += 1
```

- This code generates a tracing error:

```
IndexError: list index out of range
```

List Comprehension

- List comprehensions provide a very concise syntax for generating lists.
- A list comprehension consists of brackets containing an *expression* followed by a *for clause*, then zero or more *for* or *if clauses*.
- The result will be a list resulting from evaluating the expression.

- Compare the following:

```
l1 = list()
for x in range(0, 5):
   l1.append(x)
```

- To using a list comprehension:

```
l1 = [x for x in range(0, 5)]
# Generates [0, 1, 2, 3, 4]
```

- Other examples:

```
l2 = [0.5 * x for x in l1]
# Generates [0.0, 0.5, 1.0,
1.5, 2.0]

l3 = [x for x in l2 if x <
1.5]
# Generates [0.0, 0.5, 1.0]
```

List Details

Splitting a String to a List

- Often we need to split strings based on a delimiter (e.g., space). The string method split, generates a list.
- Example:

```
items = "Welcome to the US".split()
print(items) # ['Welcome', 'to', 'the', 'US']

items = "34#13#78#45".split("#")
print(items) # ['34', '13', '78', '45']
```

Split() Method

- Example:

```
>>> str_val="We are practicing split functions."
>>> lst=str_val.split(" ")
>>> lst
['We', 'are', 'practicing', 'split',
'functions.']
```

Split() method cont’d

- Example 2:

```
>>> num=input("Enter numbers separated by spaces")
Enter numbers separated by spaces1 2 3 4
>>> lst=num.split(" ")
>>> lst
['1', '2', '3', '4’]
```

Numbers are in string format. Can you convert them to integer format?

lst2=[int(i) for i in lst]

Iterating on Lists

```
>>> list_sample=[1,2,3,4,5,6]
>>> for i in list_sample:
         print(i)
```

```
Output:
1
2
3
4
5
6
```

Iterating on Lists

Using enumerate() function:

```
>>> list_sample=[1,2,3,4,5,6]

>>> for index_val,value in enumerate(list_sample):
      print("Index",index_val,",value",value)
```

Output:

```
Index 0 ,value 1
Index 1 ,value 2
Index 2 ,value 3
Index 3 ,value 4
Index 4 ,value 5
Index 5 ,value 6
```

enumerate(iterable, start=0)

Parameters:

Iterable: any object that supports iteration

Start: the index value from which the counter is to be started, by default it is 0

Practice Exercises

Problem 1

- Write a program that takes as input 4 numbers separated by spaces and prints the maximum and minimum numbers in the sequence.

Sample run:

```
Enter elements of list separated by spaces10 1 200 2
Maximum number is 200
Minimum number is 1
```

Problem 2

- Write a program that takes as input 4 numbers separated by commas (say list1). Create a new list (say list2) such that it contains all elements of list1 multiplied by 4.

Sample run:

```
Enter elements of list separated by commas1,2,1,2
New list [4, 8, 4, 8]
```

Problem 3

- Write a program that reads some positive integers and counts the total number of even numbers.

Sample run:

```
Enter integers separated by spaces 2 3 4 5 4 5 2
```

Output:

```
There are 4 even numbers in the sequence
```

Chapters Covered from Textbook

- Chapter 10

Thank you!
Questions?