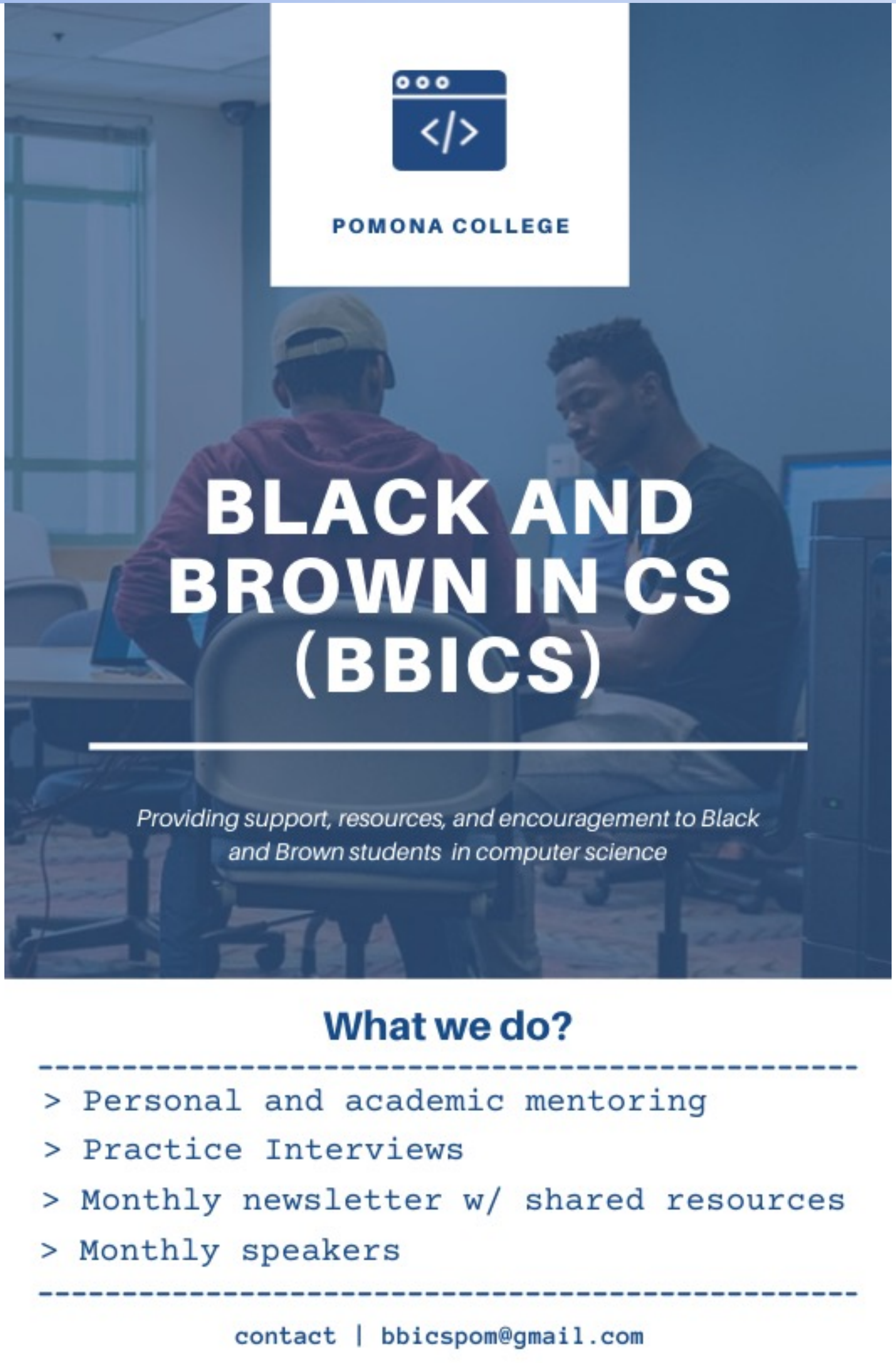
Lecture 9: Lists

CS 51P

October 3, 2022

Tom Yeh
he/him/his



POMONA COLLEGE

BLACK AND BROWN IN CS (BBICS)

Providing support, resources, and encouragement to Black and Brown students in computer science

What we do?

```
------------------------------------------------
> Personal and academic mentoring
> Practice Interviews
> Monthly newsletter w/ shared resources
> Monthly speakers
------------------------------------------------
```

contact | bbicspom@gmail.com

Learning Goals

- Learn about lists in Python
- Write code using lists

Programs operate on values

- compute new values using expressions
- store values in variables
- pass values to functions (as arguments)
- pass values to caller (as return value)

Can we operate on multiple values at the same time?

- Can we define a variable that stores the colors of the rainbow?
- Can we define a function that returns the squares of all the numbers in a specified range?
- Can we define a function that returns all the words in a string that begin with uppercase letters?

Data Structures

- a **data structure** is a type that stores a collection of values
- Python provides some built-in data structure types

Sequences

- sequences are ordered sets of values
  - ranges are sequences of integers
  - strings are sequences of characters
  - files are sequences of strings

- we can perform operations on sequences
  - indexing (e.g., "hello"[0])
  - slicing (e.g., "hello"[1:5])
  - looping (with for loop) (e.g., for i in range(1,10):)
  - check membership (e.g., char in "abcd")

Can we have a sequence of arbitrary values?

What is a List?

- a list is a way to keep track of an *ordered collection* of items
  - Items in the list are called elements
  - **Ordered**: can refer to elements by their position (start with 0)
  - **Collection**: list can contain multiple items

```
a_list = [3, 6, 2, 1]
```

- a list dynamically adjusts its size as elements are added or removed
- a list is a sequence, so can index into, loop over, check for membership, slice
  - Lots of built-in functionality

Show me a List!

- Creating lists
  - Lists start/end with brackets with elements separated by commas.
  - Call a function that returns a list

```
a_list = [3, 6, 2, 1]
float_list = [5.1, 6.2, 0.23]
str_list = [‘this’, ‘is’, ‘a’, ‘list’]
mix_list = [3, 5.1, ‘is’, True]
empty_list = []

c_list = "a b c d".split()
```

- List with 1 element is **not** the same as the element, how do you compare?
  - >>> list_one = [51]
  - >>> one = 51
  - >>> list_one == one
  - False

Accessing Elements of a List

- Consider this list:

```
a_list = [‘a’, ‘b’, ‘c’, ‘d’, ‘e’]
```

- Can think of it like a series of variables that are indexed
  - Index starts from 0

- a_list

‘a’	**‘b’**	**‘c’**	**‘d’**	**‘e’**
0	1	2	3	4

- Accessing individual elements:
  - a_list[0] is ‘a’
  - a_list[3] is ‘d’

Accessing Elements of a List

- Consider this list:

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

- Can think of it like a series of variables that are indexed
  - Index starts from 0

- a_list

'a'	**'x'**	**'c'**	**'d'**	**'e'**
0	1	2	3	4

- Accessing individual elements:
  - a_list[0] is 'a'
  - a_list[3] is 'd'
- Can modify individual elements like variables
  - a_list[1] = 'x'

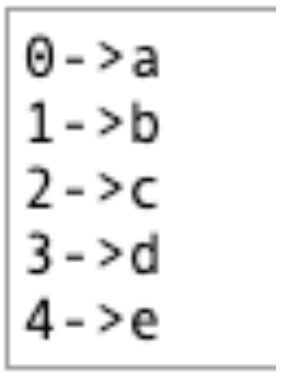
Length of a List

- Consider this list:

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

- Can get length of a list with len function:
  - len(a_list) is 5
  - Elements indexed from 0 to length - 1

- Code example:
  - for I in range(len(a_list)):
    - print(str(i) + "->" + a_list[i])



```
0->a
1->b
2->c
3->d
4->e
```

Negative indexing – like string slicing

- Consider this list:

```
a_list = [‘a’, ‘b’, ‘c’, ‘d’, ‘e’]
```

- Can do this:
  - a_list[-1] is ‘e’
  - a_list[-2] is ‘d’
- For negative index, think of –x as len(list) – x
  - a_list[-1] is the same as a_list[4]
- What about a_list[6]?

Lists as sequences

```
string = "Hello world !! "
print(string[1:3])
print(string[-1])
print(string[:2])

str_list = string.split()
print(str_list)
print(str_list[1:3])
print(str_list[-1])
print(str_list[:2])
```

Differences about Lists

- the elements of a list can have any value and any type

```
a_list = [3.5, 6, [1, 2], "abc"]
```

- lists are mutable (more on this)
  - add elements

```
a_list.append("c")
a_list.extend(["c","b"])
```

  - modify elements

```
a_list[3] = 3.33333
a_list[:2] = ["a", "b"]
```

  - remove elements

```
a_list.pop() # returns element
del(a_list[0:1])
```

List Operations

adding to a list
(updates original list)

- a_list.extend(*list*)
- a_list.append(*elem*)
  - Different than extend – e.g. [5, 1]
- a_list.insert(*index*, *elem*)

other

- min(a_list), max(a_list), len(a_list)
- *elem* in a_list
  - returns bool
- a_list.index(*elem*)
  - returns index of 1st instance of elem or error
- a_list.insert(*index, elem*)
  - Insert elem at index, shifts down
- a_list.copy()
  - Returns a copy of list
- if a_list:
  - checks is list is empty

List Operations

removing from a list

- del(a_list[*slice*])
- a_list.remove(*elem*)
  - removes 1st instance of elem
  - error if *elem* not in a_list
- a_list.pop()
  - returns (and removes) a_list[-1]
- a_list.pop(*index*)
  - returns (and removes) a_list[index]

modifying a list

- direct assignment
  - a_list[0] = 2

printing a list

```
>>> print(a_list)
[1, 2, 3, 4, 5]
```

+ and * operators

- Works on lists, but creates a new list
  - >>> a_list = [1, 2, 3]
  - >>> new_list = a_list + a_list
  - >>> new_list
  - [1,2,3,1,2,3]

Exercise

```
a_list = [3.5, 6, [1, 2], "abc"]
a_list[3] = list(range(0,5,2))
a_list[:2] = ["a", "b"]
a_list.extend([5,3,1])

print(len(a_list))
for elem in a_list:
     print(str(elem) + ":" + str(type(elem)))

del(a_list[3:5])
a_list.remove("a")
print(a_list)
```

Example

- Can we define a function that returns the squares of all the numbers in a specified range?

Exercise

- Define a function digits that takes one parameter num (an positive int) and returns a list of the digits of num

Example

- Define a function word_list that takes a filename as an argument and returns a list of all the words in that file.

Exercise

- Define a function count_words that takes a filename as input and returns the total number of unique words in that file