Introduction to Python

Lists

Topics

1) Lists
2) List indexing
3) Traversing and modifying a list
4) Summing a list
5) Maximum/Minimum of a list
6) List Methods

Containers

Python includes several built-in sequences: *lists, tuples, strings*. We will discuss these in the next few lectures. Here's a broad overview:

Lists and *tuples* are *container sequences*, which can hold items of different type.

String is a flat sequence which holds item of one type(characters).

Another way of grouping sequence types is by *mutability*. Lists are *mutable(can be modified)* sequences while strings and tuples are *immutable* sequences. We discuss lists in this lecture.

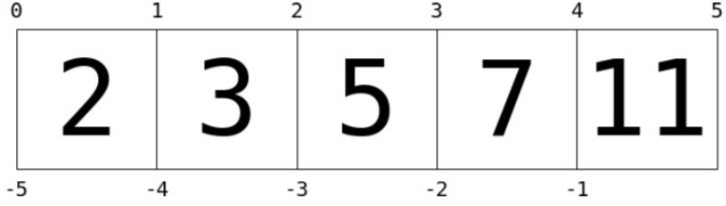
Lists

Lists are the basic *ordered* and *mutable* data collection type in Python. They can be defined with comma-separated values between square brackets.

```
L = [2, 3, 5, 7]
print(len(L))          # 4, len() also works with strings
L.append(11)           # append to the end of the list
print(L)               # [2, 3, 5, 7, 11]
```

Indexing

Indexing is a means the fetching of a single value from the list. This is a 0-based indexing scheme.




```
L = [2, 3, 5, 7, 11]
print(L[0])           # 2
print(L[1])           # 3
print(L[5])           # index out of bounds error.
```

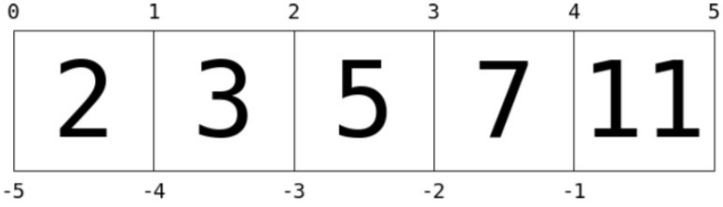
Lists can contain different types of objects

List can contain different types and even other lists.

```
L = [1, 'two', 3.14, [-2, 3, 5]]
print(L[0])          # 1
print(L[1])          # two
print(L[3])          # [-2, 3, 5]
print(L[3][0]) # -2
print(L[3][1]) # 3
print(L[3][2]) # 5
```

Modifying a List

Indexing can be used to set elements as well as access them.




```
L = [2, 3, 5, 7, 11]
L[0] = 100
print(L)        # [100, 3, 5, 7, 11]
L[2] = -4
print(L)        # [100, 3, -4, 7, 11]
```

Slicing

We saw in the previous lecture that we can slice a string by specifying a start-index and stop-index, and the result is a subsequence of the items contained within the slice.

Slicing also works with lists!(As well as many important data structures in Python) Slicing can be done using the syntax:

```
my_list[start:stop:step]
```

where

start: index of beginning of the slice(included), default is 0

stop: index of the end of the slice(excluded), default is length of the list

step: increment size at each step, default is 1.

List Slicing

```
L = [10, -2, 1, 6, 2]
print(L[1:3])          # [-2, 1]
print(L[:3])           # [10, -2, 1]
print(L[1:])           # [-2, 1, 6, 2]
print(L[0:4:2])        # [10, 1]
print(L[:])            # [10, -2, 1, 6, 2]
print(L[::-1])         # [2, 6, 1, -2, 10]
```

Traversing a list

We can traverse through a list using a for loop. We have seen this before with strings! There are two options:

1) for each loop:

```
nums = [2, -1, 3, 4, -3]
for x in nums:
      print(x, end=" ")
2 -1 3 4 -3
```

Looping through each value

2) loop using indices

```
nums = [2, -1, 3, 4, -3]
for i in range(len(nums)):
      print(nums[i], end=" ")
2 -1 3 4 -3
```

Looping through each index
i takes on values: 0,1,2,3,4.

Modifying a list

Consider the following code that is intended to change all even numbers in a list to 0.

```
nums = [24, 3, 34, 6, -5, 4]
for x in nums:
      if x % 2 == 0:
            x = 0
print(nums)
```

```
Output:
[24, 3, 34, 6, -5, 4]
```

Note: The list is unchanged? Why? How can we fix it?

Modifying a list

Here's the correct code to change all even numbers in a list to 0. Compare the following code to the previous slide.

```
nums = [24, 3, 34, 6, -5, 4]
for i in range(len(nums)):
      if nums[i] % 2 == 0:
            nums[i] = 0
print(nums)
```

```
Output:
[0, 3, 0, 0, -5, 0]
```

Creating a list

If you want to create a list containing the first five perfect squares, then you can complete these steps in three lines of code:

```
squares = []              # create empty list
for i in range(5):
      squares.append(i ** 2) # add each square to list
print(squares)
```

```
Output:
[0, 1, 4, 9, 16]
```

There is a much simpler way to create this list using list comprehensions.

Creating a list with list comprehensions

List comprehensions is a way to create a list in Python that is concise and elegant. Its main use is to create a new list from a given list.

Instead of:

```
squares = []              # create empty list
for i in range(5):
      squares.append(i ** 2) # add each square to list
```

Do this:

```
squares = [i ** 2 for i in range(5)]          # one line!
```

List comprehensions allow you to use a conditional.

```
even_squares = [i ** 2 for i in range(5) if i % 2 == 0]
print(even_squares)   # [0, 4, 16]
```

Algorithms to know

The following algorithms are useful. Know how to implement these algorithms!

1) Find sum of a list of numbers.
2) Find the average of a list of numbers.
3) Find the maximum/minimum of a list of numbers.

Sum of a list

Given a list, find the sum of its elements. We can do this by traversing through the list using a for loop.

```
nums = [2, -1, 3, 4, -3]
s = 0
for x in nums:
    s += x
print(s)
```

Whenever we have a piece of code that accomplish a useful task, we should put it in a function.

Sum Function

Write a function that accepts a list of numbers as a parameter and returns its sum.

```
def sum(nums):
      s = 0
      for x in nums:
            s += x
      return s
lst = [2, -1, 3, 4, -3]
print(sum(lst))  # 5
lst2 = [1, 5, 4, 2]
a = sum(lst2)
print(a)         # 12
```

Average Function

Write a function that accepts a list of numbers as a parameter and returns its average.

```
def average(nums):
    s = 0
    for x in nums:
        s += x
    return s/len(nums)

lst = [2, 5, 4, 3]
a = average(lst)
print(a)        # 3.5
```

Conditional Summing

Write a function that accepts a list of numbers as a parameter and returns the sum of all even numbers in the list.

```
def sum(nums):
    s = 0
    for x in nums:
        if x % 2 == 0:
            s += x
    return s
```

Find Maximum Function

Write a function that accepts a nonempty list of numbers as a parameter and returns its maximum value. Does the code below work?

```
def maximum(nums):
      current_max = 0
      for x in nums:
            if x > current_max:
                  current_max = x
      return current_max


lst = [-2, -5, -12, -3]
a = maximum(lst)
print(a)          # 0 INCORRECT!
```

No! What if the list contains only negative numbers? This function returns 0 which is not even in the list!

Find Maximum Function

Here's the correct implementation of maximum. The minimum function is similar.

```
def maximum(nums):
    current_max = nums[0]     # the first value is maximum
    for x in nums:            # until a bigger value shows up
        if x > current_max:
            current_max = x
    return current_max

lst = [2, 5, 12, 3, 4, 11]
a = maximum(lst)
print(a)        # 12
```

List Methods

The following is a short list of useful list methods.

append(value)	appends value to the end of the list
insert(index, value)	inserts value at position given by index, shifts elements to the right.
pop(index)	removes object at index from list, shifts elements left and returns removed object. Returns last element if index is omitted. The index parameter is optional(default to last element).
split()	splits a string into a list. A separator can be specified. The default separator is any whitespace.

List Methods

```
L = [3, "hi", -4, 6]
L.append(2)                 # L = [3,"hi",-4, 6, 2]
L.insert(1, "hello")        # L = [3,"hello","hi",-4, 6, 2]
a = L.pop(3)                # L = [3,"hello","hi",6, 2]
print(a)                    # -4
L.pop()                     # ok to not store popped value
print(L)                    # [3,"hello","hi",6]
```

split()

The split() method splits a string into a list. A separator can be specified. The default separator is any whitespace.

```
fruits = "apple mango banana grape"
fruits_lst = fruits.split()
print(list_fruits)   # [‘apple’, ‘mango’, ‘banana’, ‘grape’]

greeting = “hi,I am Mike,I just graduate.”
greet_lst = greeting.split(",")
print(greet_lst)    # [‘hi’, ‘I am Mike’, ‘I just graduate.’]

nums = “4 24 12”
nums_lst = nums.split()
print(nums_lst)    # [‘4’, ‘24’, ‘12’], these are still strings
```

Create a list from user inputs

Ask the user to enter a list of numbers **of any length** separated by spaces. Generate a list containing those numbers. The split() function can be used here.

```
nums = input("Enter list of numbers separated by spaces: ").split()
print(nums)
```

Sample output:

Enter list of numbers separated by spaces: 4 6 1 23

['4', '6', '1', '23']

Note that the list above is a list of strings! We like this to be a list of integers.

Create a list from user inputs

From the previous example, we can change each string in the list by manually casting it to an integer.

```
nums = input().split()
for i in range(len(nums)):
      nums[i] = int(num[i])       # cast string to integer
print(nums)
```

Sample output:

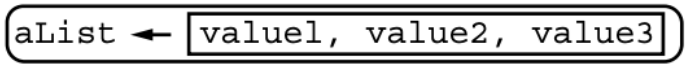
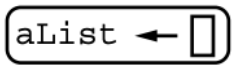
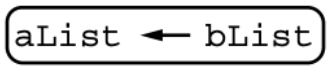
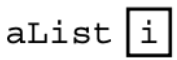
4 6 1 23

[4, 6, 1, 23]

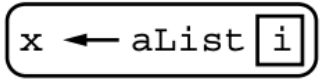
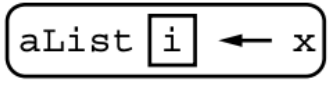
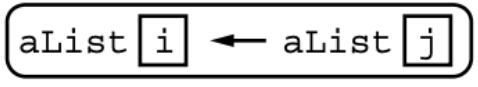
We can further simplify the code above using list comprehensions! This code below can used in many of the replit Teams problems for this lecture.

```
nums = [int(x) for x in input().split()]
```

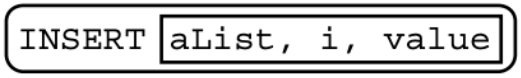
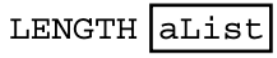
AP Exam: Lists API

Text: `aList ← [value1, value2, value3, ...]` Block: 	Creates a new list that contains the values `value1`, `value2`, `value3`, and `...` at indices `1`, `2`, `3`, and `...` respectively and assigns it to `aList`.
Text: `aList ← []` Block: 	Creates an empty list and assigns it to `aList`.
Text: `aList ← bList` Block: 	Assigns a copy of the list `bList` to the list `aList`. For example, if `bList` contains `[20, 40, 60]`, then `aList` will also contain `[20, 40, 60]` after the assignment.
Text: `aList[i]` Block: 	Accesses the element of `aList` at index `i`. The first element of `aList` is at index `1` and is accessed using the notation `aList[1]`.

AP Exam: Lists API

Text: `x ← aList[i]` Block: 	Assigns the value of `aList[i]` to the variable `x`.
Text: `aList[i] ← x` Block: 	Assigns the value of `x` to `aList[i]`.
Text: `aList[i] ← aList[j]` Block: 	Assigns the value of `aList[j]` to `aList[i]`.

AP Exam: Lists API

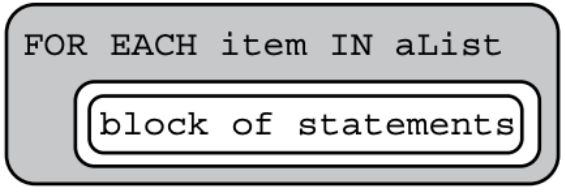
Text: `INSERT(aList, i, value)` Block: 	Any values in `aList` at indices greater than or equal to `i` are shifted one position to the right. The length of the list is increased by 1, and `value` is placed at index `i` in `aList`.
Text: `APPEND(aList, value)` Block: 	The length of `aList` is increased by 1, and `value` is placed at the end of `aList`.
Text: `REMOVE(aList, i)` Block: 	Removes the item at index `i` in `aList` and shifts to the left any values at indices greater than `i`. The length of `aList` is decreased by 1.
Text: `LENGTH(aList)` Block: 	Evaluates to the number of elements in `aList`.

AP Exam: Lists API

Text:

```
FOR EACH item IN aList
{
 <block of statements>
}
```

Block:



The variable `item` is assigned the value of each element of `aList` sequentially, in order, from the first element to the last element. The code in `block of statements` is executed once for each assignment of `item`.

Lab 1

Create a new repl on repl.it

1) Create this list and assign it to a variable [3,41,62,87,101, 88]. Use a for loop to compute the sum. Print out the sum.
2) Use a for loop to compute the sum of odd numbers from the list above.
3) Use a for loop to compute the sum of values located at even indices.(Use the len() function).

Lab 2

Create a new repl on repl.it Use list comprehensions to create the following lists.

1) `[2,4,6,8,10,…,20]`
2) `[1,8,27,64,125]`
3) `[0.5,1.0,1.5,2.0,2.5,3.0]`
4) `['1','2','3','4','5']`
5) `[1,3,5,7,9,…,99]`. Must use condition in the list comprehension.

References

1) Vanderplas, Jake, A Whirlwind Tour of Python, O’reilly Media.
2) Luciano, Ramalho, Fluent Python, O'reilly Media.