CS303E: Elements of Computers and Programming

Lists

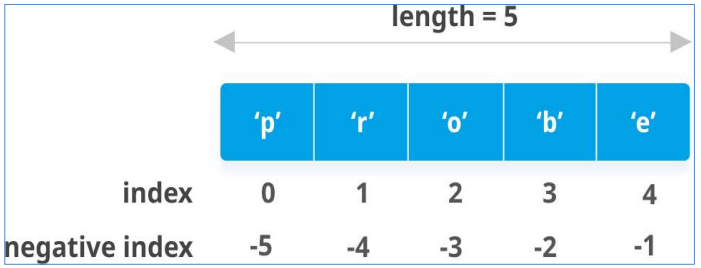
Mike Scott
Department of Computer Science
University of Texas at Austin

Adapted from
Professor Bill Young's Slides

Last updated: June 28, 2023

Lists

The `list` class is a very useful tool in Python.



Both lists and strings are sequence types in Python, so share many similar methods. Unlike strings, lists are *mutable*.

If you change a list, it doesn't create a new copy; *it changes the actual contents of the list.*

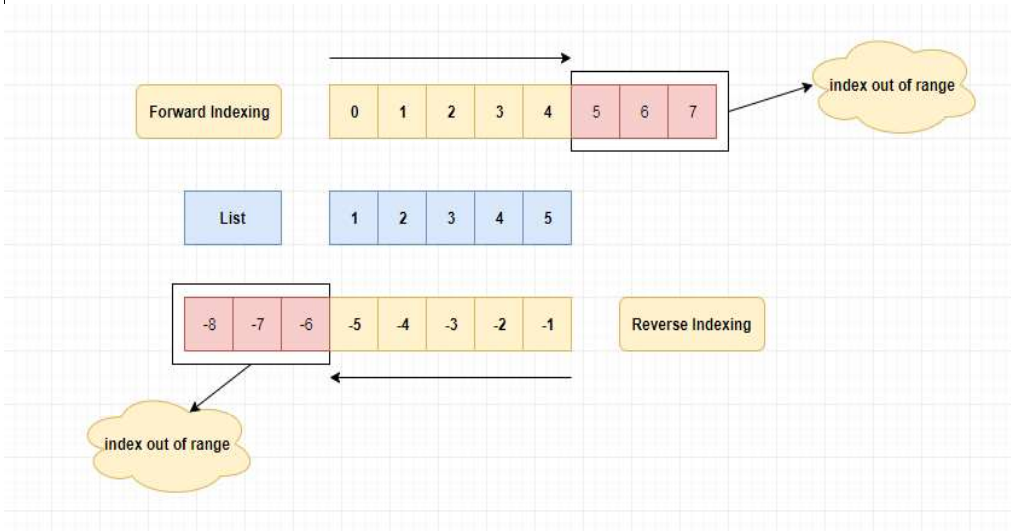
Value of Lists

Suppose you have 30 different test grades to average. You could use 30 variables: grade1, grade2, ..., grade30. Or you could use one list with 30 elements: grades[0], grades[1], ..., grades[29].

```
def grades_example():
    """Shows creation of a list and determining
    average of elements."""
    grades = [67, 82, 56, 84, 66, 77, 64, 64, 85, 67,
              73, 63, 98, 74, 81, 67, 93, 77, 97, 65,
              77, 91, 91, 74, 93, 56, 96, 90, 91, 99]
    total = 0
    for score in grades:
        total += score
    average = total / len(grades)
    print("Class average =", format(average, '.2f'))
```


Indexing and Slicing

With Lists you can get sublists using **slicing**



List Slicing

- List slicing format: *list[start : end]*
- Span is a list containing copies of elements from *start* up to, but not including, *end*
  - If *start* not specified, 0 is used for start index
  - If *end* not specified, len(list) is used for end index
- Slicing expressions can include a step value and negative indexes relative to end of list

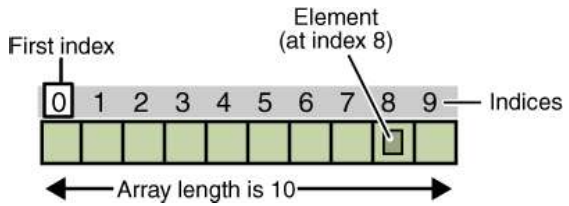
Creating Lists

Lists can be created with the list class constructor or using special syntax.

```
>>> list()                  # create empty list, with constructor
[]
>>> list([1, 2, 3])        # create list [1, 2, 3]
[1, 2, 3]
>>> list(["red", 3, 2.5])   # create heterogeneous list
['red', 3, 2.5]
>>> ["red", 3, 2.5]         # create list, no explicit constructor
['red', 3, 2.5]
>>> range(4)                # not an actual list
range(0, 4)
>>> list(range(4))          # create list using range
[0, 1, 2, 3]
>>> list("abcd")            # create character list from string
['a', 'b', 'c', 'd']
```


Lists vs. Arrays

Many programming languages have an **array** type.



Arrays are:

- homogeneous (all elements are of the same type)
- fixed size
- permit very fast access time

Python lists are:

- heterogeneous (can contain elements of different types)
- variable size
- permit fast access time

Lists and arrays are examples of ***data structures***. A **very** simple definition of a data structure is **a variable that stores other variables.**
CS313e explores many standard data structures.

Sequence Operations

Lists are sequences and inherit various functions from sequences.

Function	Description
x in s	x is in sequence s
x not in s	x is not in sequence s
s1 + s2	concatenates two sequences
s * n	repeat sequence s n times
s[i]	ith element of sequence (0-based)
s[i:j]	slice of sequence s from i to j-1
len(s)	number of elements in s
min(s)	minimum element of s
max(s)	maximum element of s
sum(s)	sum of elements in s
for loop	traverse elements of sequence
<, <=, >, >=	compares two sequences
==, !=	compares two sequences

Calling Functions on Lists

```
>>> l1 = [1, 2, 3, 4, 5]
>>> len(l1)
5
>>> min(l1)       # assumes elements are comparable
1
>>> max(l1)       # assumes elements are comparable
5
>>> sum(l1)       # assumes summing makes sense
15
>>> l2 = [1, 2, "red"]
>>> sum(l2)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str
    '
>>> min(l2)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: '<' not supported between instances of 'str' and
    'int'
>>>
```


Using Functions

We could rewrite the grades_examples function as follows:

```
def grades_example_2():
    """Shows creation of a list and determining
    average of elements. This version takes advantage
    of the sum function for sequences."""
    grades = [67, 82, 56, 84, 66, 77, 64, 64, 85, 67,
              73, 63, 98, 74, 81, 67, 93, 77, 97, 65,
              77, 91, 91, 74, 93, 56, 96, 90, 91, 99]
    average = sum(grades) / len(grades)
    print("Class average =", format(average, '.2f'))
```


Traversing Elements with a For Loop

General Form:

```
for u in list:
    body
```

In file test.py:

```
for u in range(3):              # not really a list
    print(u, end=" ")
print()

for u in [2, 3, 5, 7]:
    print(u, end=" ")
print()

for u in range(15, 1, -3):   # not really a list
    print(u, end=" ")
print()
```

```
> python test.py
0 1 2
2 3 5 7
15 12 9 6 3
```


Comparing Lists

Compare lists using the operators: >, >=, <, <=, ==, !=. Uses *lexicographic* ordering: Compare the first elements of the two lists; if they match, compare the second elements, and so on. The elements must be of *comparable* classes.

```
>>> list1 = ["red", 3, "green"]
>>> list2 = ["red", 3, "grey"]
>>> list1 < list2
True
>>> list3 = ["red", 5, "green"]
>>> list3 > list1
True
>>> list4 = [5, "red", "green"]
>>> list3 < list4
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: '<' not supported between instances of 'str' and
    'int'
>>> ["red", 5, "green"] == [5, "red", "green"]
False
```

List Comprehension

List comprehension gives a compact syntax for building lists.

```
>>> range(4)                    # not actually a list
range(0, 4)
>>> [ x for x in range(4) ]     # create list from range
[0, 1, 2, 3]
>>> [ x ** 2 for x in range(4) ]
[0, 1, 4, 9]
>>> lst = [ 2, 3, 5, 7, 11, 13 ]
>>> [ x ** 3 for x in lst ]
[8, 27, 125, 343, 1331, 2197]
>>> [ x for x in lst if x > 2 ]
[3, 5, 7, 11, 13]
>>> [s[0] for s in ["red", "green", "blue"] if s <= "green"]
['g', 'b']
>>> from IsPrime3 import *
>>> [ x for x in range(100) if isPrime(x) ]
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53,
 59, 61, 67, 71, 73, 79, 83, 89, 97]
```


List Comprehension with Files

List comprehension gives a compact syntax for building lists, even from files.

mples.py | sample.txt

```
Team by team, reporters baffled, trumped, tethered, cropped
Look at that low plane, fine, then
Uh oh, overflow, population, common group
But it'll do, save yourself, serve yourself
World serves its own needs, listen to your heart bleed
Tell me with the Rapture and the reverent in the right, right
You vitriolic, patriotic, slam fight, bright light
Feeling pretty psyched
```


List Comprehension with Files

List comprehension gives a compact syntax for building lists, even from files.

```
def list_from_file(file_path):
    """Read the lines from the given file and print them out."""
    with open(file_path, 'r') as infile:
        lines = [line.strip() for line in infile]
    print('number of lines:', len(lines))
    line_num = 1
    for line in lines:
        print(line_num, ': ', line, sep='')
        line_num += 1
```


List Comprehension with Files

List comprehension gives a compact syntax for building lists, even from files.

```
1: Team by team, reporters baffled, trumped, tethered, cropped
2: Look at that low plane, fine, then
3: Uh oh, overflow, population, common group
4: But it'll do, save yourself, serve yourself
5: World serves its own needs, listen to your heart bleed
6: Tell me with the Rapture and the reverent in the right, right
7: You vitriolic, patriotic, slam fight, bright light
8: Feeling pretty psyched
```

Let's Take a Break

More List Methods

These are methods from class `list`.
Since lists are mutable, these actually change `t`.

Method	Description
t.append(x)	add x to the end of t
t.count(x)	number of times x appears in t
t.extend(l1)	append elements of l1 to t
t.index(x)	index of first occurence of x in t
t.insert(i, x)	insert x into t at position i
t.pop()	remove and return the last element of t
t.pop(i)	remove and return the ith element of t
t.remove(x)	remove the first occurence of x from t
t.reverse()	reverse the elements of t
t.sort()	order the elements of t

List Examples

```
>>> l1 = [1, 2, 3]
>>> l1.append(4) # add 4 to the end of l1
>>> l1           # note: changes l1
[1, 2, 3, 4]
>>> l1.count(4)  # count occurrences of 4 in l1
1
>>> l2 = [5, 6, 7]
>>> l1.extend(l2)   # add elements of l2 to l1
>>> l1
[1, 2, 3, 4, 5, 6, 7]
>>> l1.index(5)   # where does 5 occur in l1?
4
>>> l1.insert(0, 0) # add 0 at the start of l1
>>> l1           # note new value of l1
[0, 1, 2, 3, 4, 5, 6, 7]
>>> l1.insert(3, 'a')        # lists are heterogenous
>>> l1
[0, 1, 2, 'a', 3, 4, 5, 6, 7]
>>> l1.remove('a')  # what goes in can come out
>>> l1
[0, 1, 2, 3, 4, 5, 6, 7]
```


List Examples

```
>>> l1.pop()                 # remove and return last element
7
>>> l1
[0, 1, 2, 3, 4, 5, 6]
>>> l1.reverse()             # reverse order of elements
>>> l1
[6, 5, 4, 3, 2, 1, 0]
>>> l1.sort()                # elements must be comparable
>>> l1
[0, 1, 2, 3, 4, 5, 6]
>>> l2 = [4, 1.3, "dog"]
>>> l2.sort()                # elements must be comparable
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: '<' not supported between instances of 'str' and
    'float'
>>> l2.pop()                 # remove 'dog'
'dog'
>>> l2
[4, 1.3]
>>> l2.sort()                # int and float are comparable
>>> l2
[1.3, 4]
```

Random Shuffle

A useful method on lists is `random.shuffle()` from the `random` module.

```
>>> list1 = [ x for x in range(9) ]
>>> list1
[0, 1, 2, 3, 4, 5, 6, 7, 8]
>>> random.shuffle(list1)
>>> list1
[7, 4, 0, 8, 1, 6, 5, 2, 3]
>>> random.shuffle(list1)
>>> list1
[4, 1, 5, 0, 7, 8, 3, 2, 6]
>>> random.shuffle(list1)
>>> list1
[7, 5, 2, 6, 0, 4, 3, 1, 8]
```


Processing CSV Lines

Suppose grades for a class were stored in a list of csv strings, such as:

```
student_data = ['Alice, 90, 75',
                'Robert, 8, 77',
                'Charlie, 60, 80']
```

Here the fields are: Name, Midterm grade, Final Exam grade.

Compute the average for each student and print a table of results.

Processing CSV Lines from List

```
def print_test_scores(student_data):
    """Print the test scores for the elements of student_data.

    student_data is a list of Strings. Each String is of the form:
    '<Name>, <Midterm Score>, <Final Score>'
    Course score is based on 1/3 of midterm score and 2/3s of
    final score.
    """
    print('Name          MT   FN  Course')
    print('----------------------------')
    for student in student_data:
        data = student.split(",")
        if len(data) != 3:
            print('Bad student data:', student)
        else:
            name = data[0].strip()
            midterm = int(data[1].strip())
            final = int(data[2].strip())
            course_score = midterm / 3 + final * 2 / 3
            print(format(name, '10s'), format(midterm, '4d'),
                  format(final, '4d'), format(course_score, '6.2f'))
```


Processing CSV Lines

```
students = ['Alice, 90 ,98', ' Robert ,58 ,77',
            'Michael, 80', 'Charlie, 60 ,80']
print_test_scores(students)
```

```
Name          MT   FN  Course
----------------------------
Alice         90   98  95.33
Robert        58   77  70.67
Bad student data: Michael, 80
Charlie       60   80  73.33
```

Copying Lists

Suppose you want to make a copy of a list. *The following won't work!*

```
>>> nums = [12, 56, 37, 12]
>>> n2 = nums
>>> n2 is nums
True
>>> n2 == nums
True
>>> n2[1] = 73
>>> n2
[12, 73, 37, 12]
>>> nums
[12, 73, 37, 12]
```


Copying Lists

But, many ways of making a copy of a list.

```
>>> nums
[12, 73, 37, 12]
>>> n2 = nums.copy()
>>> n2 is nums
False
>>> n3 = list(nums)
>>> n3 is nums
False
>>> n3 is n2
False
>>> n4 = nums[0:]
>>> n4 is nums
False
>>> n5 = [i for i in nums]
>>> n5 is nums
False
```

Passing Lists to Functions

Like any other *mutable* object, when you pass a list to a function, you're really passing a reference (pointer) to the object in memory.

```
def alter( lst ):
    lst.pop()

def main():
    lst = [1, 2, 3, 4]
    print( "Before call: ", lst )
    alter( lst )
    print( "After call: ", lst )

main()
```

```
> python ListArg.py
Before call:  [1, 2, 3, 4]
After call:  [1, 2, 3]
```


Let's Take a Break

Example Problems

To get good at working with lists, we must practice!

- CodingBat: https://codingbat.com/python
    - List1: first_last6, same_first_last, max_end3
    - List2: count_even, big_diff, has_22
- given list of ints or floats, is it sorted in descending order?
- get **last** index of a given value in list
- given two lists of ints, return a list that contains the difference between corresponding elements
    - change to be the max
- are all the elements of a given list unique? In other words, no duplicate values in the list
- given a list of ints place all even values before all odd values