Introduction to Programming in Python

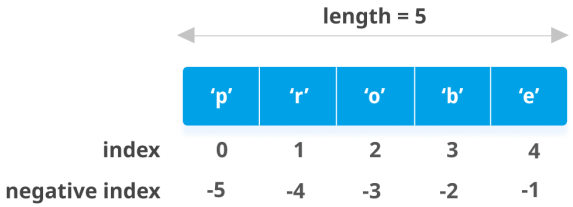
Lists

Dr. Bill Young
Department of Computer Science
University of Texas at Austin

Last updated: June 4, 2021 at 11:05

Lists

Lists are one of the most useful types in Python.



Both strings and lists 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 input 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].

In file `AverageScores.py`:

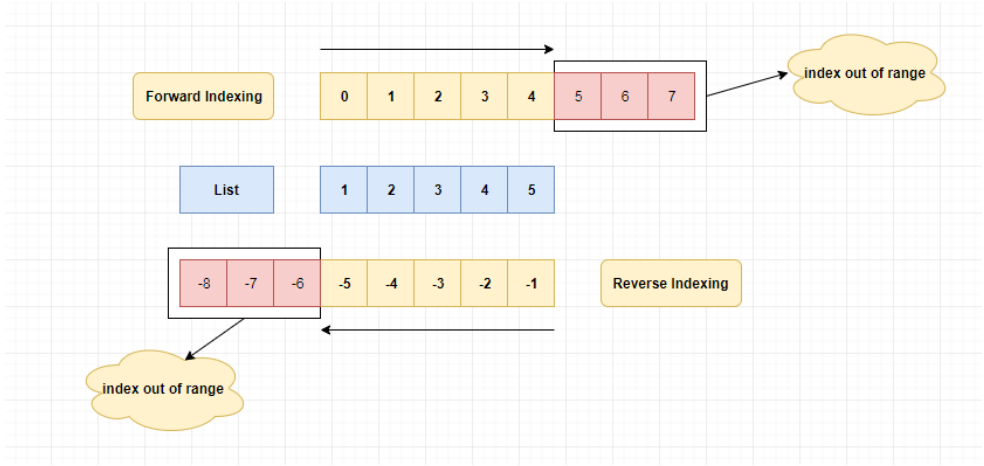
```
grades = [ 67, 82, 56, 84, 66, 77, 64, 64, 85, 67, \
           73, 63, 98, 74, 81, 67, 93, 77, 97, 65, \
           77, 91, 91, 74, 93, 56, 96, 90, 91, 99 ]

sum = 0
for score in grades:
    sum += score
average = sum / len(grades)
print("Class average:", format(average, ".2f"))
```

```
> python AverageScores.py
Class average: 78.60
```


Indexing and Slicing

Indexing and slicing on lists are as for strings, including negative indexes.



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']
```


Sequence Operations

Lists, like strings, 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 type(s) for +: 'int' and 'str'
>>> min(l2)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: '<' not supported between 'str' and 'int'
>>>
```


Using Functions

We could rewrite AverageScores.py as follows:

```
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"))
```

```
> python AverageScores.py
Class average: 78.60
```

Grade Example Using Lists

Exercise: Remember from Slideset 5, we solved a problem to compute and print out a Student Grade Report. The user input the student name and grades. Let's solve the same problem where the information is in a list.

Recall Susie Q. had grades:

```
Exam grades: 75/100, 85/90, 57/65
Project grades: 95/100, 150/200
```

But now we'll assume that the grades are given to the program in a list:

```
['Susie Q.', 75, 85, 57, 95, 150 ]
```


Grade Example Using Lists

```
EXAM1POINTS, EXAM2POINTS, EXAM3POINTS = 100, 90, 65
PROJ1POINTS, PROJ2POINTS = 100, 200

def printGradeReport( lst ):
    """ Print a grade report for the student.  The argument
    list contains the student's name and grades in format:
    [ name, exam1, exam2, exam3, proj1, proj2]. """
    student = lst[0]
    exam1Norm = (lst[1] / EXAM1POINTS) * 100.0
    exam2Norm = (lst[2] / EXAM2POINTS) * 100.0
    exam3Norm = (lst[3] / EXAM3POINTS) * 100.0

    proj1Norm = (lst[4] / PROJ1POINTS) * 100.0
    proj2Norm = (lst[5] / PROJ2POINTS) * 100.0

    # Compute the average of the three exams:
    examAvg = (exam1Norm + exam2Norm + exam3Norm) / 3

    # Compute the average of the two projects:
    projAvg = (proj1Norm + proj2Norm) / 2

    # Find the weighted average:
    courseAvg = examAvg * 0.6 + projAvg * 0.4
```


Grade Example Using Lists

```
    # Print the student's grade report.
    # (note same as previous versions)

    print()
    print("Grades for", student)
    print("  Exam1:", round(exam1Norm, 2))
    print("  Exam2:", round(exam2Norm, 2))
    print("  Exam3:", round(exam3Norm, 2))
    print("Exam average:", round(examAvg, 2))

    print("  Proj1:", round(proj1Norm, 2))
    print("  Proj2:", round(proj2Norm, 2))
    print("Proj average:", round(projAvg, 2))

    print("Course average:", round(courseAvg, 2))

def main():
    SusieRecord = ['Susie Q.', 75, 85, 57, 95, 150]
    printGradeReport( SusieRecord )

main()
```


Traversing Elements with a For Loop

General Form:

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

In file `forInListExamples.py`:

```
for i in [2, 3, 5, 7, 11, 13, 17]:
    print( i, end=", " )
print()

sum = 0
for j in range( 100 ):                  # Not really a list
    sum += j
print( "Sum:", sum )

print( "Squares:" )
for k in [ 1, 2, 3, 4, 5, 6, 7]:
    print( " "*k, k**2 )
print()
```

Traversing Elements with a For Loop

```
> python forInListExamples.py
2, 3, 5, 7, 11, 13, 17,
Sum: 4950
Squares:
  1
   4
    9
     16
      25
       36
        49
```


More List Methods

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

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

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 'str' and 'float'
>>> l2.pop()              # put the dog out
'dog'
>>> l2
[4, 1.3]
>>> l2.sort()             # int and float are comparable
>>> l2
[1.3, 4]
```

Splitting a String into a List

Splitting a string into a list of strings is often useful.

```
>>> str1 = "abc, def , ghi"
>>> str1.split(",")                # split on comma
['abc', ' def ', ' ghi']           # keeps whitespace
>>> strs = " abc def ghi "
strs.split()                       # split on whitespace
['abc', 'def', 'ghi']
>>> str3 = "\tabc\ndef\r ghi\n"
>>> str3.split()                   # split on whitespace
['abc', 'def', 'ghi']
>>> str4 = "abc / def / ghi"
>>> str4.split("/")                # split on slash
['abc ', ' def ', ' ghi']
```

Note `split` with no arguments splits on whitespace.

Copying Lists

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

```
>>> lst1 = [1, 2, 3, 4]
>>> lst2 = lst1
>>> lst1 is lst2      # there's only one list here
True
>>> print(lst1)
[1, 2, 3, 4]
>>> print(lst2)
[1, 2, 3, 4]
>>> lst1.append(5)    # changes to lst1 also change lst2
>>> print(lst2)
[1, 2, 3, 4, 5]
```

But you can do the following:

```
>>> lst2 = lst1[:]     # slicing creates a new copy
```


List Example: Counting Occurrences of Letters

Suppose we want to count the occurrences of letters in a given text. Here's an algorithm.

1. Create a list called "counts" of 26 zeros.
2. For each letter in the text
   - Convert it to lowercase
   - If it's the ith letter, increment `counts[i]` by 1
3. Print the counts list is a nice format.

List Example: Counting Occurrences of Letters

In file `CountOccurrencesInText.py`:

```
def countOccurrences( text ):
    """ Count occurrences of each of the 26 letters
    (upper or lower case) in text.  Return a list of
    counts in order. """

    # Create a list of 26 0's.
    counts = [0] * 26
    # Look at each character in text.
    for ch in text:
        # Make it lowercase.
        ch = ch.lower()
        # If it's alpha, count it.
        if ch.isalpha():
            # Turn the character into an index.
            index = ord( ch ) - ord( 'a' )
            counts[ index ] += 1
    return counts
```

List Example: Counting Occurrences of Letters

Now we want to print the counts in a nice format, 10 per line.

```
def printCounts( counts ):
    """ Print the letter counts 10 per line. """
    onLine = 0
    for i in range( 26 ):
        # Convert the index into the array into the
        # corresponding lower case letter.
        letter = chr(i + ord('a'))
        print( letter + ":", counts[i], end = " ")
        onLine += 1
        # If we've printed 10 on the line, go to the next
            line.
        if ( onLine == 10 ):
            print()
            onLine = 0
    print()
```


List Example: Counting Occurrences of Letters

```
def main():
    txt = """Once upon a midnight dreary, while I pondered,
          weak and weary, Over many a quaint and curious
          volume of forgotten lore."""
    counts = countOccurrences( txt )
    printCounts( counts )

main()
```

```
> python countOccurrencesInText.py
a: 9 b: 0 c: 2 d: 6 e: 11 f: 2 g: 2 h: 2 i: 6 j: 0
k: 1 l: 3 m: 3 n: 9 o: 10 p: 2 q: 1 r: 8 s: 1 t: 4
u: 5 v: 2 w: 3 x: 0 y: 3 z: 0
```


Searching a List

A common operation on lists is **searching**. To search a list means to see if a value is in the list.

If all you care about is *whether or not* `lst` contains value `x`, you can use: `x in lst`.

Often you want to know the *index* of the occurrence, if any.

There are many different search methods depending on the properties of the list.

Linear Searching

If the list is not *sorted*, often the best you can do is look at each element in turn. This is called a **linear search**.

From file `LinearSearch.py`:

```
def linearSearch( lst, key ):
    for i in range( len(lst) ):
        if key == lst[i]:
            return i
    return -1
```

If the item is present, you stop as soon as you find it. On average, how many comparisons would you expect to make if the item is there? How many if it's not there?

Linear Searching

```
>>> from LinearSearch import *
>>> lst = [1, 3, 5, 7, 9]
>>> linearSearch( lst, 7 )
3
>>> linearSearch( lst, 1 )
0
>>> linearSearch( lst, 8 )
-1
>>> linearSearch( [1, 2, 1, 2, 1, 2], 2 )
1
```

We use -1 to indicate that the item is not in the list, since -1 is not a legal index.

Find Multiple Occurrences

Notice that `linearSearch` only finds the *first* occurrence of the key. To find all, you might do:

```
def findAllOccurrences( lst, key ):
    # Return a list of indexes of occurrences
    # of key in lst.
    found = []
    for i in range( len(lst) ):
        if key == lst[i]:
            found.append( i )
    return found
```

```
>>> from LinearSearch import *
>>> findAllOccurrences( [1, 2, 1, 2, 1, 2], 2)
[1, 3, 5]
```

Here you do have to search the whole list.

Using Index

You can use `index` to do linear search *if you know that the item is present*.

```
>>> lst = [ 9, 3, 5, 7, 1, 2, 4, 8 ]
>>> lst.index( 7 )
3
>>> lst.index( 10 )
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: 10 is not in list
>>>
```

The `index` method is almost certainly implemented using linear search.

Two-Dimensional Lists

Recall that lists in Python are *heterogeneous*, meaning that you can have items of various types. Lists items can themselves be lists, lists of lists, etc.

```
>>> gradeSheet = [ ['Susie Q.',   75, 85, 57, 95, 150], \
                   ['Frank G.',   85, 90, 49, 24, 125], \
                   ['Albert A.',  95, 70, 65, 82,  99], \
                   ['Charles T.', 70, 82, 54, 80, 186] ]
>>> gradeSheet[0]
['Susie Q.', 75, 85, 57, 95, 150]
>>> gradeSheet[0][0]
'Susie Q.'
>>> gradeSheet[2][3]
65
```

Note that if the item at `lst[i]` is itself a list, you can index into that list. You can think of them as row and column indexes.

Grade Example with List of Lists

Suppose we have a GradeSheet like this one:

```
gradeSheet = [ ['Susie Q.',   75, 85, 57, 95, 150],
               ['Frank G.',   85, 90, 49, 24, 125],
               ['Albert A.',  95, 70, 65, 82,  99],
               ['Charles T.', 70, 82, 54, 80, 186] ]
```

How would we change our previous Grading program to print grade reports for each of the students?

Grade Example with List of Lists

In file `Grade4.py`:

```
from Grade3 import printGradeReport

def main():
    # This uses our printGradeReport from Grade3.py.

    gradeSheet = [ ['Susie Q.',   75, 85, 57, 95, 150], \
                   ['Frank G.',   85, 90, 49, 24, 125], \
                   ['Albert A.',  95, 70, 65, 82, 99], \
                   ['Charles T.', 70, 82, 54, 80, 186] ]
    for studentRecord in gradeSheet:
        printGradeReport( studentRecord )
        print()

main()
```

Also comment out the call to `main()` in `Grade3.py` so it won't run when you import it.

Running Grade4.py

```
> python Grade4.py

Grades for Susie Q.
  Exam1: 75.0
  Exam2: 94.44
  Exam3: 87.69
Exam average: 85.71
  Proj1: 95.0
  Proj2: 75.0
Proj average: 85.0
Course average: 85.43

Grades for Frank G.
   ...

Grades for Albert A.
   ...

Grades for Charles T.
   ...
```