Lists

adapted from material by Mike Scott and Bill Young at the University of Texas at Austin

The list class is one of the most useful in Python. A sequence of elements which can be accessed.

Two major differences:

- Strings are sequences of characters, while lists can be sequences of *anything*.
- Strings are immutable, lists are mutable.

When you change a list, it doesn't make a new copy--it changes the actual contents of the list.

Suppose you have 30 different test grades to average.
Is this a good solution?

```
1  with open("grades.txt", "r") as infile:
2    grade1 = int(infile.readline())
3    grade2 = int(infile.readline())
4    grade3 = int(infile.readline())
5    grade4 = int(infile.readline())
6    grade5 = int(infile.readline())
7    grade6 = int(infile.readline())
8    grade7 = int(infile.readline())
9    grade8 = int(infile.readline())
10   grade9 = int(infile.readline())
11   grade10 = int(infile.readline())
12   grade11 = int(infile.readline())
13   grade12 = int(infile.readline())
14   grade13 = int(infile.readline())
15   grade14 = int(infile.readline())
16   grade15 = int(infile.readline())
17
18 total = grade1 + grade2 + grade3 + grade4 + grade5\
19       + grade6 + grade7 + grade8 + grade9 + grade10\
20       + grade11 + grade12 + grade13 + grade14 + grade15
21 average = total / 15
22 print(f"Class average is {average}")
```

What's wrong with this solution?

```
1  grades = []
2  with open("grades.txt", "r") as infile:
3    for line in infile:
4          g = int(line)
5          grades.append(g)
6
7  total = 0
8  for score in grades:
9    total += score
10 average = total / len(grades)
11 print(f"Class average is {average}")
```

Note that we're using a for-loop here. Previously, for-loops were not that useful, but they are a natural fit for looping through lists!

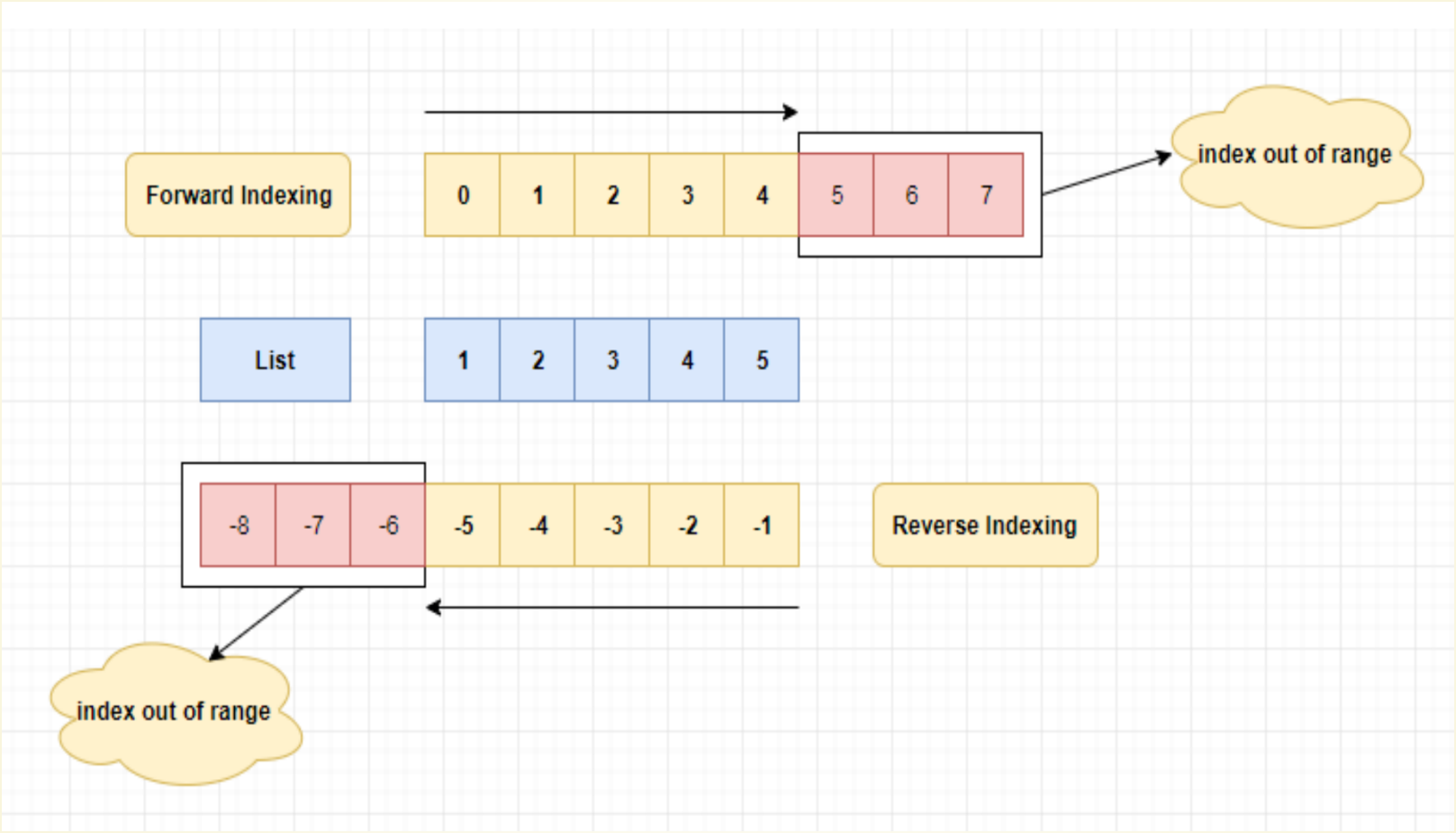
Operations on Lists

Indexing

Suppose we have a list with 10 elements.
We can get elements by *indexing* them.

```
1 lst = [1, 3, 5, 7, 9, 11, 13, 15, 17]
2
3 print(lst[0])
4 print(lst[3])
5 print(lst[-1])
6 print(lst[100])
```

Indexing out-of-bounds will give us an error.


Forward Indexing
0
1
2
3
4
5
6
7
index out of range
List
1
2
3
4
5
-8
-7
-6
-5
-4
-3
-2
-1
Reverse Indexing
index out of range

Exercise

Create a list with the numbers 1 through 10.

Then, double each number inside the list, so that we get [2,4,6,8,10,12,14,16,18,20].

Can you triple each number? Quadruple it?

Hint: to get list length, use the len() function

Slicing

Can gather elements of lists into a new list.

```
list[start:end]
```

If start is not given, assumes zero

If end is not given, assumes len(list).

Like in ranges, the last element is *not* included.

```
1 lst = [1, 3, 5, 7, 9, 11, 13, 15, 17]
2
3 print(lst[0:])
4 print(lst[4:])
5 print(lst[:3])
6 print(lst[5:-2])
```

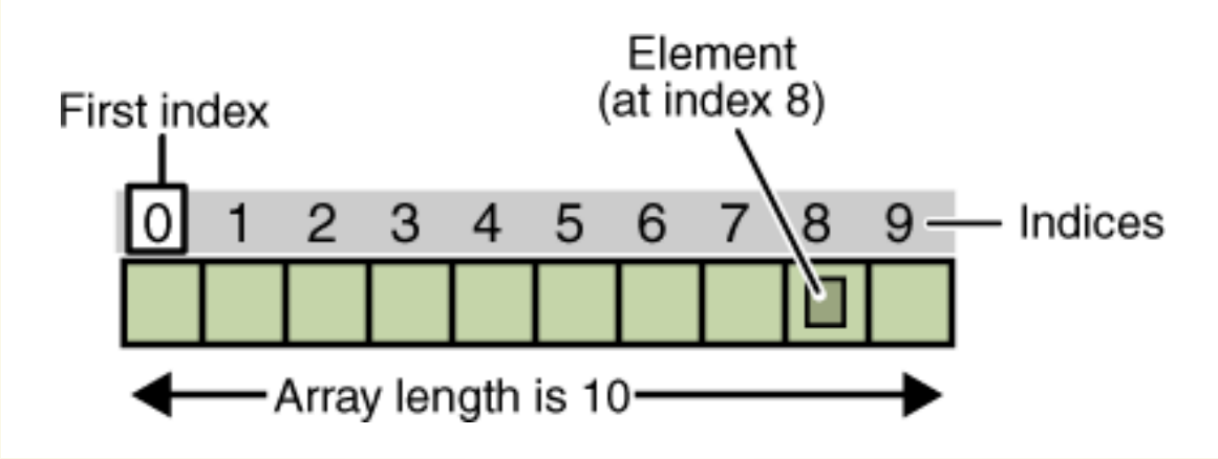
Notice how I named my list "lst" instead of "list"? That's because list is a built-in function.

Same reason we don't name strings "str" or files "file".

```
In [1]: list()
Out[1]: []

In [2]: list([1,2,3])
Out[2]: [1, 2, 3]

In [3]: list(["red", 4, 9.9])
Out[3]: ['red', 4, 9.9]

In [4]: range(4)
Out[4]: range(0, 4)

In [5]: list(range(4))
Out[5]: [0, 1, 2, 3]

In [6]: list("abcd")
Out[6]: ['a', 'b', 'c', 'd']
```

Lists vs Arrays

Many other languages have something called an "array" type. Python lists are similar, but much more powerful.



Arrays are

- All same element type
- Fixed size
- very fast access time

Lists are

- possibly mixed element types
- variable size
- fast access time

What kinds of operations should we put on lists?

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 sequences n times
s[i]	Get i-th element of sequence
s[i:j]	Slice sequence from i to j-1
len(s)	Get length of sequence
min/max/sum	Compute min/max/sum, if possible
> >= < <= == !=	Compare lists

```
In [1]: list_1 = [1,2,3,4,5]

In [2]: len(list_1)
Out[2]: 5

In [3]: min(list_1)
Out[3]: 1

In [4]: max(list_1)
Out[4]: 5

In [5]: sum(list_1)
Out[5]: 15

In [6]: list_2 = [1, 2, "red"]

In [7]: 3 in list_2
Out[7]: False

In [8]: "red" in list_2
Out[8]: True

In [9]: min(list_2)
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [9], in <cell line: 1>()
----> 1 min(list_2)

TypeError: '<' not supported between instances of 'str' and 'int'
```

Exercise

Given two input lists and a target number,
find out if the target is in the first list,
second list, both, or neither.

How should we represent this output?

Grade averages, a better version

```
1 grades = []
2 with open("grades.txt", "r") as infile:
3   g = int(infile.readline())
4   grades.append(g)
5
6 average = sum(grades) / len(grades)
7 print(f"Class average is {average}")
```

Comparing Lists

We compare the list *lexicographically*: if first elements are uneqal, return as-is. If they are equal, continue to the next, and so on.

```
In [1]: list1 = ["red", 3, "green"]

In [2]: list2 = ["red", 3, "gray"]

In [3]: list3 = ["red", 5, "green"]

In [4]: list4 = [5, "red", "green"]

In [5]: list1 < list2
Out[5]: False

In [6]: list2 == list1
Out[6]: False

In [7]: list3 > list1
Out[7]: True

In [8]: list3 < list4
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [8], in <cell line: 1>()
----> 1 list3 < list4

TypeError: '<' not supported between instances of 'str' and 'int'
```

Loops and Comprehensions

As mentioned, we can use for-loops to easily iterate over all elements of a list.

```
1 list1 = [1, 3, 5, 7]
2 for elem in list1:
3   print(elem, end=" ")
```

Could use a while-loop, but clunkier

```
1 list1 = [1, 3, 5, 7]
2 index = 0
3 while index < len(list1)
4   elem = list1[index]
5   print(elem, end=" ")
```

We can build lists using *list comprehension* syntax.

```
In [1]: range(4)
Out[1]: range(0, 4)

In [2]: [ x for x in range(4) ]
Out[2]: [0, 1, 2, 3]

In [3]: [ x**2 for x in range(4) ]
Out[3]: [0, 1, 4, 9]

In [4]: lst = [2, 3, 5, 7, 11, 13]

In [5]: [ x ** 3 for x in lst ]
Out[5]: [8, 27, 125, 343, 1331, 2197]

In [6]: [ x for x in lst if x > 2 ]
Out[6]: [3, 5, 7, 11, 13]

In [7]: [ s[0] for s in ["red", "green", "blue"] if s <= "green" ]
Out[7]: ['g', 'b']

In [8]: [ x for x in range(100) if isPrime(x) ]
```

List comprehensions let us build lists really easily, even from files!

```
1 with open("grades.txt", "r") as infile:
2   grades = [ int(entry) for entry in infile ]
3
4 total = 0
5 for score in grades:
6   total += score
7 average = total / len(grades)
8 print(f"Class average is {average}")
```

Example

Build an even filtering function. It takes an input list and *returns a new list* which contains elements of a particular type.

Do this in a single line with list comprehensions!

Project Proposals

Will be due at the same time as HW 8.

Come up with an idea for a small project you can write with Python.

Doesn't have to be flashy or traditionally-programming related.

Ideas

- Write a simple 2d game (e.g. 2D racing, or Breakout/Tetris)
- Write a chemical network simulator with a simple variant of the Gillespie algorithm
- Write a program which simulates a bridge and highlights where the weak points are.
- Create a simple scanner for known malware (e.g. computer virus) files

Written proposal is so that I can look and see if the project is reasonably-scoped!

Proposal Contents

- Your name (and partner's name, if applicable)
- Description of what you want to do
- How you're going to meet the project requirements:
  - Some input method
  - Some output method
  - Code organization requirements
- Two examples of things your program will do
- Three examples of things your program will **not** do

More List Methods

These methods work for lists, not sequences in general. Note they change the list.

Method	**Description**
t.append(x)	add x to the end of t
t.count(x)	count how many times x shows up in t
t.extend(l1)	append elements of l1 to t
t.index(x)	index of first occurrence of x in t
t.insert(x, i)	insert x into t at position i
t.pop()	remove+return the last element of t
t.pop(i)	remove+return the i-th element of t
t.remove(x)	remove the first occurrence of x from t
t.reverse()	reverse the elements of t
t.sort()	sort the elements of t

Common mistake

```
1 list1 = [1,2,3,4,5]
2 list1 = list1.sort()
3 print(f"List 1 is {list1}")
```

```
In [3]: list1 = [1,2,3,4,5]
   ...: list1 = list1.sort()
   ...: print(f"List 1 is {list1}")
List 1 is None
```

```
In [1]: l1 = [1,2,3]

In [2]: l1.append(4)

In [3]: l1
Out[3]: [1, 2, 3, 4]

In [4]: l1.count(4)
Out[4]: 1

In [5]: l2 = [5,6,7]

In [6]: l1.extend(l2)

In [7]: l1
Out[7]: [1, 2, 3, 4, 5, 6, 7]

In [8]: l1.index(5)
Out[8]: 4

In [9]: l1.insert(0, 0)

In [10]: l1
Out[10]: [0, 1, 2, 3, 4, 5, 6, 7]
```

```
In [10]: l1
Out[10]: [0, 1, 2, 3, 4, 5, 6, 7]

In [11]: l1.insert(3, 'a')

In [12]: l1
Out[12]: [0, 1, 2, 'a', 3, 4, 5, 6, 7]

In [13]: l1.remove('a')

In [14]: l1
Out[14]: [0, 1, 2, 3, 4, 5, 6, 7]

In [15]: l1.pop()
Out[15]: 7

In [16]: l1.reverse()

In [17]: l1
Out[17]: [6, 5, 4, 3, 2, 1, 0]
```

```
In [17]: l1
Out[17]: [6, 5, 4, 3, 2, 1, 0]

In [18]: l1.sort()

In [19]: l1
Out[19]: [0, 1, 2, 3, 4, 5, 6]

In [20]: l2 = [4, 1.3, "dog"]

In [21]: l2.sort()
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [21], in <cell line: 1>()
----> 1 l2.sort()

TypeError: '<' not supported between instances of 'str' and 'float'

In [22]: l2.pop()
Out[22]: 'dog'

In [23]: l2.sort()

In [24]: l2
Out[24]: [1.3, 4]
```

Exercise

List complement: given a list which contains some numbers in [0..10], return a second list which contains all numbers in [0..10] not in the first.

Example:

- Input: [1,3,5,7,9]
- Output: [0,2,4,6,8,10]

Do this without using comprehensions (prefer .append)

Everyday I'm shufflin'

Another useful method on lists is random.shuffle() from the random module, which randomizes the order of a list.

```
In [1]: import random

In [2]: l1 = [1,2,3,4,5]

In [3]: random.shuffle(l1)

In [4]: l1
Out[4]: [1, 2, 5, 3, 4]

In [5]: random.shuffle(l1)

In [6]: l1
Out[6]: [2, 5, 3, 1, 4]

In [7]: random.shuffle(l1)

In [8]: l1
Out[8]: [2, 5, 4, 1, 3]

In [9]: random.shuffle(l1)

In [10]: l1
Out[10]: [3, 5, 1, 2, 4]
```

List Mutability

Surprises and Traps

What does this code do?

```
1  nums = [12, 56, 37, 12]
2  n2 = nums
3  n2[1] = 73
4  print(nums)
```

```
In [1]: nums = [12, 56, 37, 12]

In [2]: n2 = nums

In [3]: n2[1] = 73

In [4]: print(nums)
[12, 73, 37, 12]

In [5]: print(n2)
[12, 73, 37, 12]
```

Lots of ways to make a copy

```
1  l = [1,2,3,4,5]
2  l1 = l                  # Not a copy! Same list!
3  l2 = l.copy()           # makes a copy
4  l3 = list(l)            # makes a copy
5  l4 = l[:]               # makes a copy
6  l5 = [i for i in l]  # makes a copy
```

Passing to Function

When you pass a list to a function, the original can be changed.

```
1  def alter_list(lst):
2    lst.pop()
3
4  def main():
5    l1 =[1,2,3,4]
6    print("Before call:", l1)
7    alter(l1)
8    print("After call:", l2)
9
10 main()
```

Practice

Working with lists can be hard!

Even 2nd and 3rd year computer science students struggle with some tasks.

To get better, we have to practice.

Practice Problems

- Given a list of numbers (either int or float), check if it is stored in ascending order
- Get last index of a given value in a list (opposite of .index() method)
- Given two array of ints, return an array that contains difference between corresponding elements.
  - What about max? Sum?
  - What do we do if it's a different size?
- Are all elements of a given list unique?
- Given a list of ints, place all even values before any odd values.

Even More Practice

- https://codingbat.com/python
- List-1 and List-2 problem sets