CHAPTER 10
LISTS

ACKNOWLEDGEMENT: THESE SLIDES ARE ADAPTED FROM SLIDES PROVIDED WITH INTRODUCTION TO PROGRAMMING USING PYTHON, LIANG (PEARSON 2013)
MOTIVATION

• Read one hundred numbers, compute their average, and find out how many numbers are above the average.

• Store and manipulate large amounts of data
  • 52 playing cards in a deck
  • 3 thousand undergrads at UR
  • 140 characters per Tweet
  • 4 billion nucleotides in a DNA strand
  • 50 trillion cells in the human body
  • $6.022 \times 10^{23}$ particles in a mole
INTRODUCING LISTS

• **List** is a data structure that represents a collection of data of any size.

• List objects are **references**.

```python
myList = [5.6, 4.5, 3.3, 13.2, 4.0, 34.33, 34.0, 45.45, 99.993, 11123]
```

![Diagram showing list elements and references]
TRACE PROBLEM WITH LISTS

1. `values = [0, 0, 0, 0, 0]`
2. `for i in range(1, 5):`
3. `values[i] = i + values[i-1]`

Create a list with 5 zeroes in it named values

<table>
<thead>
<tr>
<th>Index</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>
TRACE PROBLEM WITH LISTS

1. values = [0, 0, 0, 0, 0, 0]
2. for i in range(1, 5):
3.   values[i] = i + values[i-1]

<table>
<thead>
<tr>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>values 0xA</td>
</tr>
<tr>
<td>i    1</td>
</tr>
<tr>
<td>0xA</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Index</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>
1. values = [0, 0, 0, 0, 0, 0]
2. for i in range(1, 5):
3.   values[i] = i + values[i-1]
TRACE PROBLEM WITH LISTS

1. values = [0, 0, 0, 0, 0]

2. for i in range(1, 5):

3.   values[i] = i + values[i-1]


<table>
<thead>
<tr>
<th>Index</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>
TRACE PROBLEM WITH LISTS

1. values = [0, 0, 0, 0, 0, 0]
2. for i in range(1, 5):
3.     values[i] = i + values[i-1]

Memory
values  0xA
i  2
0xA

<table>
<thead>
<tr>
<th>Index</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>

Set values[2] to 3
TRACE PROBLEM WITH LISTS

1. values = [0, 0, 0, 0, 0, 0]
2. for i in range(1, 5):
3.     values[i] = i + values[i-1]

Memory
values  0xA
i       3
0xA

<table>
<thead>
<tr>
<th>Index</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>
TRACE PROBLEM WITH LISTS

1. values = [0, 0, 0, 0, 0]
2. for i in range(1, 5):
3.     values[i] = i + values[i-1]

Memory:
values  0xA
i  3
0xA

Set values[3] to 6
TRACE PROBLEM WITH LISTS

1. values = [0, 0, 0, 0, 0]
2. for i in range(1, 5):
3.   values[i] = i + values[i-1]

<table>
<thead>
<tr>
<th>Index</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>

Memory
values  0xA
i       4
0xA
TRACE PROBLEM WITH LISTS

1. `values = [0, 0, 0, 0, 0]`
2. `for i in range(1, 5):`
3. `values[i] = i + values[i-1]`

Set values[4] to 10

Memory

<table>
<thead>
<tr>
<th>Index</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
</tr>
</tbody>
</table>
TRACE PROBLEM WITH LISTS

1. values = [0, 0, 0, 0, 0]
2. for i in range(1, 5):
3.   values[i] = i + values[i-1]

The end has been reached.
TRACE PROBLEM WITH LISTS

1. values = [0, 0, 0, 0, 0]
2. for i in range(1, 5):
3.   values[i] = i + values[i-1]

Set values[0] to 11
EXERCISES AS A TABLE

- With using just looping over a list
  - 1 – Printing a list
  - 2 – Summing a list of numbers
  - 3 – Finding the largest element in a list
  - 4 – Finding the largest index of the smallest element in a list
- 1,3 – Randomly shuffling a list
- 2,4 – Rotate the elements of a list by 1 index
LIST SYNTAX AND OPERATORS
CREATING LISTS

- You can create lists using the list class constructor:

  list1 = list()  # Create an empty list
  list2 = list([2, 3, 4])  # Create a list with elements 2, 3, 4
  list3 = list(["red", "green", "blue"])  # Create a list with strings
  list4 = list(range(3, 6))  # Create a list with elements 3, 4, 5
  list5 = list("abcd")  # Create a list with characters a, b, c

- For convenience, you may create a list using the following syntax:

  list1 = []  # Same as list()
  list2 = [2, 3, 4]  # Same as list([2, 3, 4])
  list3 = ["red", "green"]  # Same as list(["red", "green"])
## LIST METHODS

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>append(x: object): None</code></td>
<td>Add an item x to the end of the list.</td>
</tr>
<tr>
<td><code>insert(index: int, x: object): None</code></td>
<td>Insert an item x at a given index. Note that the first element in the list has index 0.</td>
</tr>
<tr>
<td><code>remove(x: object): None</code></td>
<td>Remove the first occurrence of the item x from the list.</td>
</tr>
<tr>
<td><code>index(x: object): int</code></td>
<td>Return the index of the item x in the list.</td>
</tr>
<tr>
<td><code>count(x: object): int</code></td>
<td>Return the number of times item x appears in the list.</td>
</tr>
<tr>
<td><code>sort(): None</code></td>
<td>Sort the items in the list.</td>
</tr>
<tr>
<td><code>reverse(): None</code></td>
<td>Reverse the items in the list.</td>
</tr>
<tr>
<td><code>extend(l: list): None</code></td>
<td>Append all the items in L to the list.</td>
</tr>
<tr>
<td><code>pop([i]): object</code></td>
<td>Remove the item at the given position and return it. The square bracket denotes that parameter is optional. If no index is specified, list.pop() removes and returns the last item in the list.</td>
</tr>
</tbody>
</table>
BUILT-IN FUNCTIONS FOR LISTS

- Let:
  \[ l = [2, 3, 4, 1, 32] \]

- `len(l)` — computes the number of entries in the list (in this case 5)
- `max(l)` — computes the maximum element of the list (in this case 32)
- `min(l)` — computes the minimum element of the list (in this case 1)
- `sum(l)` — computes the summation of the elements in the list (in this case 42)

- Other libraries contain more functionality. Example of shuffling a list:
  ```python
  import random
  random.shuffle(l)  # Shuffle the items in the list
  print(l)  # Shows: [4, 1, 2, 32, 3]
  ```
INDEXER OPERATOR []

- The index operator [ ] selects the object at a specific location (index) in the data
THE +, *, [ : ], AND IN OPERATORS

• Lists are similar to strings. Consider:

11 = [2, 3]
12 = [1, 9]
13 = 11 + 12  # 13 contains [2, 3, 1, 9]
14 = 3*11     # 14 contains [2, 3, 2, 3, 2, 3]
15 = 14[2:4]  # 15 contains [2, 3]

contains4 = 4 in 15                      # contains4 stores False
doesn'tcontain5 = 5 not in 15            # doesn'tcontains5 stores True
THE +, *, [ : ], AND IN OPERATORS

• + – is an operator that concatenates (joins/append) two lists and returns the result

• * – is an operator that repeats a list some amount of times and returns the result (called the repetition operator)

• [:] – is an operator that returns a sublist of the list, called the slicing operator. The slice returned begins at the first index and ends at the second index -1.

• in and not in – are containment operators returning Boolean values whether an object/sublist is contained/not contained within a list.
NEGATIVE INDEX IN A SLICING OPERATOR

• Consider:

\[
l_1 = [2, 3, 5, 2, 33, 21]\]

\[
\text{print}(l_1[-1]) \# 21
\]

\[
\text{print}(l_1[-3]) \# 2
\]

• A negative index counts from the end of the list
COMMON PITFALL
OFF-BY-ONE ERRORS

• Be careful of indexing and slicing operators, it is easy to get an index that is not valid.

• Consider:
  
  ```
  lst = [0, 1, 2, 3]
  i = 0
  while i <= len(lst):
    print(lst[i])
    i += 1
  ```

• This code generates a tracing error:
  ```
  IndexError: list index out of range
  ```
LIST COMPREHENSION

- List comprehensions provide a very concise syntax for generating lists.
- A list comprehension consists of brackets containing an expression followed by a for clause, then zero or more for or if clauses.
- The result will be a list resulting from evaluating the expression.

- Compare the following:
  ```python
  l1 = list()
  for x in range(0, 5):
    l1.append(x)
  # Generates [0, 1, 2, 3, 4]
  ```

- To using a list comprehension:
  ```python
  l1 = [x for x in range(0, 5)]
  # Generates [0, 1, 2, 3, 4]
  ```

- Other examples:
  ```python
  l2 = [0.5 * x for x in l1]
  # Generates [0.0, 0.5, 1.0, 1.5, 2.0]
  l3 = [x for x in l2 if x < 1.5]
  # Generates [0.0, 0.5, 1.0]
  ```
COMPARING LISTS

• Relational operators work on lists as well. It considers lexicographic ordering (a more general form of alphabetical order). Consider:

```python
l1 = ["red", "blue", "green"]
l2 = ["red", "green", "blue"]
l1 == l2  # False
l1 != l2  # True
l1 >  l2  # False
l1 >= l2  # False
l1 <  l2  # True
l1 <= l2  # True
```
SPLITTING A STRING TO A LIST

• Often we need to split strings based on a delimiter (e.g., space). The string method `split` generates a list.

• Example:
  
  ```python
  items = "Welcome to the US".split()
  print(items)  # ['Welcome', 'to', 'the', 'US']
  
  items = "34#13#78#45".split("#")
  print(items)  # ['34', '13', '78', '45']
  ```
COPYING LISTS

• Often, in a program, you need to duplicate a list or a part of a list.

• In such cases, you could attempt to use the assignment statement (\(=\)), as follows:
  
  \[
  l_2 = l_1
  \]

• But, this copies the reference, not the list. Do this trick instead:
  
  \[
  l_2 = [] + l_1
  \]
  
  # Append to a new list
PASSING LISTS TO FUNCTIONS

• Passing lists to functions is perfectly normal. Consider:

```python
def printList(lst):
    for x in lst:
        print(x, end=" ")
    print()

l = [3, 1, 2, 6, 4, 2]
printList(l)

printList(["Hi", 5, 2.3]) # Anonymous list
```
PASSING LISTS TO FUNCTIONS

• Python uses *pass-by-object-reference* to pass arguments to a function. There are important differences between passing the values of variables of numbers and strings and passing lists.
  • Immutable objects act like pass-by-value (numbers and strings)
  • Mutable objects can have their memory altered (lists and other objects)
PASSING LISTS TO FUNCTIONS

EXAMPLE

def main():
    x = 1  # x represents an int value
    y = [1, 2, 3]  # y represents a list
    m(x, y)    # Invoke f with arguments x and y
    print("x is " + str(x))  # Prints 1, not 1001
    print("y[0] is " + str(y[0]))  # Prints 5555

def m(number, numbers):
    number = 1001  # Assign a new value to number
    numbers[0] = 5555  # Assign a new value to numbers[0]

main()
SUBTLE ISSUES REGARDING DEFAULT ARGUMENTS

```
def add(x, lst = []):  
    if x not in lst:  
        lst.append(x)  
    return lst

l1 = add(1)  
print(l1)    # [1]
l2 = add(2)  
print(l2)    # [1, 2]
l3 = add(3, [11, 12, 13])
print(l3)    # [11, 12, 13, 3]
l4 = add(4)  
print(l4)    # [1, 2, 4]
```

- Default values are only created only once.
- Consider this program. Its output is:
  
  [1]
  [1, 2]
  [11, 12, 13, 3]
  [1, 2, 4]
Returning a list from a function is also normal. Consider:

```python
def reverse(lst):
    result = []
    for x in lst:
        result.insert(0, x)
    return result
```

```python
l1 = [1, 2, 3, 4, 5, 6]
l2 = reverse(l1)
```

Note that list already has the reverse method `list.reverse()`
SUMMARY

• Lists.
  • Organized way to store huge quantities of data.
  • Almost as easy to use as primitive types.
  • Can directly access an element given its index.

http://imgs.xkcd.com/comics/Donald_Knuth.png