Sequence Types

FEB 27TH, 2012

What we have not learned so far…

- How to store, organize, and access large amounts of data?
- **Examples:**
  - **Read a sequence of million numbers and output these in sorted order.**
  - **Read a text, correct all spelling errors in the text, and output the corrected text.**
- Programming languages typically provide tools and techniques to store and organize data. In Python we can use *sequence types* to do this.

Strings and *Lists* are examples of Sequence Types

- A *string* is a sequence of characters enclosed in quotes.
  Examples: "hello", "8.397", "7", '34'
  (The quotes can be single or double quotes)

- A *list* is a sequence of objects enclosed in square brackets.
  Examples: [0, 1, 2, 3], ["Alice", "Bob", "Catherine"],
  ["hello", 4.567, -22, 87L, 'bye']
  (Objects of different types can be part of the same list)

- Lists are more "general" than strings; strings can be viewed as special instances of lists.

Two simple operations on lists

- The `in` operator is used as `x in L`, where `x` is an object and `L` is a list. This expression evaluates to `True` if `x` is an *element* in `L`; evaluates to `False` otherwise.

  Examples: `67 in [34, 12, 45]` evaluates to `False`

  `"hi" in []` evaluates to `False`, etc.

- Python has a built-in function `len(L)` that returns the length, i.e., the number of elements, in list `L`.

  Examples: `len([])` is `0`, `len([34, 12, 45])` is `3`, etc.

Both of these work on strings as well

Examples:

"hi" in "history" evaluates to True

"ei" in "piece" evaluates to False

"ace" in "Wallace" evaluates to True

Examples:

len("history") returns 7

len("") returns 0

len("piece") returns 5

Generating lists

- Python has a built-in function called `range` that allows us to generate lists using *arithmetic progressions*.
- It can have one, two, or three arguments, all of which must be integers.

```
>>> range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
 >>> range(1, 11)
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> range(0, 30, 5)
[0, 5, 10, 15, 20, 25]
>>> range(0, 10, 3)
[0, 3, 6, 9]
>>> range(0, -10, -1)
[0, -1, -2, -3, -4, -5, -6, -7, -8, -9]
>>> range(0)
[]
>>> range(1, 0)
 []
```

The range function is useful in for-loops

```
for i in range(1, 10, 2):
        print i*i
```

- Repeats the execution of the body of the for-loop for each value of i = 1, 3, 5, 7, and 9.
- Equivalent to

```
i = 1
while i < 10:
        print i*i
        i = i + 2
```

- But more convenient for simple loops because no need to initialize before loop and no need to update within loop.

More examples of `for`-loops

```
L = ["hello", "hi", "bye"]
for e in L:
    print e + e
```

```
s = "What is this sentence?"
for ch in s:
    print ch
```

Generating Lists: Initialization

- Here is another useful way of generating lists , particularly for initializing them, i.e., assign them "initial" values at the start of a program.

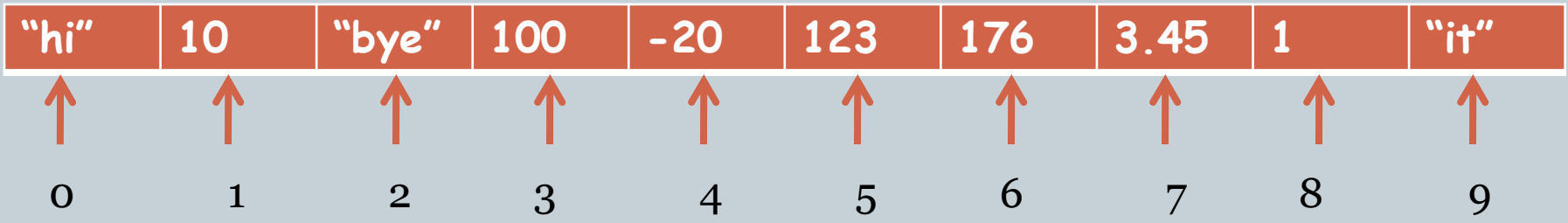
Example:

```
n = 25
L = [8]*n
```

This assigns to L a list of length 25 consisting of the integer 8.

Accessing lists and strings

L = ["hi", 10, "bye", 100, -20, 123, 176, 3.45, 1, "it"]



- One of the most useful features of sequence types is that elements in a sequence can be accessed efficiently and conveniently using their *position* in the sequence.
- **Example:**

  L[0] is "hi", L[1] is 10, L[2] is "bye", ..., L[9] is "it"

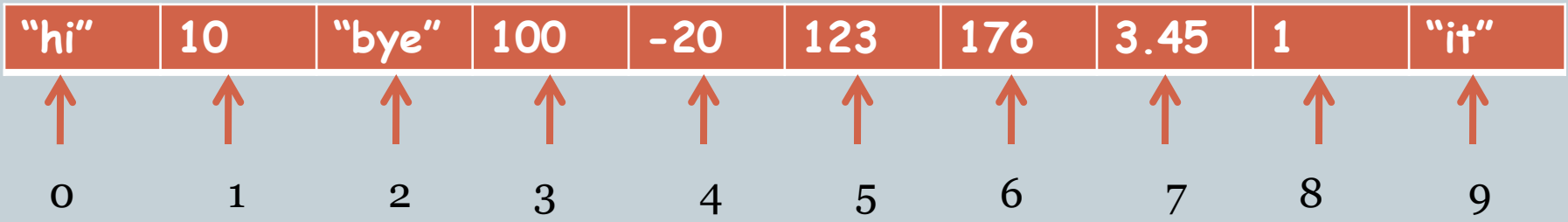
Example

- This program walks through the list, printing each element.
- The program uses the positions of the elements to index into the list.

```
L = ["hi", 109, "go", 111, 1.16, [122,30], "hello"]
i = 0
while i < len(L):
    print L[i]
    i = i + 1
```

Accessing slices of lists and strings

L = ["hi", 10, "bye", 100, -20, 123, 176, 3.45, 1, "it"]



- L[2:5] is ["bye", 100, -20]
- L[:2] is ["hi", 10]
- L[4:4] is []
- L[4] = -20
- L[:len(L):2] = ["hi", "bye", -20, 176, 1]
- L[2:5][1] = 100
- L[1:5][:2] = [10, "bye"]

Problem

- Write a program that rolls two n-sided dice a million times and records the number of times 2, 3, …, 2n show up as the sum of the two dice rolls.