THE UNIVERSITY OF WESTERN AUSTRALIA

Lecture 9
Strings and Lists are Sequences

Objectives

- To understand the list data type and how strings and lists are subclasses of sequences
- To understand the differences between mutable and immutable sequences.
- To get familiar with various operations that can be performed on lists through built-in functions.

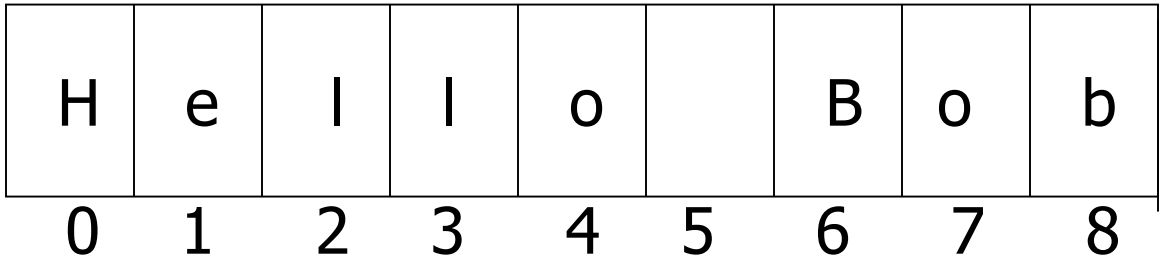
Revision: The String Data Type

- The most common use of personal computers is word processing.
- Text is represented in programs by the string data type.
- A string is a sequence of characters enclosed within quotation marks (") or apostrophes (').

Revision: Indexing

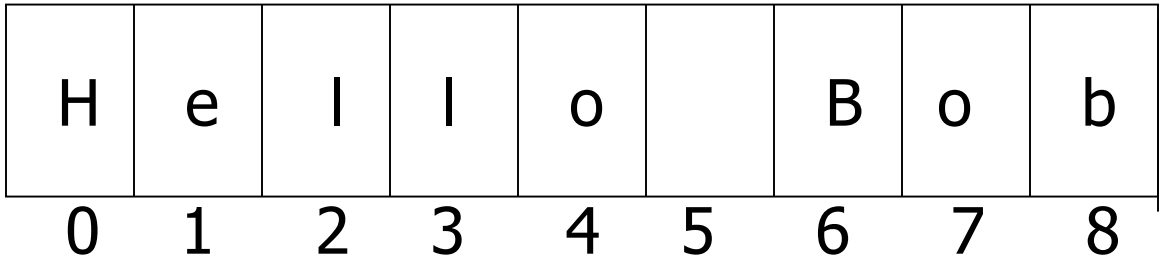
- We can access the individual characters in a string through indexing.
- The positions in a string are numbered from the left, starting with 0.
- The general form is <string>[<expr>] where the value of expr (i.e. an integer) determines which character is selected from the string.

Revision: Indexing




```
>>> greet = "Hello Bob"
>>> greet[0]
'H'
>>> print(greet[0], greet[2], greet[4])
H l o
>>> x = 8
>>> print(greet[x - 2])
B
```

Revision: Slicing




```
>>> greet[0:3]
'Hel'
>>> greet[5:9]
' Bob'
>>> greet[:5]
'Hello'
>>> greet[5:]
' Bob'
>>> greet[:]
'Hello Bob'
```

This is same as `greet`

Int to Month

- Converting an `int` that stands for a month into the three letter abbreviation for that month.

- Store all the names in one big string:

```
months= "JanFebMarAprMayJunJulAugSepOctNovDec"
```

- Use the month number as an index for slicing this string:

```
pos *= 3
monthAbbrev = months[pos:pos+3]
```

Still not right

Int to Month

```
# month.py
#  A program to print the abbreviation of a month, given its number

def main():
    # months is used as a lookup table
    months = "JanFebMarAprMayJunJulAugSepOctNovDec”
    n = int(input("Enter a month number (1-12): "))

    # compute starting position of month n in months
    pos = (n-1) * 3

    # Grab the appropriate slice from months
    monthAbbrev = months[pos:pos+3]

    # print the result
    print ("The month abbreviation is", monthAbbrev + ".")
```

Int to Month

```
>>> main()
Enter a month number (1-12): 1
The month abbreviation is Jan.
>>> main()
Enter a month number (1-12): 12
The month abbreviation is Dec.
```

- One weakness – this method only works where the potential outputs all have the same length.
- How could you handle spelling out the names of the months?

Lists as Sequences

- Strings are always sequences of characters, but lists can be sequences of arbitrary values.
- Lists can have numbers, strings, or both!

```
myList = [1, "Spam ", 3.142, True]
```

Lists as Sequences

- We can use the idea of a list to make our previous month program even simpler!
- We change the lookup table for months to a list:

```
months = ["Jan", "Feb", "Mar", "Apr",
          "May", "Jun", "Jul", "Aug",
          "Sep", "Oct", "Nov", "Dec"]
```

- Note that the months line overlaps a line. Python knows that the expression isn't complete until the closing] is encountered.

Lists as Sequences

- To get the months out of the sequence, do this:

```
monthAbbrev = months[n-1]
```

  Rather than this:

```
monthAbbrev = months[pos:pos+3]
```

Lists as Sequences

```
# month2.py
#  A program to print the month name, given it's number.
#  This version uses a list as a lookup table.

def main():

    # months is a list used as a lookup table
    months = ["Jan", "Feb", "Mar", "Apr", "May", "Jun",
              "Jul", "Aug", "Sep", "Oct", "Nov", "Dec”]
    n = int(input("Enter a month number (1-12): "))
    print ("The month abbreviation is", months[n-1] + ".”).
```

Note: Since the list is indexed starting from 0, the *n-1* calculation is straight-forward enough to put in the print statement without needing a separate step.

Lists as Sequences

- This version of the program is easy to extend to print out the whole month name rather than an abbreviation

```
months = ["January", "February", "March",
          "April", "May", "June", "July",
          "August", "September", "October",
          "November", "December"]
```

Lists as Sequences

- It turns out that strings are really a special kind of sequence, so these operations also apply to other sequences, particularly lists.

```
>>> [1,2] + [3,4]  # + is concatenate, or vector addition!
[1, 2, 3, 4]
>>> [1,2]*3     # This is NOT scalar multiplication!
[1, 2, 1, 2, 1, 2]
>>> grades = ['HD', 'D', 'Cr', 'P', 'N']
>>> grades[0]
'HD'
>>> grades[2:4]
['Cr', 'P']
>>> len(grades)
5
```

Lists as Sequences

- Lists are mutable, meaning they can be changed. Strings can **not** be changed.

```
>>> myList = [34, 26, 15, 10]
>>> myList[2]
15
>>> myList[2] = 0
>>> myList
[34, 26, 0, 10]
>>> myString = "Hello World"
>>> myString[2]
'l'
>>> myString[2] = "p”
Traceback (most recent call last):
  File "<pyshell#16>", line 1, in -toplevel-
    myString[2] = "p"
TypeError: object doesn't support item assignment
```

Lists Have Methods, Too

- Back at the decoder program

```
# Loop through each substring and build Unicode message
    message = ""
    for numStr in inString.split(',’):
        codeNum = int(numStr)     # convert the (sub)string to a number
        # append character to message
        message = message + chr(codeNum)
```

- Each iteration a copy of the message so far is created and another character tacked onto the end. *New string*
- As we build up the message, we keep recopying a longer and longer string just to add a single character at the end!

Lists Have Methods, Too

- We can avoid this recopying by creating a list of characters and then using `append()`
  - *each new character is added to the end of the existing list.*
- Since lists are mutable, the list is changed "in place" without having to copy the content over to a new object.
- When done, we can use `join()` to concatenate the characters into a string.

Lists Have Methods, Too

```
# numbers2text2.py
#     A program to convert a sequence of Unicode numbers into
#         a string of text. Efficient version using a list accumulator.

def main():
    print("This program converts a sequence of Unicode numbers into")
    print("the string of text that it represents.\n")
    # Get the message to encode
    inString = input("Please enter the Unicode-encoded message: ")
    # Loop through each substring and build Unicode message
    chars = []
    for numStr in inString.split():
        codeNum = int(numStr)              # convert digits to a number
        chars.append(chr(codeNum))         # accumulate new character

    message = "".join(chars)      # join with empty string separator
    print("\nThe decoded message is:", message)
```

Lists Have Methods, Too

- List specific functions
  - `min()` # Watch out for mixed types
  - `max()`
  - `list()` – convert sequence into a list
  - `append()` – Add in place to list
  - `reverse()` – Reverse in place a list
- Strings and lists share sequence functions
  - `len()`
  - `+` (concatenation)
  - Slicing using `[:]`
  - `in` (but just letters in strings)

Summary

- We have learned that strings and lists are just different sorts of sequences
- Lists have fewer restrictions than strings
- Many (not all) of the operations that work on strings also work on lists. There are also list specific functions.