Strings, Lists, and Sequences

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

```python
>>> [1,2] + [3,4]
[1, 2, 3, 4]
>>> [1,2]*3
[1, 2, 1, 2, 1, 2]
>>> grades = ['A', 'B', 'C', 'D', 'F']
>>> grades[0]
'A'
>>> grades[2:4]
['C', 'D']
>>> len(grades)
5
```
Strings, Lists, and 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 ", 4, "U"]
Strings, Lists, and 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:

```python
```
Strings, Lists, and Sequences

- To get the months out of the sequence, do this:
  
  ```python
  monthAbbrev = months[n-1]
  ```

  Rather than this:
  
  ```python
  monthAbbrev = months[pos:pos+3]
  ```
# month2.py
#  A program to print the month name, given it's number.
#  This version uses a list as a lookup table.

def main0:

    # months is a list used as a lookup table
    months = ["Jan", "Feb", "Mar", "Apr", "May", "Jun",

    n = input("Enter a month number (1-12): ")

    print "The month abbreviation is", months[n-1] + "."

main0

- Note that the months line overlaps a line.
  Python knows that the expression isn’t complete until the closing ] is encountered.
Strings, Lists, and 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",

    n = input("Enter a month number (1-12): ")

    print "The month abbreviation is", months[n-1] + "."

main()

- 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.
Strings, Lists, and Sequences

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

Strings, Lists, and Sequences

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

```python
>>> 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
```
Strings and Secret Codes

- Inside the computer, strings are represented as sequences of 1’s and 0’s, just like numbers.
- A string is stored as a sequence of binary numbers, one number per character.
- It doesn’t matter what value is assigned as long as it’s done consistently.
Strings and Secret Codes

- In the early days of computers, each manufacturer used their own encoding of numbers for characters.
- Today, American computers use the ASCII system (American Standard Code for Information Interchange).
Strings and Secret Codes

- 0 – 127 are used to represent the characters typically found on American keyboards.
  - 65 – 90 are “A” – “Z”
  - 97 – 122 are “a” – “z”
  - 48 – 57 are “0” – “9”
- The others are punctuation and control codes used to coordinate the sending and receiving of information.
One major problem with ASCII is that it’s American-centric, it doesn’t have many of the symbols necessary for other languages.

Newer systems use Unicode, an alternate standard that includes support for nearly all written languages.
Strings and Secret Codes

- The `ord` function returns the numeric (ordinal) code of a single character.
- The `chr` function converts a numeric code to the corresponding character.

```python
>>> ord("A")
65
>>> ord("a")
97
>>> chr(97)
'a'
>>> chr(65)
'A'
```
Strings and Secret Codes

- Using ord and char we can convert a string into and out of numeric form.
- The encoding algorithm is simple: get the message to encode for each character in the message: print the letter number of the character
- A for loop iterates over a sequence of objects, so the for loop looks like: for ch in <string>
Strings and Secret Codes

# text2numbers.py
# A program to convert a textual message into a sequence of
# numbers, utilizing the underlying ASCII encoding.

def main():
    print "This program converts a textual message into a sequence"
    print "of numbers representing the ASCII encoding of the message."
    print

    # Get the message to encode
    message = raw_input("Please enter the message to encode: ")

    print
    print "Here are the ASCII codes:"

    # Loop through the message and print out the ASCII values
    for ch in message:
        print ord(ch),   # use comma to print all on one line.

    print

main()
We now have a program to convert messages into a type of “code”, but it would be nice to have a program that could decode the message!

The outline for a decoder:

- get the sequence of numbers to decode
- message = ""
- for each number in the input:
  - convert the number to the appropriate character
  - add the character to the end of the message
- print the message
Strings and Secret Codes

- The variable message is an accumulator variable, initially set to the empty string, the string with no characters (""). 
- Each time through the loop, a number from the input is converted to the appropriate character and appended to the end of the accumulator.
Strings and Secret Codes

- How do we get the sequence of numbers to decode?
- Read the input as a single string, then split it apart into substrings, each of which represents one number.
Strings and Secret Codes

- **The new algorithm**
  
  get the sequence of numbers as a string, `inString`
  
  `message = ""`
  
  for each of the smaller strings:
    
    change the string of digits into the number it represents
    
    append the ASCII character for that number to `message`
  
  print `message`

- **Just like there is a math library, there is a string library with many handy functions.**
One of these functions is called `split`. This function will split a string into substrings based on spaces.

```python
>>> import string
>>> string.split("Hello string library!")
['Hello', 'string', 'library!']
```
Strings and Secret Codes

- Split can be used on characters other than space, by supplying that character as a second parameter.

```python
>>> string.split("32,24,25,57", ",")
['32', '24', '25', '57']

>>> string.split("32,24,25,57", ",")
['32', '24', '25', '57']
```

Python Programming, 1/e 21
Strings and Secret Codes

- How can we convert a string containing digits into a number?
- Python has a function called eval that takes any strings and evaluates it as if it were an expression.

```python
>>> numStr = "500"
>>> eval(numStr)
500
>>> x = eval(raw_input("Enter a number "))
Enter a number 3.14
>>> print x
3.14
>>> type (x)
<type 'float'>
```
Strings and Secret Codes

# numbers2text.py
#     A program to convert a sequence of ASCII numbers into
#     a string of text.

import string  # include string library for the split function.

def main():
    print "This program converts a sequence of ASCII numbers into"
    print "the string of text that it represents."
    print

    # Get the message to encode
    inString = raw_input("Please enter the ASCII-encoded message: ")

    # Loop through each substring and build ASCII message
    message = ""
    for numStr in string.split(inString):
        # convert the (sub)string to a number
        asciiNum = eval(numStr)
        # append character to message
        message = message + chr(asciiNum)

    print "The decoded message is:", message

main()
Strings and Secret Codes

- The split function produces a sequence of strings. numString gets each successive substring.
- Each time through the loop, the next substring is converted to the appropriate ASCII character and appended to the end of message.
Strings and Secret Codes

>>> main()
This program converts a textual message into a sequence
of numbers representing the ASCII encoding of the message.

Please enter the message to encode: CS120 is fun!

Here are the ASCII codes:
67 83 49 50 48 32 105 115 32 102 117 110 33

>>> 
This program converts a sequence of ASCII numbers into
the string of text that it represents.

Please enter the ASCII-encoded message: 67 83 49 50 48 32 105 115 32 102 117 110 33
The decoded message is: CS120 is fun!
Other String Operations

There are a number of other string processing functions available in the string library. Try them all!

- `capitalize(s)` – Copy of `s` with only the first character capitalized
- `capwords(s)` – Copy of `s`; first character of each word capitalized
- `center(s, width)` – Center `s` in a field of given width
Other String Operations

- `count(s, sub)` – Count the number of occurrences of sub in s
- `find(s, sub)` – Find the first position where sub occurs in s
- `join(list)` – Concatenate list of strings into one large string
- `ljust(s, width)` – Like center, but s is left-justified
Other String Operations

- `lower(s)` – Copy of `s` in all lowercase letters
- `lstrip(s)` – Copy of `s` with leading whitespace removed
- `replace(s, oldsub, newsub)` – Replace occurrences of `oldsub` in `s` with `newsub`
- `rfind(s, sub)` – Like `find`, but returns the right-most position
- `rjust(s, width)` – Like `center`, but `s` is right-justified
Other String Operations

- rstrip(s) – Copy of s with trailing whitespace removed
- split(s) – Split s into a list of substrings
- upper(s) – Copy of s; all characters converted to uppercase
Other String Operations

```python
>>> s = "Hello, I came here for an argument"
>>> string.capitalize(s)
'Hello, i came here for an argument'
>>> string.capwords(s)
'Hello, I Came Here For An Argument'
>>> string.lower(s)
'hello, i came here for an argument'
>>> string.upper(s)
'HELLO, I CAME HERE FOR AN ARGUMENT'
>>> string.replace(s, "I", "you")
'Hello, you came here for an argument'
>>> string.center(s, 30)
'Hello, I came here for an argument'
```
Other String Operations

```python
>>> string.center(s, 50)
'    Hello, I came here for an argument    '
>>> string.count(s, 'e')
5
>>> string.find(s, ',')
5
>>> string.join(['Number', 'one,', 'the', 'Larch'])
'Number one, the Larch'
>>> string.join(['Number', 'one,', 'the', 'Larch'], 'foo')
'Numberfooone,foothefooLarch'
```
From Encoding to Encryption

- The process of encoding information for the purpose of keeping it secret or transmitting it privately is called encryption.
- Cryptography is the study of encryption methods.
- Encryption is used when transmitting credit card and other personal information to a web site.
From Encoding to Encryption

- Strings are represented as a sort of encoding problem, where each character in the string is represented as a number that’s stored in the computer.

- The code that is the mapping between character and number is an industry standard, so it’s not “secret”.
From Encoding to Encryption

- The encoding/decoding programs we wrote use a substitution cipher, where each character of the original message, known as the plaintext, is replaced by a corresponding symbol in the cipher alphabet.

- The resulting code is known as the ciphertext.
From Encoding to Encryption

- This type of code is relatively easy to break.
- Each letter is always encoded with the same symbol, so using statistical analysis on the frequency of the letters and trial and error, the original message can be determined.
From Encoding to Encryption

- Modern encryption converts messages into numbers.
- Sophisticated mathematical formulas convert these numbers into new numbers – usually this transformation consists of combining the message with another value called the “key”
From Encoding to Encryption

- To decrypt the message, the receiving end needs an appropriate key so the encoding can be reversed.

- In a private key system the same key is used for encrypting and decrypting messages. Everyone you know would need a copy of this key to communicate with you, but it needs to be kept a secret.
From Encoding to Encryption

- In public key encryption, there are separate keys for encrypting and decrypting the message.
- In public key systems, the encryption key is made publicly available, while the decryption key is kept private.
- Anyone with the public key can send a message, but only the person who holds the private key (decryption key) can decrypt it.