CSC 110: Week 3, Lecture 2

Dr. Bodine

On Tuesday…

Today…

- String methods
- Encoding (and decoding)
- Input/Output string
- File processing

Parsing strings as characters and words

- **Index** string to get characters
  - mystring = “This is a string”
  - mystring[0] = ‘T’
  - mystring[1] = ‘h’

- Can **slice** to get substrings
  - mystring[10:] = ‘string’

- Can **split** to get words
  - mystring.split() produces a list of strings
  - teststring = mystring.split()
  - teststring = [‘This’, ‘is’, ‘a’, ‘test’]
  - Index teststring to get a word

split method

- Can be used to parse long strings into segments

- Example: take a sentence and store each word separately
  - mystring = “This is a string”
  - mylistofwords = mystring.split()

- Example 2: Take a comma delimited list and store each entry
  - mystring2=“Laura,Taylor,Rachel,Brett”
  - mystring2.split(“,”)

Other string methods

- capitalize
- title
- lower
- upper
- replace
- center
- count
- find
- join

- Remember notation:
  - mystring.capitalize()
  - mystring.upper()

- Some accept other parameters

Let's play around with it…

- Write a test sentence and assign mystring the value
  - Example: mystring = "My name is Dr. Bodine."
- What happens you call various methods?
  - mystring.capitalize()
  - mystring.title()
- What happens when you try to split it? Are there multiple ways to do accomplish this?
- When I call one of these methods, does it alter the string?
  - Difference between altering and reassinging

Encoding

- Strings are stored numerically via encoding
  - Associate a character/symbol with a number
  - Example: a= 1, b = 2, c= 3, …, z= 26
- Converting between encoding and character
  - ord(<char>) gives us the number associated with the character
  - chr(<number>) gives us the character associated with the number
- Encryption
  - Protect privacy

Encoding algorithm

- Ask user for message to encode
- Loop over the characters in the string
- Use ord(ch) to get the number
- Print the number

```
>>> message = input("Please enter a message to encode: ")
Please enter a message to encode: My name is Inigo Montoya
>>> for ch in message:
        print(ord(ch))
```

- Prints one number per line. What if we want to print them differently?

Encoding algorith, cont…

- Can tell print not to end the line
  - Add argument: end= “ ”

```
>>> message = input("Please enter a message to encode: ")
Please enter a message to encode: My name is Inigo Montoya
>>> for ch in message:
        print(ord(ch), end = " ")


77 121 32 110 97 109 101 32 105 115 32 73 110 105 103 111 32 77 111 110 116 111
121 97
```

Decoding

- What about the reverse?
- Given a sequence of numbers, want to write the encoded message
- We have an algorithm for doing this as well

1: get the message as input

```
>>> inString= input("Please enter the Unicode-encoded message: ")
Please enter the Unicode-encoded message: 89 111 117 32 107 105 108 108 101 100
32 109 121 32 102 97 116 104 101 114 46 32 32 80 114 101 112 97 114 101 32 116 1
11 32 100 105 101 46
>>> inString
'89 111 117 32 107 105 108 108 101 100 32 109 121 32 102 97 116 104 101 114 46 3
2 32 80 114 101 112 97 114 101 32 116 111 32 100 105 101 46 '
```

2: assign a placeholder message

```
>>> inString= input("Please enter the Unicode-encoded message: ")
Please enter the Unicode-encoded message: 89 111 117 32 107 105 108 108 101 100
32 109 121 32 102 97 116 104 101 114 46 32 32 80 114 101 112 97 114 101 32 116 1
11 32 100 105 101 46
>>> inString
'89 111 117 32 107 105 108 108 101 100 32 109 121 32 102 97 116 104 101 114 46 3
2 32 80 114 101 112 97 114 101 32 116 111 32 100 105 101 46 '
>>> message = ""
```

3: loop over the entries…

```
>>> inString= input("Please enter the Unicode-encoded message: ")
Please enter the Unicode-encoded message: 89 111 117 32 107 105 108 108 101 100
32 109 121 32 102 97 116 104 101 114 46 32 32 80 114 101 112 97 114 101 32 116 1
11 32 100 105 101 46
>>> inString
'89 111 117 32 107 105 108 108 101 100 32 109 121 32 102 97 116 104 101 114 46 3
2 32 80 114 101 112 97 114 101 32 116 111 32 100 105 101 46 '
>>> message = ""
>>> for numStr in inString.split():
```

4: evaluate the entries!

```
>>> inString= input("Please enter the Unicode-encoded message: ")
Please enter the Unicode-encoded message: 89 111 117 32 107 105 108 108 101 100
32 109 121 32 102 97 116 104 101 114 46 32 32 80 114 101 112 97 114 101 32 116 1
11 32 100 105 101 46
>>> inString
'89 111 117 32 107 105 108 108 101 100 32 109 121 32 102 97 116 104 101 114 46 3
2 32 80 114 101 112 97 114 101 32 116 111 32 100 105 101 46 '
>>> message = ""
>>> for numStr in inString.split():
        codeNum = eval(numStr)
```

5: add the characters to the message

```
>>> inString= input("Please enter the Unicode-encoded message: ")
Please enter the Unicode-encoded message: 89 111 117 32 107 105 108 108 101 100
32 109 121 32 102 97 116 104 101 114 46 32 32 80 114 101 112 97 114 101 32 116 1
11 32 100 105 101 46
>>> inString
'89 111 117 32 107 105 108 108 101 100 32 109 121 32 102 97 116 104 101 114 46 3
2 32 80 114 101 112 97 114 101 32 116 111 32 100 105 101 46 '
>>> message = ""
>>> for numStr in inString.split():
        codeNum = eval(numStr)
        message = message + chr(codeNum)
```

6: print the message

```
>>> inString= input("Please enter the Unicode-encoded message: ")
Please enter the Unicode-encoded message: 89 111 117 32 107 105 108 108 101 100
32 109 121 32 102 97 116 104 101 114 46 32 32 80 114 101 112 97 114 101 32 116 1
11 32 100 105 101 46
>>> inString
'89 111 117 32 107 105 108 108 101 100 32 109 121 32 102 97 116 104 101 114 46 3
2 32 80 114 101 112 97 114 101 32 116 111 32 100 105 101 46 '
>>> message = ""
>>> for numStr in inString.split():
        codeNum = eval(numStr)
        message = message + chr(codeNum)


>>> message
'You killed my father.  Prepare to die.'
```

Think-pair-share

- What was different about the encoding and decoding process? Why?
- What if we wanted to use a different encoding? How could we go about writing that?

Input/Output String Manipulation

- Formatting is necessary for input and output
- Splitting sentences into words
- Removing newline characters
  - Often a newline character appears at the end of statements, like with print
- May need to cast strings to numeric types
  - Example: when we want to interpret “2” as a the number 2 and do arithmetic
  - int(“1”), float(“1”), float(“1.1”) but not int(“1.1”), not int(“01”)
- May need to cast numeric types to strings
  - str(42) gives us ‘42’

File Processing

- Different in different languages but basic idea is same

- Opening the file
  - Associates files with objects in the program
  - Doesn't actually read the information in the file!

- Manipulating the information from the file
  - Read the data, for example so we can assign variables values from the file
  - Write new data to the file, for example output of our computations

- Closing the file
  - Bookkeeping between the objects in program and file
  - Be careful about how file writing and closing relate

Opening files in python

- <variable> = open(<name>, <mode>)
  - Associates the file on disk with the named variable
  - Encapsulate the filename and mode
  - Modes: read only (“r”), write (“w”), append (“a”), read and write (“r+”)

- Example: infile = open(“testdata.dat”, “r”)
  - Associates the file named names.dat with the variable labeled infile
  - Use the variable infile to access data in the file

- Now you have access to the data in the file
  - List of lines

Reading files in python

- <file>.read()
  - Single (possibly multi-line) string containing remaining contents of file
- <file>.readline()
  - String containing the next line of the file, including the newline character
- <file>.readlines()
  - List of remaining lines in the file. Each list item is a single line including the newline character

Challenge problem

- Write a batch-oriented program that reads a set of first and last names from a file and prints “Hello, <first name> <last name>. Your initials are <First initial> <Second initial>.”
  - Input file will look like

    Laura Bodine
    Inigo Montoya
    Princess Buttercup
    Miracle Max

- Have the same input file but save the results to a new file called welcome.dat

Start by opening the file…

- Start with our file named “names.dat”
- Open the file in python, associating it with the variable infile
  - infile = open(“names.dat”, “r”)

- Loop over the lines and print them to the screen