Vahé Karamian

Python Programming

CS-110

CHAPTER 5

Sequences: Strings, Lists, and Files

OBJECTIVES

- To understand the string data type and how strings are represented in the computer.
- To become familiar with various operations that can be performed on strings through built-in functions and string methods.
- To understand the basic idea of sequences and indexing as they apply to Python strings and lists.
- To be able to apply string formatting to produce attractive, informative program output.

OBJECTIVES

- To be able to apply string formatting to produce attractive, informative program output.
- To understand basic file-processing concepts and techniques for reading and writing text files in Python.
- To understand basic concepts of cryptography.
- To be able to understand and write programs that process textual information.

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 (‘)

THE STRING DATA TYPE

```
>>>str1 = “Hello”
>>>str2 = ‘spam’
>>>print(str1,str2)
Hello spam
>>>type(str1)
<type ‘str’>
>>>type(str2)
<type ‘str’>
```

THE STRING DATA TYPE

- >>>firstName = input("Enter first name:")
- The input statement is a delayed expression.
- When you enter a name, it's doing the same thing as: firstName = <value>
- We can access the individual characters in a string through indexing.
- The positions in a string are numbered from left, starting with 0.
- The general form is <string>[<expr>], where the value of expression determines which character is selected from the string.

THE STRING DATA TYPE

```
>>>greet = “Hello Bob”
>>>greet[0]
‘H’
>>>print(greet[0], greet[2], greet[4])
H l o
>>>x = 8
>>>print(greet[x-2])
B
```

THE STRING DATA TYPE

- In a string of n characters, the last character is at position n-1 since we start counting with 0.
- We can index from the right side using negative indexes.

```
>>>greet[-1]
'b'
>>>greet[-3]
'B'
```

THE STRING DATA TYPE

- Indexing returns a string containing a single character from a larger string.
- We can also access a contiguous sequence of characters, called a substring, through a process called slicing.
- Slicing: <string>[<start>:<end>]
- <start> and <end> should both be integers.
- The slice contains the substring beginning at position <start> and runs up **but does not include** the position <end>.

THE STRING DATA TYPE

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

THE STRING DATA TYPE

- If either expression is missing, then the start or the end of the string are used.
- Can we put two strings together into a longer string?
- Yes! *Concatenation*, “glues” two strings together using the (+) operator.
- *Repetition* builds up a string by multiple concatenation of a string with itself (*).
- The function *len* will return the length of a string.

THE STRING DATA TYPE

```
>>>”spam” + ”eggs”
‘spameggs’
>>>”spam” + “and” + “eggs”
‘spamandeggs’
>>>3 * “spam”
‘spamspamspam’
>>>”spam” * 5
‘spamspamspamspamspam’
>>>(3*”spam”)+(2*”eggs”)
‘spamspamspameggseggs’
```

THE STRING DATA TYPE

```
>>>len(“spam”)
4
>>>for ch in “Spam!”:
        print(ch, end=‘ ‘)
```

- S p a m !

THE STRING DATA TYPE

Operator	Meaning
+	Concatenation
*	Repetition
<string>[]	Indexing
<string>[:]	Slicing
len(<string>	Length
for <var> in <string>	Iteration through characters

SIMPLE STRING PROCESSING

- Usernames on a computer system
  - First Initial, first seven characters of last name

```
#get user's first and last name
>>>first = input("First name: ")
>>>last = input("Last name: ")
>>>username = first[0] + last[:7]
```

SIMPLE STRING PROCESSING

- Another example: converting an int that stands for the month into the three letter abbreviation for that month.
- Store all the names in one big string:

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

  monthAbbrev = month[pos:pos+3]

SIMPLE STRING PROCESSING

MONTH	NUMBER	POSITION
JAN	1	0
FEB	2	3
MAR	3	6
APR	4	9
…	…	…

To get the correct position, subtract one from the month number and multiply by three

SIMPLE STRING PROCESSING

```
Def main():
        month = "JANFEBMARAPRMAYJUNJULAUGSEPOCTNOVDEC"

        n = eval(input("Enter a month number: "))

        pos = (n-1)*3

        monthAbbrev = month[pos:pos+3]

        print("The month abbreviation is ", monthAbbrev)
```

SIMPLE STRING PROCESSING

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

STRINGS, LISTS, & SEQUENCES

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

```
>>>[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, & 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, & SEQUENCES

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

  months = [“JAN”, “FEB”, “MAR”, …, “DEC”]

- To get the month out of the sequence, we do:

  monthAbbrev = months[n-1]

STRINGS, LISTS, & SEQUENCES

```
def main():
        months = ["JAN","FEB","MAR","APR","MAY","JUN","JUL","AUG","SEP","OCT","NOV","DEC"]

        m = eval(input("Enter month number (1-12): "))

        print("The month abbreviation is: ", months[m-1])
```

- Since the list is indexed starting from 0, the n-1 calculation is straight-forward enough to put it in the print statement.
- This version of the program is easy to extend to print out the whole month name.

STRINGS, LISTS, & 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
[35, 26, 0, 10]
>>>myString = "Hello World"
>>>myString[2]
l
>>>myString[2] = "p"
You will get an error!

STRINGS, LISTS, & SEQUENCES

- Inside the computer, strings are represented as sequence 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.
- In the early days of computers, each manufacturer used their own encoding of numbers for characters.
- Today, American computers use the *ASCII (American Standard Code for Information Interchange)* system.

STRINGS & SECRET CODES

- 0 – 127 are used to represent the characters typically found on American keyboards:
  - 65 – 90 are: “A” to “Z”
  - 97 – 122 are: “a” to “z”
  - 48 – 57 are: “0” to “9”
- The rest are punctuation and control codes used to coordinate the sending and receiving of information.

STRINGS & SECRET CODES

- 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 & 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.

```
>>>ord("A")
65
>>>chr(65)
'A'
```

STRINGS & SECRET CODES

- How would we write a simple encoding program?
- Using *ord* and *chr* 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 & SECRET CODES

- How would we write a simple encoding program?
- Using *ord* and *chr* 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 & SECRET CODES

CODE

```
>>> def main():
    message = input("Please enter the message to encode: ")
    print("Here are the ASCII codes:")
    for ch in message:
        print(ord(ch), end=' ')

    print()
```

```
Please enter the message to encode: Vahe Karamian
Here are the ASCII codes:
86 97 104 101 32 75 97 114 97 109 105 97 110
```

STRINGS & SECRET CODES

- Ok, so now we have a program to convert messages into a type of “code”.
- But how can we decode what we just encoded?
- We need to write 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 & 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.
- 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 & SECRET CODES

- Our new algorithm:

  Get the sequence of numbers as a string, inString

  Message = “”

  For each of the smaller strings:

    Change the string to digits into the number it represents

    Append the ASCII character for that number to message

  Print message

- How are we going to split a string?

STRINGS & SECRET CODES

- Just like there is a math library, there is also 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.

```
>>> str.split("Hello There Buddy!")
['Hello', 'There', 'Buddy!']
```

STRINGS & SECRET CODES

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

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

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

STRINGS & SECRET CODES

```
>>> numStr = "500"
>>> eval(numStr)
500
>>> x = eval(input("Enter a number: "))
Enter a number: 3.33
>>> print(x)
3.33
>>> type(x)
<class 'float'>
```

STRINGS & SECRET CODES

CODE

```
>>> def main():
    inString = input("Enter ASCII encoded message: ")
    message = ""
    for numStr in str.split(inString):
        asciiNum = eval(numStr)
        message = message + chr(asciiNum)

    print("The decoded message is: ", message)
```

```
>>> main()
Enter ASCII encoded message: 86 97 104 101 32 75 97 114 97 109 105 97 110
The decoded message is:  Vahe Karamian
```

STRINGS & SECRET CODES

- The split function produces a sequence of strings. *numStr* 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 the message
- There are a number of other string processing functions available in the string library. Let's take a look.

FUNCTION	DESCRIPTION
s.capitalize()	Copy of s with only the first character capitalized.
s.center(width)	Copy of s centered in field of given width.
s.count(sub)	Count the number of occurrences of sub in s.
s.find(sub)	Find the first position where sub occurs in s.
s.join(list)	Concatenate list into a string, using s as separator.
s.ljust(width)	Like center, but s is left-justified.
s.lower()	Copy of s in all lowercase characters.
s.lstrip()	Copy of s with leading whitespace removed.
s.replace(oldsub, newsub)	Replace all occurrences of oldsub in s with newsub.
s.rfind(sub)	Like find, but return the rightmost position.
s.rjust(width)	Like center, but s is right-justified.
s.rstrip()	Copy of s with trailing white spaces removed.
s.split()	Split s into a list of substrings.
s.title()	Copy of s with first character of each word capitalized.
s.upper()	Copy of s with all characters converted to uppercase.

OTHER STRING OPERATIONS

```
>>> s = "i came here to study computer science"
>>> s.capitalize()
'I came here to study computer science'
>>> s.center(50)
'     i came here to study computer science      '
>>> s.count(c)
Traceback (most recent call last):
  File "<pyshell#38>", line 1, in <module>
    s.count(c)
NameError: name 'c' is not defined
>>> s.count("c")
4
>>> s.find("computer")
21
>>> s.ljust(50)
'i came here to study computer science          '
```

```
>>> s.lower()
'i came here to study computer science'
>>> s.lstrip()
'i came here to study computer science'
>>> s.replace("came", "am")
'i am here to study computer science'
>>> s.rfind("computer")
21
>>> s.rjust(50)
'          i came here to study computer science'
>>> s.rstrip()
'i came here to study computer science'
>>> s.split()
['i', 'came', 'here', 'to', 'study', 'computer', 'science']
>>> s.title()
'I Came Here To Study Computer Science'
>>> s.upper()
'I CAME HERE TO STUDY COMPUTER SCIENCE'
```

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 is 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 program 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 *cipher text*.
- 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

- Of course modern encryptions are much more complex.
- Sophisticated mathematical formulas are used to 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 in the circle would need a copy of this key to communicate with you.

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!

INPUT/OUTPUT AS STRING MANIPULATION

- Often we will need to do some string operations to prepare our string data for output. (make it look nice and neat)
- Let's say we want to enter a date in the format "05/24/2011" and output "May 24, 2011."
- How could we do that?

INPUT/OUTPUT AS STRING MANIPULATION

- Input the date in mm/dd/yyyy format (strDate)
- Split strDate into month, day and year strings.
- Convert the month string into a month number.
- Use the month number to lookup the month name.
- Create a new date string in the form “Month Day, Year”.
- Output the new date string.

INPUT/OUTPUT AS STRING MANIPULATION

- Input the date in mm/dd/yyyy format (strDate)
- Split strDate into month, day and year strings.
- Convert the month string into a month number.
- Use the month number to lookup the month name.
- Create a new date string in the form “Month Day, Year”.
- Output the new date string.

INPUT/OUTPUT AS STRING MANIPULATION

- The first two lines are easily implemented:
  - strDate = input("Enter a date (mm/dd/yyyy):")
  - monthStr, dayStr, yearStr = str.split(strDate, "/")
- The date is input as a string, and then "unpacked" into the three variables by splitting it at the slashes using simultanious assignment!
- Next step: convert monthStr into a number.

INPUT/OUTPUT AS STRING MANIPULATION

- We can use the eval function on monthStr to convert “05”, for example, into the integer 5.
  - eval(“05”) = 5
- Another conversion method/technique would be to use the int function:
  - int(“05”) = 5
- There is only one thing you should be very careful with!

INPUT/OUTPUT AS STRING MANIPULATION

- And that is the leading zero!

>>>int(“05”)

5

>>>eval(“05”)

Invalid token

- Will throw an exception!
- In general better to use int() to convert string into integer than eval.

INPUT/OUTPUT AS STRING MANIPULATION

```
months = [“January”, “February”,…, “December”]
monthStr = months[int(monthStr)-1]
```

- Remember that since we start counting at 0, we need to subtract one from the month.
- Now let’s concentrate on the output string.

```
print(“The converted date is:”, monthStr, dayStr + ”,”, yearStr)
```

- Notice how the comma is appended to dayStr with concatenation!

INPUT/OUTPUT AS STRING MANIPULATION

- Sometimes we want to convert a number into a string.
- We can use the str() function for converting numbers to strings!

```
>>>str(500)
'500'
>>>value = 3.14
>>>str(value)
'3.14'
>>>print("The value is ", str(value), ".")
The value is 3.14.
```

INPUT/OUTPUT AS STRING MANIPULATION

- Make sure you understand the following:
- String + String is a string operation.
- Integer + Integer is a mathematical operation.
- You can not have the following:
  - String + Integer, this is not possible
- You can do the following however:
  - String + str(Integer), this is possible, why?

INPUT/OUTPUT AS STRING MANIPULATION

Function	Meaning
float(<expr>)	Convert expression into floating point value.
int(<expr>)	Convert expression into integer value.
long(<expr>)	Convert expression into long integer value.
str(<expr>)	Return a string representation of expression.
eval(<string>)	Evaluate string as an expression.

STRING FORMATTING

- String formatting is an easy way to get beautiful output!
- We can format our output by modifying the print statement to print out as we desire.

print(“The total value of your change is ${0:0.2f} “.format(total))

- String formatting takes the form
- <template-string>.format(<values>)

STRING FORMATTING

print(“Hello {0} {1}, you may have won ${2}”.format(“Vahe”, “Karamian”, 10000))

“This in, {0:5}, was placed in a field of width 5”.format(7)

“This in, {0:10}, was placed in a field of width 10”.format(7)

“This in, {0:10.5}, was placed in a field of width 10 and precision 5”.format(3.1415926)

“This in, {0:10.5f}, is fixed at 5 decimal places”.format(3.1415926)

“This in, {0:0.5}, has width 0 and precision 5”.format(3.1415926)

BETTER CHANGE COUNTER

- With what we know about floating point numbers, we might be uneasy to use them for money situations.
- Now with our new knowledge of string formatting, we can rest assured that we can format our output in any way we like.
- Let take a look and see how we can trace money in cents using an int or long int, and convert it into dollars and cents when output.

BETTER CHANGE COUNTER

- If total is the value in cents (an integer)
  - Dollars = total // 100
  - Cents = total % 100
- Let us take a look at the change calculation program.

BETTER CHANGE COUNTER

```
#change calculation
def main():
    print("Change counter\n")

    print("Please enter the count of each coin type.")
    quarters = eval(input("Quarters: "))
    dimes = eval(input("Dimes: "))
    nickels = eval(input("Nickels: "))
    pennies = eval(input("Pennies: "))

    total = 25 * quarters + 10 * dimes + 5 * nickels + pennies

    print("The total value of your change is ${0}.{1:0>2}".format(total//100,
total%100))
```

MULTI-LINE STRINGS

- A file is a sequence of data that is stored in secondary memory (disk drive).
- Files can contain any data type, but the easiest to work with are text.
- A file usually contains more than one line of text
- Lines of text are separated with a special character, the newline character.

MULTI-LINE STRINGS

- You can think of the newline as the character produced when you press the <Enter> key.
- In Python, this character is represented as '\n', just as tab is represented as '\t'

Hello
World

Goodbye 32

- In a file it is stored as:
  - Hello\nWorld\n\nGoodbye 32\n

FILE PROCESSING

- The process of opening a file involves associating a file on disk with a variable.
- We can manipulate the file by manipulating this variable.
  - Read from the file.
  - Write to the file.
- When done with the file, it needs to be closed. Closing the file causes any outstanding operations and other bookkeeping for the file to be completed.
- Not properly closing a file could result in data loss.

FILE PROCESSING

- Reading a File
  - File opened
  - Content of file read into RAM (Main Memory)
  - File closed
  - Changes to the file are made to the copy stored in memory (RAM), not on the disk.

FILE PROCESSING

- Saving a File
  - The original file on the disk is reopened in a mode that will allow writing.
  - File writing operation copy the version of document in RAM to the disk
  - File is closed.

FILE PROCESSING

- Working with text files is easy in Python.
- Associate a file with a variable using the open function.
  - <variable> = open(<name>, <mode>)
  - Infile = open(“numbers.dat”, “r”)
- Now we can use the file called “numbers.dat” for reading.

FILE PROCESSING

File Operations	Description
<file>.read()	Return the entire remaining contents of the file as a single (potentially large, multi-line) string
<file>.readline()	Return the next line of the file. That is all the text up to and including the next newline character.
<file>.readlines()	Returns a list of the remaining lines in the file. Each list item is a single line including the newline character at the end.

FILE PROCESSING

Go ahead and create a text file using your notepad. Enter something in the file such as:

Hello World!

I am coming from a file!

Can you guess when I was saved?

Save the file under your c:\myData.txt

FILE PROCESSING – OPEN FILE

```
def main():
    fileName = input("Enter filename: ")
    infile = open(fileName, "r")
    data = infile.read()
    print(data)
```

This code reads the whole documents (file)

FILE PROCESSING – OPEN FILE

```
def main():
    fileName = input("Enter filename: ")
    infile = open(fileName, "r")
    for i in range(5):
        line = infile.readline()
        print(line[:-1])
```

This code reads the documents one line at a time.

FILE PROCESSING – OPEN FILE

```
def main():
    fileName = input("Enter filename: ")
    infile = open(fileName, "r")
    for line in infile.readlines():
            print(line[:-1])
```

Another way of reading the file in Python.

FILE PROCESSING – OPEN FILE

```
def main():
    fileName = input("Enter filename: ")
    infile = open(fileName, "r")
    for line in infile:
            print(line[:-1])
     infile.close()
```

Another way of reading the file in Python.

FILE PROCESSING – WRITE FILE

- Opening a file for writing prepares that file to receive data.
- If no file with the given name exists, a new file will be created.

outputFile = open(“myData.txt”, “w”)

- The easiest way to write information to a file is to use the print function.

print(…., file=<outputFile>)

NOTE: A word of caution, if a file already exists, Python will delete the existing file, and create a new, empty file.

EXAMPLE PROGRAM: BATCH USERNAME

```
def main():
    print("This program creates a file of usernames from a")
    print("file of names.")

    #get the file names
    inputFileName = input("What file are the names in? ")
    outputFileName = input("What file should the usernames go in? ")

    #open the files
    inFile = open(inputFileName, "r")
    outFile = open(outputFileName, "w")

    #process each line of the input file
    for line in inFile:
        #get the first and last names from line
        first, last = line.split()
        #create the username
        username = (first[0]+last[:7]).lower()
        #write it to the output file
        print(username, file=outFile)

    #close both files
    inFile.close()
    outFile.close()

    print("Usernames have been written to ", outputFileName)
```

CHAPTER 5 - SUMMARY

- Strings are sequence of characters. String literals can be delaminated with either single or double quotes.
- Strings and lists can be manipulated with the built-in sequence operations for concatenation (+), repetition (*), indexing ([]), slicing ([:]), and length (len()). A for loop can be used to iterate through the characters of a string, items in a list, or lines of a file.
- One way to converting numeric information into string information is to use a string or a list as a lookup table.
- Lists are more general than strings.
  - Strings are always sequences of characters, whereas lists can contain values of any type.
  - Lists are mutable, which means that items in a list can be modified by assigning new values.

CHAPTER 5 - SUMMARY

- Strings are represented in the computer as numeric codes. ASCII and Unicode are compatible standards that are used for specifying the correspondence between characters and the underlying codes. Python provides the ord and chr functions for translating between Unicode codes and characters.
- Python string and list objects include many useful built-in methods for string and list processing.
- The process of encoding data to keep it private is called encryption. There are two different kinds of encryption systems: private key and public key.

CHAPTER 5 - SUMMARY

- Program input and output often involve string processing. Python provides numerous operators for converting back and forth between numbers and strings. The string formatting method (format) is particularly useful for producing nicely formatted output.
- Text files are multi-line strings stored in secondary memory. A file may be opened for reading or writing. When opened for writing, the existing contents of the file are erased.
- Python provides three file-reading methods: read(), readline(), readlines(). It is also possible to iterate through the lines of a file with a for loop.
- Data is written to a file using the print function. When processing is finished, a file should be closed.