Data Processing and Analysis in Python
Lecture 5
Strings and Text Files

UNIVERSITY OF MARYLAND

ROBERT H. SMITH SCHOOL OF BUSINESS

DR. ADAM LEE

The Structure of Strings

- An integer can't be factored into more primitive parts
- A string is a data structure
- **Data structure**: Consists of smaller pieces of data
- **len()** function returns the string's length
  - the number of characters it contains (0+)

```
>>> len("Hi there!")
9
>>> len("")
0
```

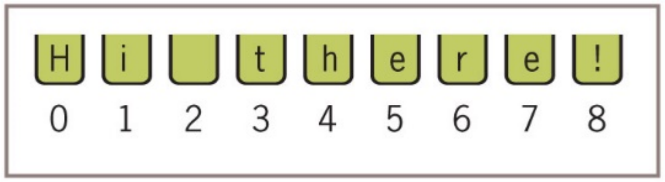


Figure 4-1 Characters and their positions in a string

Subscript Operator

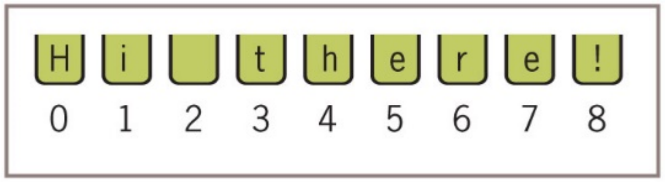


Figure 4-1 Characters and their positions in a string

- <a string>[<an integer expression>]

```
>>> name = "Hi there!"
>>> name[0] # Examine the first character
'H'
>>> name[3] # Examine the fourth character
't'
>>> name[len(name)] # Oops! An index error!
IndexError: string index out of range
>>> name[len(name) - 1] # Examine the last character
'!'
>>> name[-1] # Shorthand for the last character
'!'
>>> name[-2] # Shorthand for next to last character
'e'
```

Example: Count-Controlled Loop

```
>>> data = "Hi there!"
>>> for index in range(len(data)):
    print(index, data[index])
0 H
1 i
2  
3 t
4 h
5 e
6 r
7 e
8 !
```

Slicing for Substrings

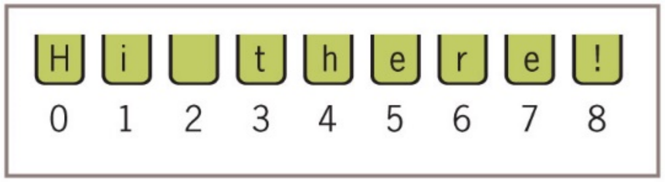


Figure 4-1 Characters and their positions in a string

- **Slicing**: to obtain a substring using subscript operator
  - Place : in the subscript
  - An integer value can appear on either side of the colon

```
>>> name = "Hi there!"
>>> name[0:]
'Hi there!'
>>> name[0:2] # The first two characters
'Hi'
>>> name[:len(name)] # The entire string
'Hi there!'
>>> name[-3:] # The last three characters
're!'
>>> name[3:6] # Drill to extract 'the'
'the'
```

Substring with in Operator

- The left operand of **in** is a target substring
- The right operand is the string to be searched
- Returns True if target string is somewhere in search string, or False otherwise
- Example: traverses a list of filenames and prints just the filenames that have a .txt extension:

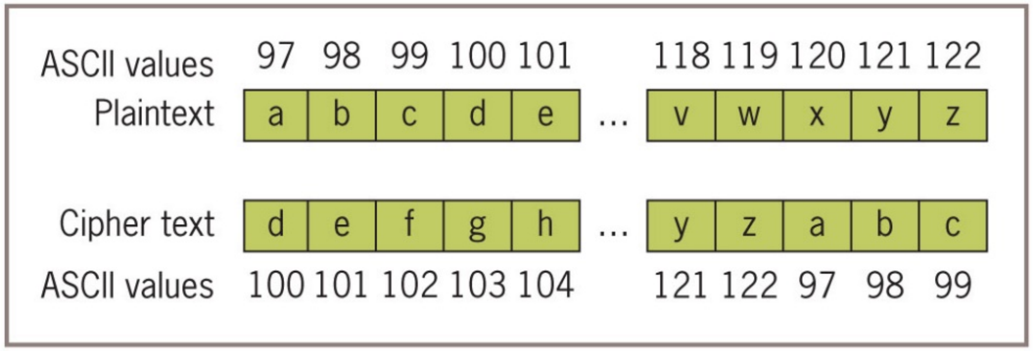
```
>>> fileList = ["myfile.txt", "myprogram.exe",
"yourfile.txt"]
>>> for fileName in fileList:
    if ".txt" in fileName:
        print(fileName)
my file.txt
your file.txt
```

Data Encryption

- It is easy to observe data crossing a network, particularly in wireless networks
  - Attacker may use **sniffing software**
- Data encryption can be used to protect information transmitted on networks
  - Many protocols have secure versions (e.g., HTTPS)
- One or more **keys** are use to **encrypt** messages to produce **cipher text**, and to **decrypt** cipher text back to its original plain text form
  - Examples: Caesar cipher, block cipher

Data Encryption

- **Caesar cipher** replaces each character in plain text with a character a given distance away
  - Example if Caesar cipher equals three characters: the string "invaders" would be encrypted as "lqydghuv"



ASCII values	97	98	99	100	101		118	119	120	121	122
Plaintext	a	b	c	d	e	…	v	w	x	y	z
Cipher text	d	e	f	g	h	…	y	z	a	b	c
ASCII values	100	101	102	103	104		121	122	97	98	99

Figure 4-2 A Caesar cipher with distance +3 for the lowercase alphabet

- To decrypt:
  - Apply a method that uses the same distance value but looks to the left of each character for replacement value

Converting Binary to Decimal

- A positional value is computed by using the ** operator

```
bitString = input("Enter a string of bits: ")
decimal = 0
exponent = len(bitString) - 1
for digit in bitString:
    decimal = decimal + int(digit) * 2 ** exponent
    exponent = exponent - 1
print("The integer value is", decimal)

Enter a string of bits: 1111
The integer value is 15
Enter a string of bits: 1 0 1
The integer value is 5
```

Converting Decimal to Binary

- One algorithm uses division and subtraction (instead of multiplication and addition)
  - Repeatedly divides the decimal number by 2
  - After each division, the remainder (either a 0 or 1) is placed at the beginning of a string of bits
  - Quotient becomes the next dividend in the process
  - Process continues while the decimal number is greater than 0

Converting Decimal to Binary

```
decimal = int(input("Enter a decimal integer: "))
if decimal == 0:
    print(0)
else:
    print("Quotient Remainder Binary")
bitString = " "
while decimal > 0:
    remainder = decimal % 2
    decimal = decimal // 2
    bitString = str(remainder) + bitString
    print("%5d%8d%12s" % (decimal, remainder,
bitString))
print("The binary representation is", bitString)
```

Converting Decimal to Binary

```
Enter a decimal integer: 34
Quotient Remainder Binary
   17        0          0
    8        1         10
    4        0        010
    2        0       0010
    1        0      00010
    0        1     100010
The binary representation is 100010
```

Binary and Hexadecimal Numbers

- To convert from hex to bin, replace each hex digit with the corresponding 4-bit bin number

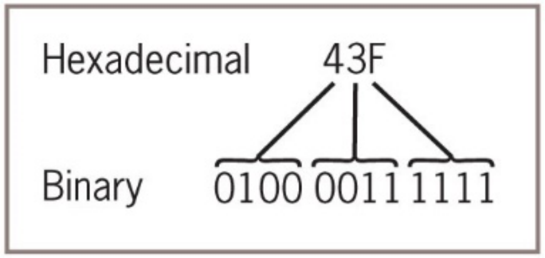


Figure 4-5 The conversion of hexadecimal to binary

- To convert from bin to hex, factor the bits into groups of 4 and look up the corresponding hex digits

String Methods

- A method behaves like a function, but has a slightly different syntax
- A method is always called with a given object **<an object>**.**<method name>**(<argument-1>,..., <argument-*n*>)
- Methods can expect arguments and return values

- **dir**(**str**) to view a complete list and documentation
- **help**(**str**.<method>) to receive documentation

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

String Methods

- Example: extracting a filename's extension

```
>>> "myfile.txt".split('.')
['myfile', 'txt']
>>> "myfile.py".split('.')
['myfile', 'py']
>>> "myfile.html".split('.')
['myfile', 'html']
```

  - Split then subscript [–1] extracts the last element
  - Can be used to write a general expression for obtaining any filename's extension, as follows:

```
>>> filename.split('.')[-1]
```

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

Text Files

- A text file is software object that stores data on permanent medium (e.g. secondary storage)
- When compared to keyboard input from human user, the main advantages of taking input data from a file are:
  - The data set can be much larger
  - The data can be input much more quickly and with less chance of error
  - The data can be used repeatedly with the same program or with different programs

Text Files

- Using a text editor, such as Notepad or TextEdit, you can create, view, and save data in a text file
  - Example: A text file containing six floating-point numbers might look like:

```
34.6 22.33 66.75
77.12 21.44 99.01
```

- All data output to or input from a text file must be strings
  - Number must be converted to string before output
  - Numeric datum from input string must be converted to number

Writing to a File

- Data can be output to a text file using a **file** object
  - To open a file for output:

```
>>> file = open("my file.txt", 'w')
```

  - If file does not exist, it is created
  - If it already exists, any data previously existing are replaced
  - This statement writes two line of text to the file:

```
>>> file.write("First line.\nSecond line.\n")
```

  - This statement writes an integer as text to the file:

```
>>> file.write(str(random.randint(1, 100)) + "\n")
```

- When all outputs are finished, close the file:

```
>>> file.close()
```

File Methods

Method	What it Does
open(filename, mode)	Opens a file given filename and returns a file object. The mode can be 'r', 'w', 'rw', or 'a', means read, write, read/write or append.
file.close()	Closes an output file. Not needed for input files.
file.write(aString)	Outputs aString to a file.
file.read()	Inputs the contents of a file and returns them as a single string. Returns "" if the end of file is reached.
file.readline()	Inputs a line of text and returns it as a string, including the newline. Returns "" if the end of file is reached.

Reading from a File

- You open a file for input in a manner similar to opening a file for output

```
>>> file = open("myfile.txt", 'r')
```

  - If the filename is not accessible from the current working directory, Python raises an error

- There are several ways to read data from a file:
  - Example: the **read()** method

```
>>> text = file.read()
>>> print(text)
First line.
Second line.
64
```

Reading from a File

- After input is finished (i.e. end-of-file), read()/readline() returns an empty string
  - Example: **readline()** reads inputs line by line

```
>>> file = open("myfile.txt", 'r')
>>> while True:
    line = file.readline()
    if line == "":
        break
    print(line)
First line.
Second line.
64
```