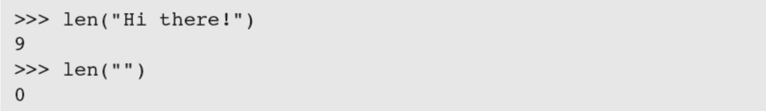
Accessing Characters and Substrings in Strings

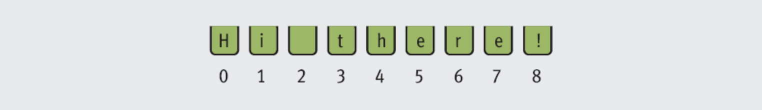
- In this section, we examine the internal structure of a string more closely
- You will learn how to extract portions of a string called **substrings**

The Structure of Strings

- An integer can’t be factored into more primitive parts
- A string is an **immutable data structure**
  - Data structure: Consists of smaller pieces of data
  - String’s length: Number of characters it contains (0+)



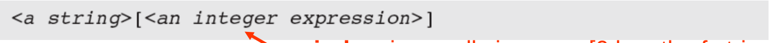
```
>>> len("Hi there!")
9
>>> len("")
0
```



[FIGURE 4.1] Characters and their positions in a string

The Subscript Operator

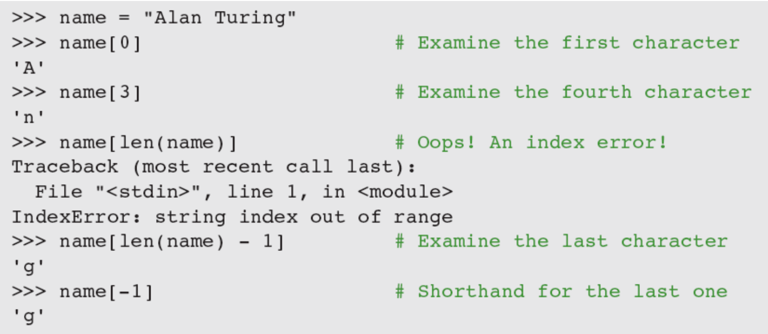
- The form of the **subscript operator** is:



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

index is usually in range [0,length of string – 1]; can be negative

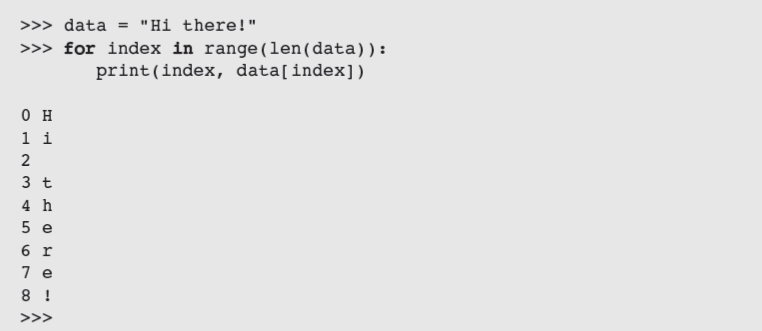
- Examples:



```
>>> name = "Alan Turing"
>>> name[0]                    # Examine the first character
'A'
>>> name[3]                    # Examine the fourth character
'n'
>>> name[len(name)]            # Oops! An index error!
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: string index out of range
>>> name[len(name) - 1]        # Examine the last character
'g'
>>> name[-1]                   # Shorthand for the last one
'g'
```

The Subscript Operator (continued)

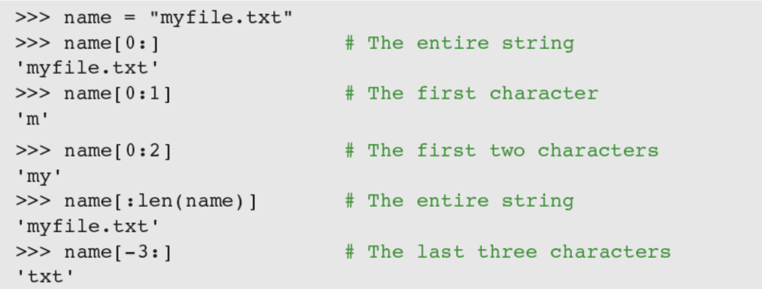
- Subscript operator is useful when you want to use the positions as well as the characters in a string
  - Use a count-controlled loop



```
>>> 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

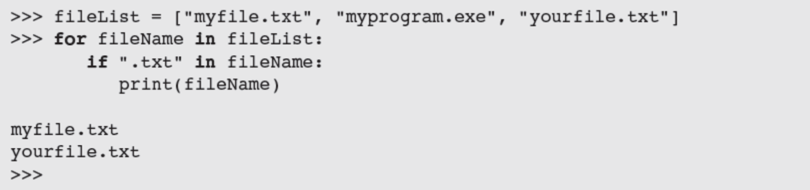
- Python’s subscript operator can be used to obtain a substring through a process called **slicing**
  - Place a colon (**:**) in the subscript; an integer value can appear on either side of the colon



```
>>> name = "myfile.txt"
>>> name[0:]                # The entire string
'myfile.txt'
>>> name[0:1]               # The first character
'm'
>>> name[0:2]               # The first two characters
'my'
>>> name[:len(name)]        # The entire string
'myfile.txt'
>>> name[-3:]               # The last three characters
'txt'
```

Testing for a Substring with the `in` Operator

- When used with strings, the left operand of **`in`** is a target substring and the right operand is the string to be searched
  - Returns **`True`** if target string is somewhere in search string, or **`False`** otherwise



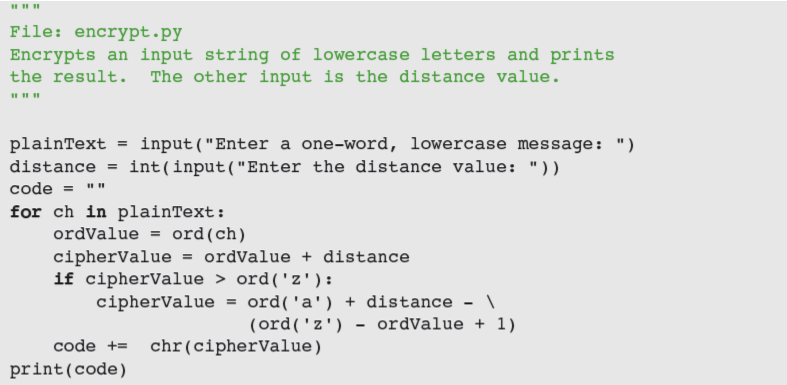
```
>>> fileList = ["myfile.txt", "myprogram.exe", "yourfile.txt"]
>>> for fileName in fileList:
       if ".txt" in fileName:
          print(fileName)

myfile.txt
yourfile.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 (continued)

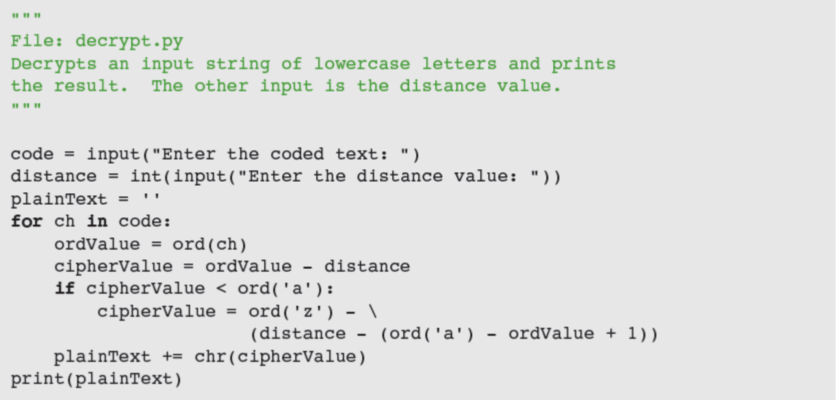
- **Caesar cipher** replaces each character in plain text with a character a given distance away



```python
"""
File: encrypt.py
Encrypts an input string of lowercase letters and prints
the result.  The other input is the distance value.
"""

plainText = input("Enter a one-word, lowercase message: ")
distance = int(input("Enter the distance value: "))
code = ""
for ch in plainText:
    ordValue = ord(ch)
    cipherValue = ordValue + distance
    if cipherValue > ord('z'):
        cipherValue = ord('a') + distance - \
                      (ord('z') - ordValue + 1)
    code +=  chr(cipherValue)
print(code)
```

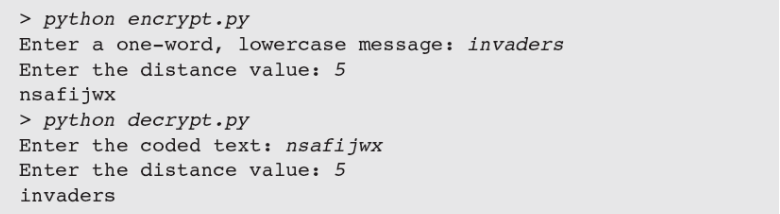
Data Encryption (continued)

- To decrypt, use inverse method



```python
"""
File: decrypt.py
Decrypts an input string of lowercase letters and prints
the result.  The other input is the distance value.
"""

code = input("Enter the coded text: ")
distance = int(input("Enter the distance value: "))
plainText = ''
for ch in code:
    ordValue = ord(ch)
    cipherValue = ordValue - distance
    if cipherValue < ord('a'):
        cipherValue = ord('z') - \
                      (distance - (ord('a') - ordValue + 1))
    plainText += chr(cipherValue)
print(plainText)
```

Data Encryption (continued)



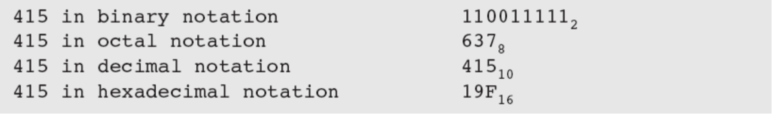
```
> python encrypt.py
Enter a one-word, lowercase message: invaders
Enter the distance value: 5
nsafijwx
> python decrypt.py
Enter the coded text: nsafijwx
Enter the distance value: 5
invaders
```

- Caesar cipher worked well in ancient times, but is easy to break using modern computers

Data Encryption (continued)

- **Block cipher**
  - Uses plaintext character to compute two or more encrypted characters
  - Each encrypted character is computed using two or more plaintext characters
  - Uses an **invertible matrix**

Strings and Number Systems

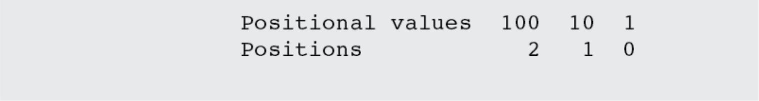


415 in binary notation	110011111_2
415 in octal notation	637_8
415 in decimal notation	415_{10}
415 in hexadecimal notation	$19F_{16}$

- The digits used in each system are counted from 0 to *n* - 1, where *n* is the **system's base**
- To represent digits with values larger than 9_{10}, systems such as base 16 use letters
  - – Example: A_{16} represents the quantity 10_{10}, whereas 10_{16} represents the quantity 16_{10}

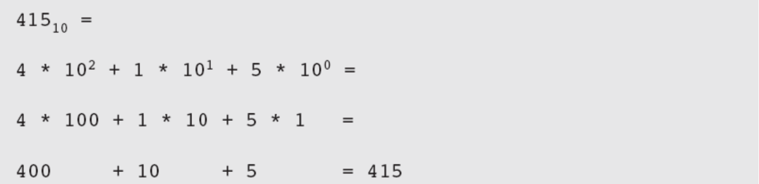
The Positional System for Representing Numbers

- In **positional notation**, a digit has a **positional value**, determined by raising the base to the power specified by the position (*base*^{position})



Positional values	100	10	1
Positions	2	1	0

[FIGURE 4.2] The first three positional values in the base 10 number system



$415_{10} =$

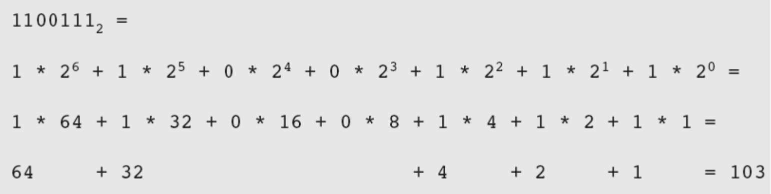
$4 * 10^2 + 1 * 10^1 + 5 * 10^0 =$

$4 * 100 + 1 * 10 + 5 * 1 =$

$400 + 10 + 5 = 415$

Converting Binary to Decimal

- Each digit or bit in binary number has positional value that is power of 2
- We occasionally refer to a binary number as a string of bits or a **bit string**
- To determine the integer quantity that a string of bits represents:



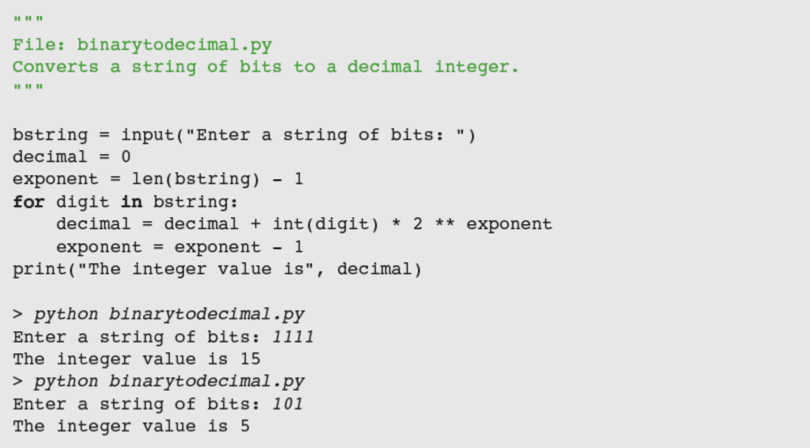
$1100111_2 =$

$1 * 2^6 + 1 * 2^5 + 0 * 2^4 + 0 * 2^3 + 1 * 2^2 + 1 * 2^1 + 1 * 2^0 =$

$1 * 64 + 1 * 32 + 0 * 16 + 0 * 8 + 1 * 4 + 1 * 2 + 1 * 1 =$

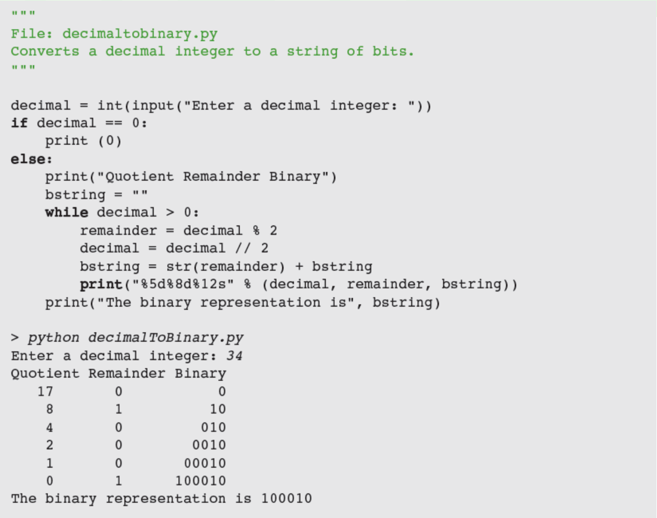
$64 + 32 + 4 + 2 + 1 = 103$

Converting Binary to Decimal (continued)



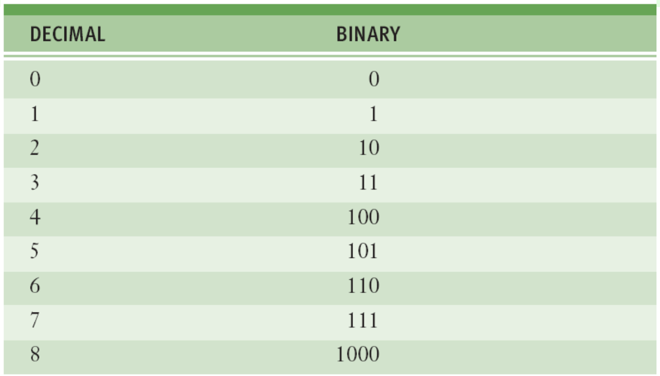
```
"""
File: binarytodecimal.py
Converts a string of bits to a decimal integer.
"""

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

> python binarytodecimal.py
Enter a string of bits: 1111
The integer value is 15
> python binarytodecimal.py
Enter a string of bits: 101
The integer value is 5
```

Converting Binary to Decimal (cont.)



```
"""
File: decimaltobinary.py
Converts a decimal integer to a string of bits.
"""

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

> python decimalToBinary.py
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
```

Conversion Shortcuts



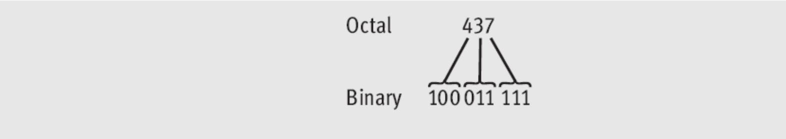
DECIMAL	BINARY
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000

[TABLE 4.1] The numbers 0 through 8 in binary

- Thus, a quick way to compute the decimal value of the number 11111_2 is 2^5 - 1, or 31_{10}

Octal and Hexadecimal Numbers

- To convert from octal to binary, start by assuming that each digit in the octal number represents three digits in the corresponding binary number

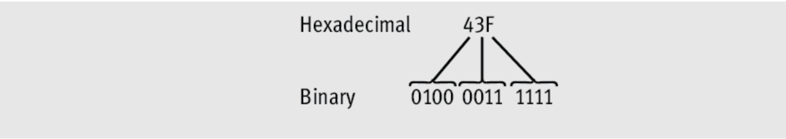


[FIGURE 4.3] The conversion of octal to binary

- To convert binary to octal, you begin at the right and factor the bits into groups of three bits each

Octal and Hexadecimal Numbers (continued)

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

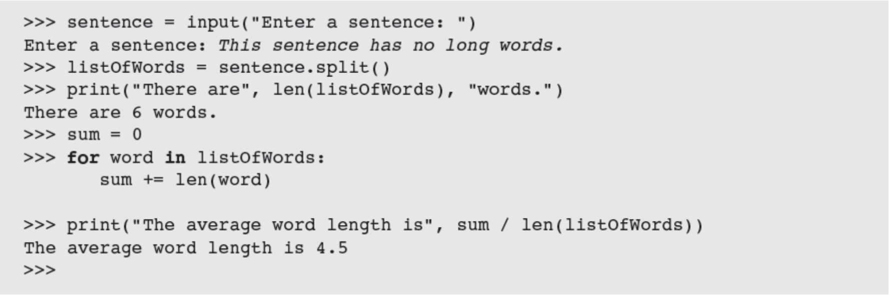


[FIGURE 4.4] The conversion of hexadecimal to binary

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

String Methods

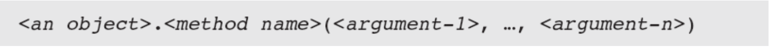
- Python includes a set of string operations called **methods** that make tasks like counting the words in a single sentence easy



```
>>> sentence = input("Enter a sentence: ")
Enter a sentence: This sentence has no long words.
>>> listOfWords = sentence.split()
>>> print("There are", len(listOfWords), "words.")
There are 6 words.
>>> sum = 0
>>> for word in listOfWords:
       sum += len(word)

>>> print("The average word length is", sum / len(listOfWords))
The average word length is 4.5
>>>
```

String Methods (continued)

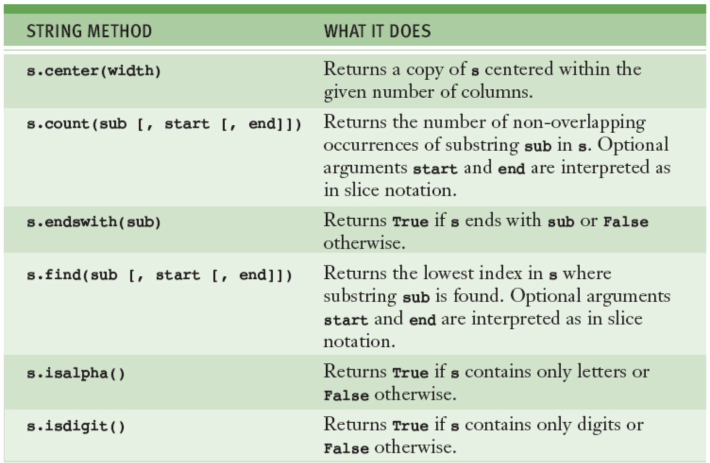
- A method behaves like a function, but has a slightly different syntax
  - A method is always called with a given data value called an **object**



```
<an object>.<method name>(<argument-1>, …, <argument-n>)
```

- Methods can expect arguments and return values
- A method knows about the internal state of the object with which it is called
- In Python, all data values are objects

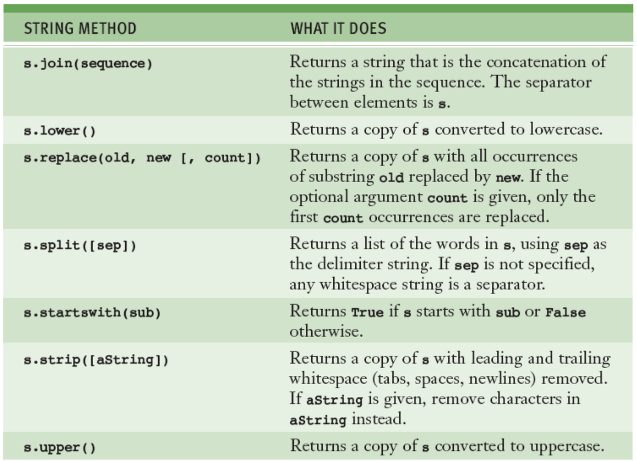
String Methods (continued)



STRING METHOD	WHAT IT DOES
`s.center(width)`	Returns a copy of `s` centered within the given number of columns.
`s.count(sub [, start [, end]])`	Returns the number of non-overlapping occurrences of substring `sub` in `s`. Optional arguments `start` and `end` are interpreted as in slice notation.
`s.endswith(sub)`	Returns `True` if `s` ends with `sub` or `False` otherwise.
`s.find(sub [, start [, end]])`	Returns the lowest index in `s` where substring `sub` is found. Optional arguments `start` and `end` are interpreted as in slice notation.
`s.isalpha()`	Returns `True` if `s` contains only letters or `False` otherwise.
`s.isdigit()`	Returns `True` if `s` contains only digits or `False` otherwise.

[TABLE 4.2] Some useful string methods, with the code letter `s` used to refer to any string

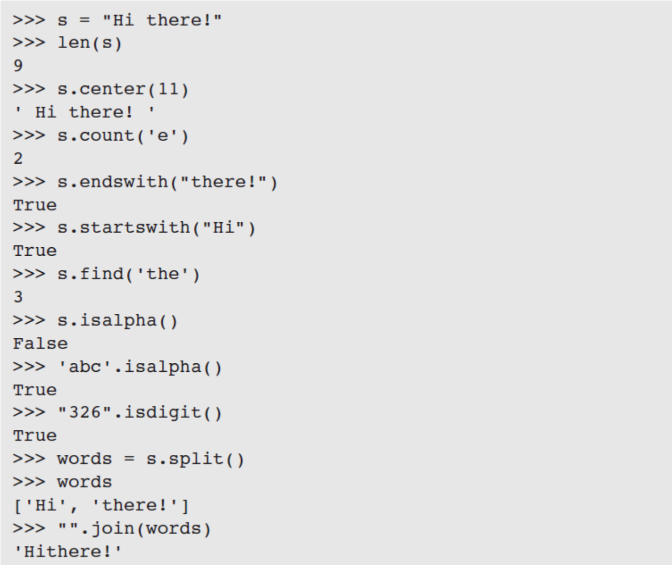
String Methods (continued)



STRING METHOD	WHAT IT DOES
`s.join(sequence)`	Returns a string that is the concatenation of the strings in the sequence. The separator between elements is `s`.
`s.lower()`	Returns a copy of `s` converted to lowercase.
`s.replace(old, new [, count])`	Returns a copy of `s` with all occurrences of substring `old` replaced by `new`. If the optional argument `count` is given, only the first `count` occurrences are replaced.
`s.split([sep])`	Returns a list of the words in `s`, using `sep` as the delimiter string. If `sep` is not specified, any whitespace string is a separator.
`s.startswith(sub)`	Returns `True` if `s` starts with `sub` or `False` otherwise.
`s.strip([aString])`	Returns a copy of `s` with leading and trailing whitespace (tabs, spaces, newlines) removed. If `aString` is given, remove characters in `aString` instead.
`s.upper()`	Returns a copy of `s` converted to uppercase.

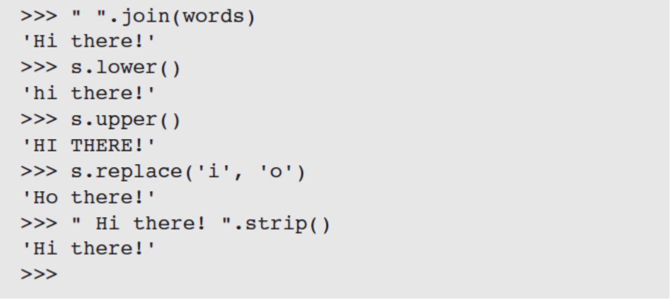
[TABLE 4.2] Some useful string methods, with the code letter `s` used to refer to any string

String Methods (continued)



```
>>> s = "Hi there!"
>>> len(s)
9
>>> s.center(11)
' Hi there! '
>>> s.count('e')
2
>>> s.endswith("there!")
True
>>> s.startswith("Hi")
True
>>> s.find('the')
3
>>> s.isalpha()
False
>>> 'abc'.isalpha()
True
>>> "326".isdigit()
True
>>> words = s.split()
>>> words
['Hi', 'there!']
>>> "".join(words)
'Hithere!'
```

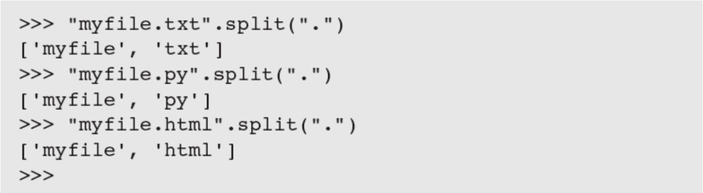
String Methods (continued)



```
>>> " ".join(words)
'Hi there!'
>>> s.lower()
'hi there!'
>>> s.upper()
'HI THERE!'
>>> s.replace('i', 'o')
'Ho there!'
>>> " Hi there! ".strip()
'Hi there!'
>>>
```

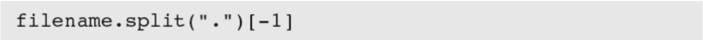
String Methods (continued)

- Example: extracting a filename’s extension



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

- The subscript **`[-1]`** extracts the last element
  - Can be used to write a general expression for obtaining any filename’s extension, as follows:



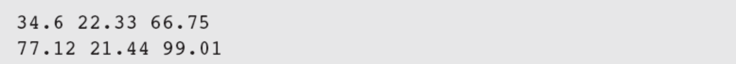
```
filename.split(".")[-1]
```

Text Files

- A text file is software object that stores data on permanent medium such as disk or CD
- 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 and Their Format

- Using a text editor such as Notepad or TextEdit, you can create, view, and save data in a text file

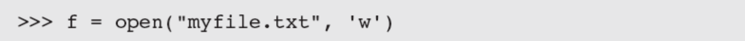


```
34.6 22.33 66.75
77.12 21.44 99.01
```

- All data output to or input from a text file must be strings

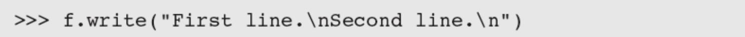
Writing Text to a File

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

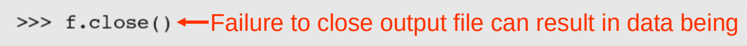


```
>>> f = open("myfile.txt", 'w')
```

  - If file does not exist, it is created
  - If it already exists, Python opens it; when data are written to the file and the file is closed, any data previously existing in the file are erased



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

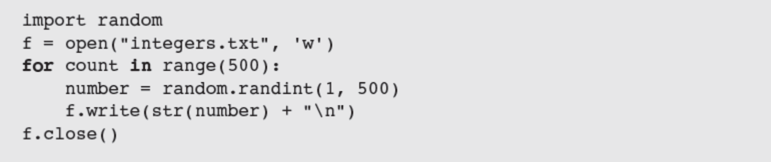


```
>>> f.close()
```

← Failure to close output file can result in data being lost

Writing Numbers to a File

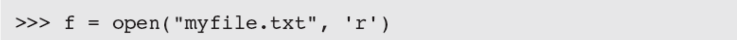
- The **`file`** method **`write`** expects a string as an argument
  - Other types of data must first be converted to strings before being written to output file (e.g., using `str`)



```
import random
f = open("integers.txt", 'w')
for count in range(500):
    number = random.randint(1, 500)
    f.write(str(number) + "\n")
f.close()
```

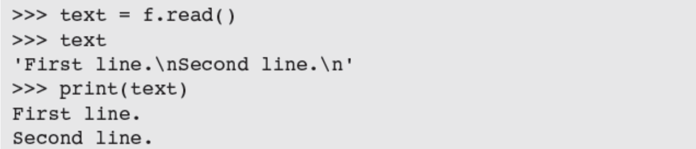
Reading Text from a File

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



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

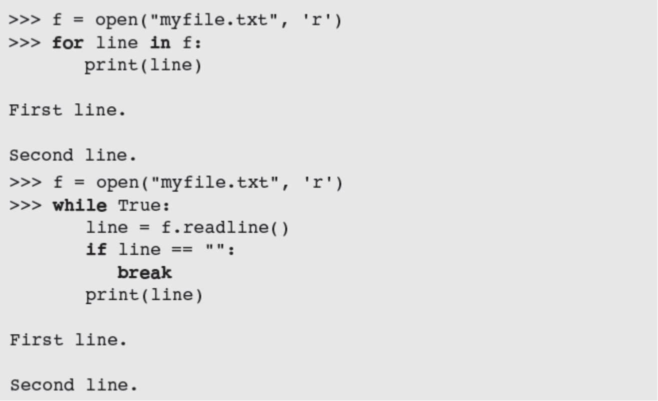
  - If the path name is not accessible from the current working directory, Python raises an error
- There are several ways to read data from a file
  - Example: the **`read`** method



```
>>> text = f.read()
>>> text
'First line.\nSecond line.\n'
>>> print(text)
First line.
Second line.
```

Reading Text from a File (continued)

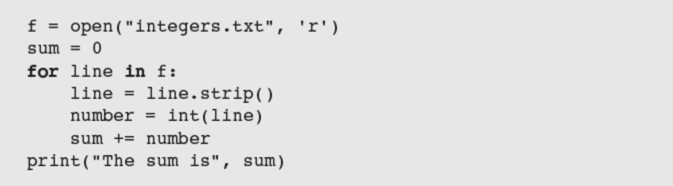
- After input is finished, **`read`** returns an empty string



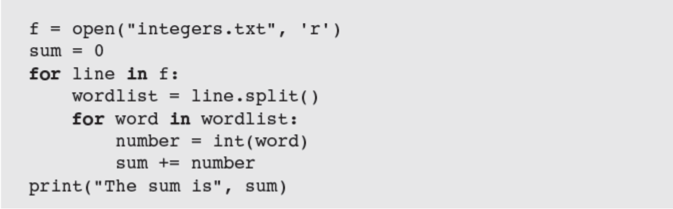
```
>>> f = open("myfile.txt", 'r')
>>> for line in f:
       print(line)

First line.

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

First line.

Second line.
```

Reading Numbers from a File

- Examples:

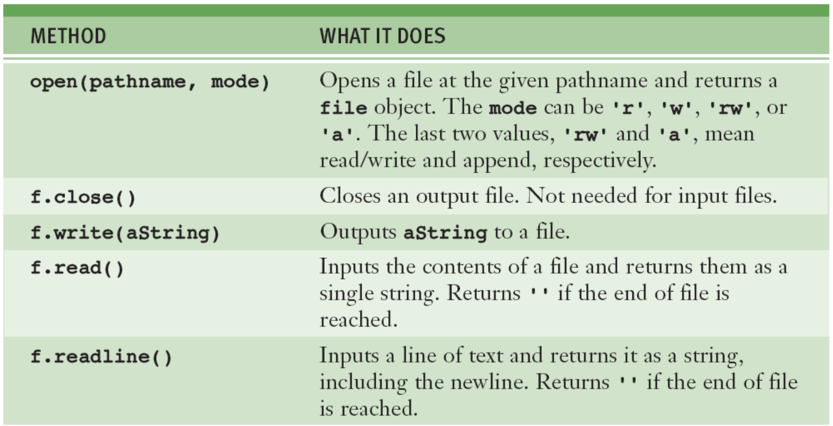


```
f = open("integers.txt", 'r')
sum = 0
for line in f:
    line = line.strip()
    number = int(line)
    sum += number
print("The sum is", sum)
```



```
f = open("integers.txt", 'r')
sum = 0
for line in f:
    wordlist = line.split()
    for word in wordlist:
        number = int(word)
        sum += number
print("The sum is", sum)
```

Reading Numbers from a File (continued)

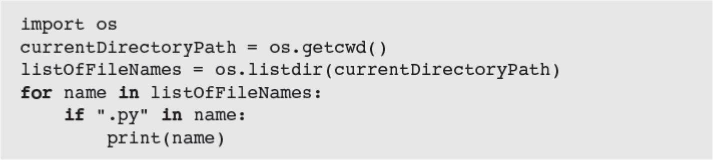


METHOD	WHAT IT DOES
`open(pathname, mode)`	Opens a file at the given pathname and returns a **`file`** object. The **`mode`** can be **`'r'`**, **`'w'`**, **`'rw'`**, or **`'a'`**. The last two values, **`'rw'`** and **`'a'`**, mean read/write and append, respectively.
`f.close()`	Closes an output file. Not needed for input files.
`f.write(aString)`	Outputs **`aString`** to a file.
`f.read()`	Inputs the contents of a file and returns them as a single string. Returns **`''`** if the end of file is reached.
`f.readline()`	Inputs a line of text and returns it as a string, including the newline. Returns **`''`** if the end of file is reached.

[TABLE 4.3] Some **`file`** operations

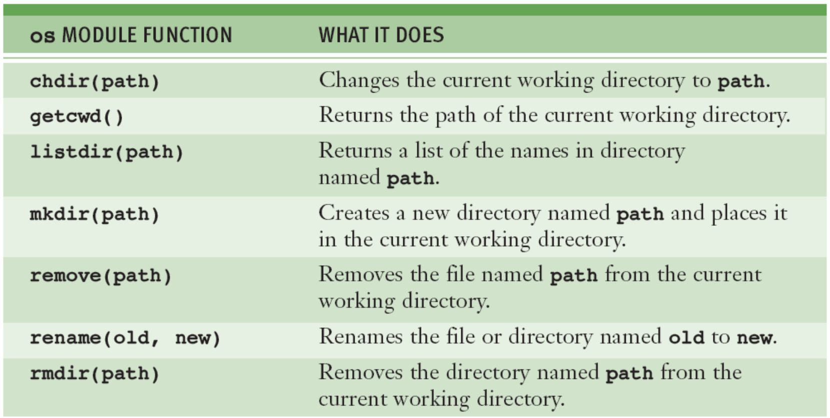
Accessing and Manipulating Files and Directories on Disk

- When designing Python programs that interact with files, it's a good idea to include error recovery
- For example, before attempting to open a file for input, you should check to see if file exists
  - Function **`os.path.exists`** supports this checking
- Example: To print all of the names of files in the current working directory with a **`.py`** extension:



```
import os
currentDirectoryPath = os.getcwd()
listOfFileNames = os.listdir(currentDirectoryPath)
for name in listOfFileNames:
    if ".py" in name:
        print(name)
```

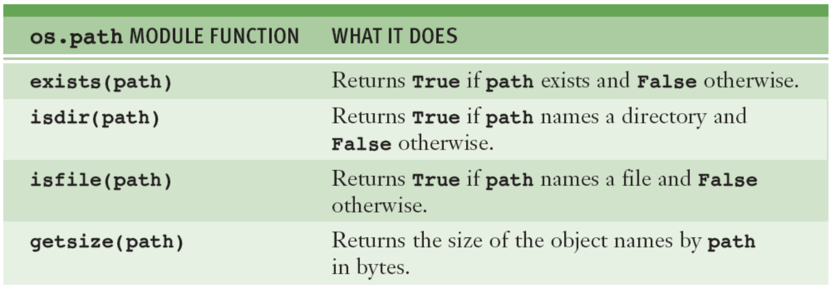
Accessing and Manipulating Files and Directories on Disk (continued)



os MODULE FUNCTION	WHAT IT DOES
`chdir(path)`	Changes the current working directory to **`path`**.
`getcwd()`	Returns the path of the current working directory.
`listdir(path)`	Returns a list of the names in directory named **`path`**.
`mkdir(path)`	Creates a new directory named **`path`** and places it in the current working directory.
`remove(path)`	Removes the file named **`path`** from the current working directory.
`rename(old, new)`	Renames the file or directory named **`old`** to **`new`**.
`rmdir(path)`	Removes the directory named **`path`** from the current working directory.

[TABLE 4.4] Some file system functions

Accessing and Manipulating Files and Directories on Disk (continued)

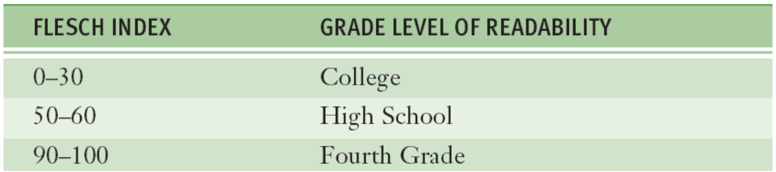


os.path MODULE FUNCTION	WHAT IT DOES
`exists(path)`	Returns **`True`** if **`path`** exists and **`False`** otherwise.
`isdir(path)`	Returns **`True`** if **`path`** names a directory and **`False`** otherwise.
`isfile(path)`	Returns **`True`** if **`path`** names a file and **`False`** otherwise.
`getsize(path)`	Returns the size of the object names by **`path`** in bytes.

[TABLE 4.5] More file system functions

Case Study: Text Analysis

- In 1949, Dr. Rudolf Flesch proposed a measure of text readability known as the **Flesch Index**
  - Index is based on the average number of syllables per word and the average number of words per sentence in a piece of text
  - Scores usually range from 0 to 100, and indicate readable prose for the following grade levels:



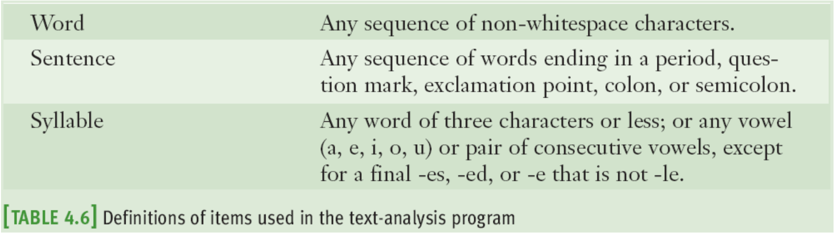
FLESCH INDEX	GRADE LEVEL OF READABILITY
0–30	College
50–60	High School
90–100	Fourth Grade

Case Study: Request

- Write a program that computes the Flesch index and grade level for text stored in a text file

Case Study: Analysis

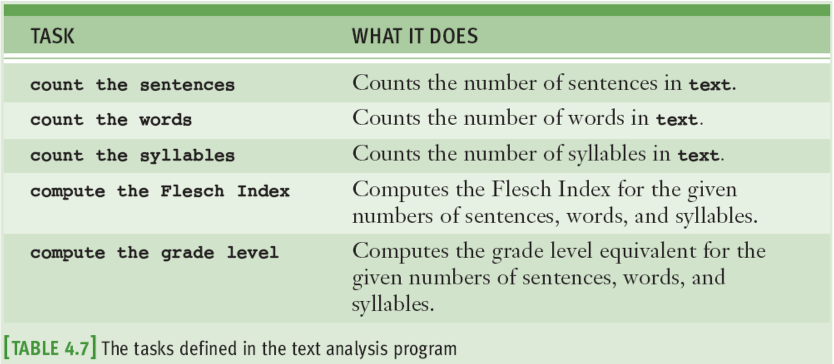
- Input is the name of a text file
- Outputs are the number of sentences, words, and syllables in the file, as well as the file’s Flesch index and grade-level equivalent



Word	Any sequence of non-whitespace characters.
Sentence	Any sequence of words ending in a period, question mark, exclamation point, colon, or semicolon.
Syllable	Any word of three characters or less; or any vowel (a, e, i, o, u) or pair of consecutive vowels, except for a final -es, -ed, or -e that is not -le.

[TABLE 4.6] Definitions of items used in the text-analysis program

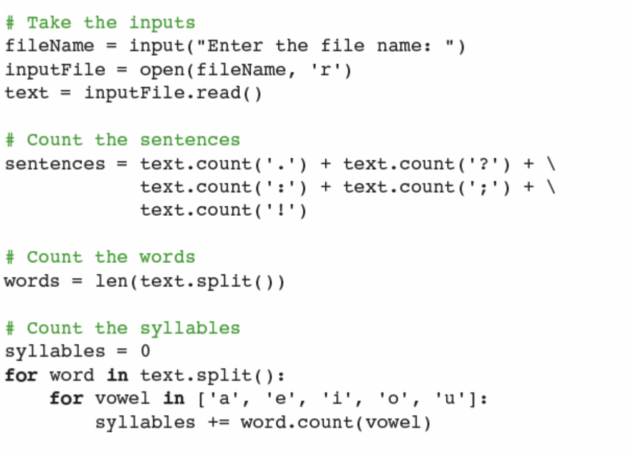
Case Study: Design



TASK	WHAT IT DOES
`count the sentences`	Counts the number of sentences in **`text`**.
`count the words`	Counts the number of words in **`text`**.
`count the syllables`	Counts the number of syllables in **`text`**.
`compute the Flesch Index`	Computes the Flesch Index for the given numbers of sentences, words, and syllables.
`compute the grade level`	Computes the grade level equivalent for the given numbers of sentences, words, and syllables.

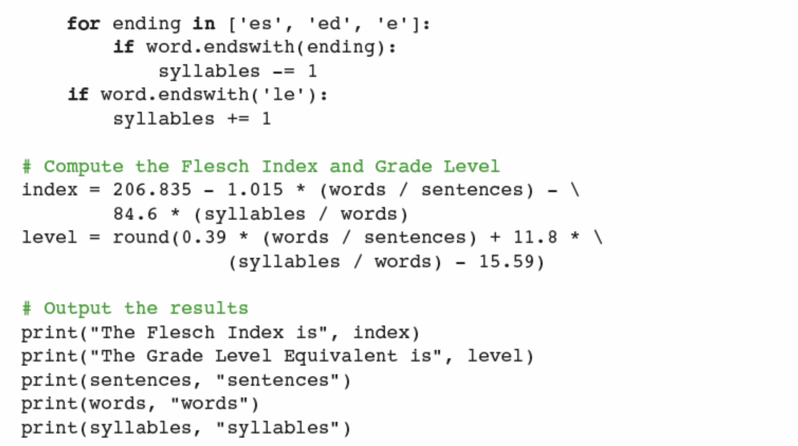
[TABLE 4.7] The tasks defined in the text analysis program

Case Study: Implementation (Coding)



```
# Take the inputs
fileName = input("Enter the file name: ")
inputFile = open(fileName, 'r')
text = inputFile.read()

# Count the sentences
sentences = text.count('.') + text.count('?') + \
            text.count(':') + text.count(';') + \
            text.count('!')

# Count the words
words = len(text.split())

# Count the syllables
syllables = 0
for word in text.split():
    for vowel in ['a', 'e', 'i', 'o', 'u']:
        syllables += word.count(vowel)
```

Case Study: Implementation (Coding) (continued)



```
    for ending in ['es', 'ed', 'e']:
        if word.endswith(ending):
            syllables -= 1
    if word.endswith('le'):
        syllables += 1

# Compute the Flesch Index and Grade Level
index = 206.835 - 1.015 * (words / sentences) - \
        84.6 * (syllables / words)
level = round(0.39 * (words / sentences) + 11.8 * \
                  (syllables / words) - 15.59)

# Output the results
print("The Flesch Index is", index)
print("The Grade Level Equivalent is", level)
print(sentences, "sentences")
print(words, "words")
print(syllables, "syllables")
```

Case Study: Testing

- **Bottom-up testing:**
  - Each task is coded and tested before it is integrated into the overall program
  - After you have written code for one or two tasks, you can test them in a short script, called a **driver**

Summary

- A string is a sequence of zero or more characters
  - Immutable data structure
  - `[]` used to access a character at a given position
    - Can also be used for **slicing** (`[<start>:<end>]`)
- `in` operator is used to detect the presence or absence of a substring in a string
- Method: operation that is used with an object
- The string type includes many useful methods for use with string objects

Summary (continued)

- A text file is a software object that allows a program to transfer data to and from permanent storage
- A **`file`** object is used to open a connection to a text file for input or output
  - Some useful methods: **`read`**, **`write`**, **`readline`**
- **`for`** loop treats an input file as a sequence of lines
  - On each pass through the loop, the loop's variable is bound to a line of text read from the file