CSC 221: Introduction to Programming

Fall 2011

Input & file processing

- input vs. raw_input
- files: input, output
- opening & closing files
- read(), read(#), readline()
- print, write

1

User interaction

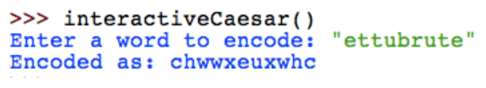
so far, all information needed by a function has been passed as inputs

- e.g., Caesar cipher had string to be encoded as input
```
>>> caesar("ettubrute")
```
- e.g., hailstone sequence had starting number as input
```
>>> hailstone(5)
```

this was reasonable when input sizes were small

- with text processing, may now want to enter lots of text

alternatively, could have the function *prompt* the for the input value



```
>>> interactiveCaesar()
Enter a word to encode: "ettubrute"
Encoded as: chwwxeuxwhc
```

2

User input via `input`

the input function reads a value in from the user

```
input(PROMPT_MESSAGE)
```

- when called, the (optional) prompt message is displayed
- the value entered by the user is returned by the function

```
ALPHABET = "abcdefghijklmnopqrstuvwxyz"

def interactiveCaesar():
    word = input("Enter a word to encode: ")
    encoded = ""
    for ch in word:
        index = ALPHABET.find(ch)
        nextIndex = (index + 3) % 26
        encoded += ALPHABET[nextIndex]
    print "Encoded as:", encoded
```

is this better than the old version?

3

Better design

most would agree that the first solution is better

- more general – could call the function in a variety of applications
- doesn't matter where the string came from
- since the value is returned, can be used in different settings

better solution

- if we really want to utilize user input, keep general-purpose version
- call it from another function that does the input.output

```
from words import caesar

def interactiveCaesar():
    word = input("Enter a word to encode: ")
    encoded = caesar(word)
    print "Encoded as:", encoded
```

4

raw_input

entering text with quotes can be tedious

- a alternative input function does not require the quotes (the quotes are implicit)

```
raw_input(PROMPT_MESSAGE)
```

```
from words import caesar

def interactiveCaesar():
    word = raw_input("Enter a word to encode: ")
    encoded = caesar(word)
    print "Encoded as:", encoded
```

```
>>> interactiveCaesar()
Enter a word to encode: ettubrute
Encoded as: hwwxeuxwh
```

5

Inputting numbers

`input` and `raw_input` can read in numbers as well

- since the hailstone function prints its results, reading the initial value seems reasonable

```
def hailstone():
    num = input("Enter the starting number: ")
    print num
    while num != 1:
        if num % 2 == 0:
            num = num / 2
        else:
            num = 3*num + 1
        print num
```

6

Raw numbers

have to be careful reading numbers with `raw_input`

- `raw_input` always returns a string (assumes implicit quotes are the input)

```
def hailstone():
    num = raw_input("Enter the starting number: ")
    print num
    while num != 1:
        if num % 2 == 0:
            num = num / 2
        else:
            num = 3*num + 1
        print num
```

```
>>> hailstone()
Enter the starting number: 5
5

Traceback (most recent call last):
  File "<pyshell#13>", line 1, in <module>
    hailstone()
  File "/Users/davereed/Documents/Classes/CSC221/CSC 221.F11/Code/file.py",
line 39, in hailstone
    if num % 2 == 0:
TypeError: not all arguments converted during string formatting
```

```
def hailstone():
    num = int(raw_input("Enter the starting number: "))
    print num
    while num != 1:
        if num % 2 == 0:
            num = num / 2
        else:
            num = 3*num + 1
        print num
```

can use `int()` to convert the string into a number value

7

Reading from a file

for larger amounts of input, more convenient to enter into a file and read directly from the file

```
def showFile(filename):
    infile = open(filename, "r")
    text = infile.read()
    print text
    infile.close()
```

`open(filename, "r")` opens the file named `filename` for input

- returns a file object

the `read()` method reads the entire contents of the file object into a string

- returns that string

the `close()` method closes the file

- should call when done reading

8

read() vs. read(#) vs. readline()

```
from sys import stdout

def showFile(filename):
    infile = open(filename, "r")
    text = infile.read(1)
    while text != "":
      stdout.write(text)
      text = infile.read(1)
    infile.close()
```

`read` has an optional input, the number of characters to read

- e.g., `infile.read(1)` reads a single character
- returns empty string if at end of file

recall that each call to `print` displays a new line

- could print char-by-char using the `stdout` file object (from `sys.py`)
- the `write` method does not skip lines

```
def showFile(filename):
    infile = open(filename, "r")
    text = infile.readline()
    while text != "":
      stdout.write(text)
      text = infile.readline()
    infile.close()
```

`readline` reads an entire line of text

- returns empty string if at end of file

9

Example: file stats

suppose we want to generate stats on a file

- # of lines
- # of characters
- average # of characters per line

which type of read do we need?

```
def fileStats(filename):
    numChars = 0
    numLines = 0
    infile = open(filename, "r")
    text = infile.readline()
    while text != "":
        numLines += 1
        numChars += len(text)
        text = infile.readline()
    print "Number of lines =", numLines
    print "Number of chars =", numChars
    print "Avg line length =", float(numChars)/numLines
```

10

output files

```
def copyFile1(oldfile, newfile):
    infile = open(oldfile, "r")
    outfile = open(newfile, "w")
    text = infile.read()
    outfile.write(text);
    infile.close()
    outfile.close()

def copyFile2(oldfile, newfile):
    infile = open(oldfile, "r")
    outfile = open(newfile, "w")
    text = infile.read(1)
    while text != "":
        outfile.write(text);
        text = infile.read(1)
    infile.close()
    outfile.close()

def copyFile3(oldfile, newfile):
    infile = open(oldfile, "r")
    outfile = open(newfile, "w")
    text = infile.readline()
    while text != "":
        outfile.write(text);
        text = infile.readline()
    infile.close()
    outfile.close()
```

here, three different versions of a function that copies a file

- open an output file using "w" as the 2nd input
- can write to an output file object using the `write` method

11

Example: encoding a file

could extend our encryption algorithms to files

- instead of requiring the user to enter each string, read in from a file
- could save encoded/decoded text in a different file
- or, could overwrite the original file by using it for both "r" and "w"

```
from words import rotate

def rotateFile(numChars, filename):
    infile = open(filename, "r")
    text = infile.read()
    infile.close()
    outfile = open(filename, "w")
    coded = rotate(numChars, text)
    outfile.write(coded)
```

12

Browsing for files

the `open` function assumes a relative file address

- will look for the filename in the same directory as the program
- can specify a path prefix, relative to the current directory

```
infile = open("../Data/"+filename, "r")

outfile = open("csc221/Results/"+filename, "w")
```

if you want to be able to search for files, there is a module for that

```
from tkFileDialog import askopenfilename, asksaveasfilename

inputfilename = askopenfilename()

outputfilename = asksaveasfilename()
```

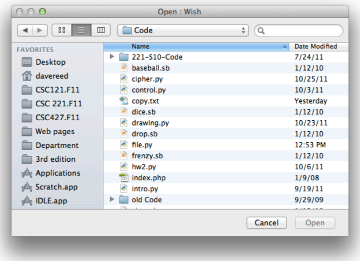
13

File dialogs

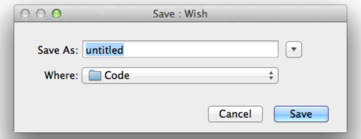
```
from tkFileDialog import askopenfilename, asksaveasfilename

def showFile():
    filename = askopenfilename()
    infile = open(filename, "r")
    text = infile.readline()
    while text != "":
      stdout.write(text)
      text = infile.readline()
    infile.close()

def copyFile():
    oldfilename = askopenfilename()
    infile = open(oldfilename, "r")
    newfilename = asksaveasfilename()
    outfile = open(newfilename, "w")
    text = infile.readline()
    while text != "":
        outfile.write(text);
        text = infile.readline()
    infile.close()
    outfile.close()
```



Name	Date Modified
221-S10-Code	7/24/11
baseball.sb	1/12/10
cipher.py	10/25/11
control.py	10/3/11
copy.txt	Yesterday
dice.sb	1/12/10
drawing.py	10/23/11
drop.sb	1/12/10
file.py	12:53 PM
frenzy.sb	1/12/10
hw2.py	10/6/11
index.php	1/9/08
intro.py	9/19/11
old Code	9/29/09



Save As: untitled
Where: Code

14

Exercise: pig latin dictionary

on the class code page is a dictionary of ~75K words: words.txt

you are to download this file, and generate a corresponding pig latin dictionary

- the dictionary has one word per line
- you may ignore bad results due to spaces or capitalization

CAREFUL:

- the `readline()` method (for file objects) returns the entire line of text, including end-of-line characters
- the `strip()` method (for strings) can be called to remove leading and trailing whitespace

- but if you strip the end-of-line characters off, successive writes will put everything on one line
- you can write "\n" to print a line break in the file

15