Topics

This week:
- File input and output
- OS module and path
- encoding trees
Project 3, todo 6

The word “extract” seems to need some lawyering… here is what we want you to do:

- For hiding a “guest picture” in a “host picture”:
  - Get the (i,k) pixel of the host, say red is \textit{aaaaabbbb}
  - Get the (i,k) pixel of the guest, say red is \textit{ccccddddd}
  - Combine as (i,k) pixel of the result with red \textit{aaaacccccc}
  - Do this for all pixels, for their red, green and blue components

- For recovering the guest picture:
  - Get the (i,k) pixel, say red = \textit{aaaacccccc}
  - Make the (i,k) pixel of the result with red \textit{cccc0000}
  - Do this for all pixels, for their red, green and blue components
Operations:
- <
- >
- &
- |
Announcements

- Project 4: First Team Project
  - To be published this Thursday, Oct 18
  - **Urgently**: Pick a team and register it with us, or let us know you need a team
  - Instructions are on the course wiki under *Projects*
  - Teams should be of size 3

- Mid-semester course evaluations are now open
  - Course home page has details
Input/Output

- Thus far we have only been able to get input from the user and produce output to the screen
  - Limits the scope of our programs
  - What if we wanted to search in a book?
  - We would have to type the book into our program each time!

- Our output was limited by what we could display to the screen
  - After our program completed the output was gone!
Files: Multi-line Strings

- A *file* is a sequence of data that is stored in secondary, persistent memory (such as a disk drive).
- Files can contain any data type, but the easiest to work with would be text.
- A text file usually contains more than one line of text.
- Python uses the standard newline character (`\n`) to mark line breaks.
Multi-Line Strings

- Hello
  World

  Goodbye 32

- When stored in a file:
  Hello
  World

  Goodbye 32
Multi-Line Strings

- This is exactly the same thing as embedding \n in print statements.

- Remember, these special characters only affect things when printed. They don’t do anything during evaluation.
File Processing

- The process of *opening* a file involves associating a file on disk with an object in program memory.

- We can manipulate the file by manipulating this object.
  - Read from the file
  - Write to the file
File Processing

- 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.

- In some cases, not properly closing a file could result in data loss.
File Processing Example

- Reading a file into a word processor
  - File opened for input
  - Contents read into RAM
  - File closed
  - Changes to the file are made to the copy stored in memory, not on the disk.

- Save:
  - Backup copy of file made
  - File opened for output
  - RAM version written to file
  - File closed
File Processing

- Working with text files in Python
  - Associate a file with a file object using the open function
    `<filevar> = open(<name>, <mode>)`
  
  - Name is a string with the actual file name on the disk. The mode is either ‘r’ or ‘w’ depending on whether we are reading or writing the file.
    - There are also other modes
  
- Infile = open("numbers.dat", "r")
File Methods

- `<file>.read()` – returns the entire remaining contents of the file as a single (possibly large, multi-line) string
- `<file>.readline()` – returns the next line of the file. This is all text – up to and including the next newline character at the end of the line
- `<file>.readlines()` – returns a list of the remaining lines in the file. Each list item is a single line including the newline characters.
File Processing

# Prints a file to the screen.
def main():
    fname = input("Enter filename: ")
    infile = open(fname,'r')
    data = infile.read()
    print(data)
    infile.close()
main()

- Prompt the user for a file name
- Open the file for reading
- The file is read as one single string and stored in the variable named data
File Processing

- readline can be used to read the next line from a file, including the trailing newline character

  ```python
  infile = open(someFile, "r")
  for i in range(5):
      line = infile.readline()
      print(line[:-1])
  infile.close()
  ```

- This reads the first 5 lines of a file, then closes it

- Slicing is used to strip out the newline characters at the ends of the lines
Another way to loop through the contents of a file is to read it in with readlines and then loop through the resulting list.

```python
infile = open(someFile, "r")
for line in infile.readlines():
    # Line processing here
infile.close()
```
File Processing

- Python treats the file object itself as a sequence of lines!

- `infile = open(someFile, "r")`
  for line in infile:
    # process the line here
  infile.close()
File Processing

- Opening a file for writing prepares the file to receive data.

- If you open an existing file for writing, you wipe out the file’s contents. If the named file does not exist, a new one is created.

- `Outfile = open("mydata.out", "w")`

- Actual writing:
  - `print(<expressions>, file=Outfile)`
  - `Outfile.write(<string>)`
  - Print takes multiple arguments; write only one, a string.
Example Program: Batch Usernames

- *Batch* mode processing is where program input and output are done through files (the program is not designed to be interactive)

- Let’s create usernames for a computer system where the first and last names come from an input file.
Helpful String Method

- One of these methods is `split`. This will split a string into substrings based on spaces.

```python
>>> "Hello string methods!".split()
["Hello", 'string', 'methods!']
```
Another String Method

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

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

>>>`
CQ: How many?

What does the following program print?

S = "a,b,,d,e"
print(len(S.split("","")))

A. 8
B. 5
C. 4
Example Program: Batch Usernames

# userfile.py
#    Program to create a file of usernames in batch mode.

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

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

    # open the files
    infile = open(infileName, 'r')
    outfile = open(outfileName, 'w')
Example Program: Batch Usernames

# process each line of the input file
for line in infile:
    # get the first and last names from line
    first, last = line.split()
    # create a username
    uname = (first[0]+last[:7]).lower()
    # write it to the output file
    outfile.write(uname+"\n")

# close both files
infile.close()
outfile.close()

print("Usernames have been written to", outfileName)
Example Program: Batch Usernames

- Things to note:
  - It’s not unusual for programs to have multiple files open for reading and writing at the same time.
  - The `<string>`.lower() method is used to convert the names into all lower case, in the event the names are mixed upper and lower case.
  - We manually added a newline “\n” after each name, this ensures each id is on a separate line
    - What happens if we do not do this?

- When we split the string we were “parsing”
Methods on Files

- object.method() syntax: this time files are our object
  - Example: `file = open("myfile", "w")`

- `file.read()` -- reads the file as one string
- `file.readlines()` – reads the file as a list of strings
- `file.readline()` – reads one line from the file
- `file.write()` – allows you to write a string to a file
- `file.close()` – closes a file
Announcements

- Project 4: First Team Project
  - To be published today
  - **Urgently**: Pick a team and register it with us, or let us know you need a team
  - Instructions are on the course wiki under **Projects**
  - Teams should be of size 3

- Mid-semester evaluations are now open
  - Course home page has details
Example: Writing to a File

```python
def formLetter(gender, lastName, city):
    file = open("formLetter.txt","w")
    file.write("Dear ")
    if gender == "F":
        file.write("Ms. "+lastName+":
")
    if gender == "M":
        file.write("Mr. "+lastName+":
")
    file.write("I am writing to remind you of the offer ")
    file.write("that we sent to you last week. Everyone in ")
    file.write(city+" knows what an exceptional offer this is!" )
    file.write("Sincerely ,\n")
    file.write("I.M. Acrook, Attorney at Law")
    file.close()
```
Dear Mr. Guzdial:

I am writing to remind you of the offer that we sent to you last week. Everyone in Decatur knows what an exceptional offer this is!

Sincerely,

I.M. Acrook, Attorney at Law
File Output is Important

- Allows your programs to produce results that are viewable by other programs
- Allows you to retain computed results after a program terminates
Python’s Standard Library

- Python has an extensive *library* of modules that come with it.

- The Python standard library includes modules that allow us to access the Internet, deal with time, generate random numbers, and…access files in a directory.
Accessing pieces of a module

- We access the additional capabilities of a module using dot notation, after we *import* the module.

- How do you know what code is there?
  - Check the online documentation.
  - There are books like *Python Standard Library* that describe the modules and provide examples.
The OS Module

- The OS module provides an interface to the underlying operating system
  - Allows your program to request resources/actions to be performed on your program’s behalf
- `os.chdir(path)` – changes the current working directory
- `os.listdir(path)` – returns a list of all entries in the directory specified by path
- `os.curdir` – returns what directory the program is executing in (i.e. the current directory)
The OS Path submodule

- Once you import os – import os – you can also use the path module

- os.path.isfile(path) – returns true if the path specified is a file, returns false otherwise
  - Use this method to perform a check to see if the user provided input for a valid file
import os

def chkPath():
    path = input("type a file name: ")
    if os.path.isfile(path):
        return True,path
    else:
        return False,path

def main():
    chk = True
    while chk:
        chk,file = chkPath()
        if chk: print('file', file, 'exists')
        else: print('file', file, 'does not exist')

main()
Trees

- Tree data structure
- Encoding and access
- Some uses and operations
Example: Directory trees

![Example directory tree](image)
We call this structure a tree
How might we encode such a structure?

Tree = ['Root', 'Leaf1', 'Leaf2', 'Leaf3']
Trees can be more complex

Tree = ['Root', 'Leaf1', 'Leaf2', ['Node1', 'Leaf3', 'Leaf4', 'Leaf5']]
Trees can be more complex

Tree = ['Root', 'Leaf1', 'Leaf2', ['Node1', 'Leaf3', 'Leaf4', 'Leaf5']]
Trees can be more complex

Tree = ['Root', 'Leaf1', 'Leaf2', ['Node1', 'Leaf3', 'Leaf4', 'Leaf5']]
Trees can be more complex

Tree = ['Root', 'Leaf1', 'Leaf2', ['Node1', 'Leaf3', 'Leaf4', 'Leaf5']]
Trees can be more complex

Tree = ['Root', ['Node1', 'Leaf0', 'Leaf1'], 'Leaf2', ['Node2', 'Leaf3', 'Leaf4', 'Leaf5']]
Trees can be more complex

```
Root
  Node1
    Leaf0
    Leaf1
  Leaf2
    Leaf3
    Leaf4
      Node2
        Leaf5
        Leaf6
  Node2
  Node3
```

Tree = ['Root', ['Node1', 'Leaf0', 'Leaf1'], 'Leaf2', ['Node2', 'Leaf3', 'Leaf4', ['Node3', 'Leaf5', 'Leaf6']]]
What is the intuition

- Each sub list encodes a ‘node’ of the tree plus the ‘branches’ of the tree
- We can think of each sub list as a ‘subtree’ rooted in the node in the leading element position
- We can use indices (the bracket notation [ ] ) to select out elements or subtrees
How can we select out the leaves?

Tree = ['Root', 'Leaf1', 'Leaf2', 'Leaf3']
Indices allow us to “traverse” the tree.

Tree = ['Root', ['Node1', 'Leaf0', 'Leaf1'],
             'Leaf2',
             ['Node2', 'Leaf3', 'Leaf4', ['Node3', 'Leaf5', 'Leaf6']]]
Indices allow us to “traverse” the tree

```
Tree[3] = ['Node2', 'Leaf3', 'Leaf4', ['Node3', 'Leaf5', 'Leaf6']]
Tree[3][3] = ['Node3', 'Leaf5', 'Leaf6']
```
Indices allow us to “traverse” the tree

```
Tree[3][0] = 'Node2'
Tree[3][1] = 'Leaf3'
Tree[3][2] = 'Leaf4'
Tree[3][3] = ['Node3', 'Leaf5', 'Leaf6']
Tree[3][3][0] = 'Node3'
Tree[3][3][1] = 'Leaf5'
```
CQ: How do we select ‘Leaf4’ from the Tree?

Tree = ['Root', ['Node1', 'Leaf0', 'Leaf1'], 'Leaf2', ['Node2', 'Leaf3', 'Leaf4', ['Node3', 'Leaf5', 'Leaf6']]]

A: Tree[4][3]
B: Tree[3][2]
C: Tree[8]
Example: Expressions

\[ E = 3 \times 5 + 2 \times (6 - 1) \]

\[ [+,[*,3,5],[*,2,[-,6,1]]] \]
Operations on Trees

- Trees, since they are encoded via lists, support the same operations that lists support. But what do they mean?

- From a ‘tree’ perspective:
  - We can make one tree subtree of another (substitution or extension)
  - We can replace a subtree with a leaf (evaluation)
  - We can drop a subtree (pruning)
  - We can visit each tree node in some order (e.g., depth-first traversal, preorder traversal, etc.)
T2 as subtree of T1 (Extension)

T1 = ['root1', 'leaf11', 'leaf12']
T2 = ['root2', 'leaf21', 'leaf22', 'leaf23']

T1.append(T2)
T1.insert(2, T2)
T2 as subtree of T1 (Substitution)

T1 = ['root1', 'leaf11', 'leaf12']
T2 = ['root2', 'leaf21', 'leaf22', 'leaf23']

T1[2] = T2
Tree Evaluation

Requires a notion of evaluating a list. Typical list structure

\[ [\text{operation, operand}, \ldots, \text{operand}] \rightarrow \text{operand} \]

where operands are compatible with the operation
Tree Evaluation

Iterable recursively

```
+  
|   +  
|   |   *  
|   |   |   *  
|   |   |   |   -  
|   |   |   |   |   6  
|   |   |   |   |   1  
3   5   2   10
```
Tree pruning

- Example: Computer Chess
  - Tree records possible moves and responses to some depth (around 6)
  - Each subtree is graded by how desirable its result configuration is. Undesirable choices are dropped

```
T = ['root1', 'leaf1', ['root2', 'leaf21', 'leaf22', 'leaf23'], 'leaf12']
```
del T[2]
Tree Traversals

Preorder:  +, *, 3, 5, *, 2, -, 6, 1

Post order: 3, 5, *, 2, 6, 1, -, *, +

[[+, [*], [*]], [-, [6, 1]]]
Why are trees important?

- They are a fundamental structure in computer science
- They enable us to search very quickly, for instance
  - We will revisit trees later in the course
- What have we covered so far that is simple:
  - We can encode a tree as a list of lists
  - Given this encoding, we can select elements like for complex lists, using the index mechanism[
- What is more intricate:
  - Tree constructions and operations arising form the application
Announcements

- Project 4: First Team Project
  - To be published today
  - **Urgently**: Pick a team and register it with us, or let us know you need a team
  - Instructions are on the course wiki under **Projects**
  - Teams should be of size 3

- Mid-semester evaluations are now open
  - Course home page has details