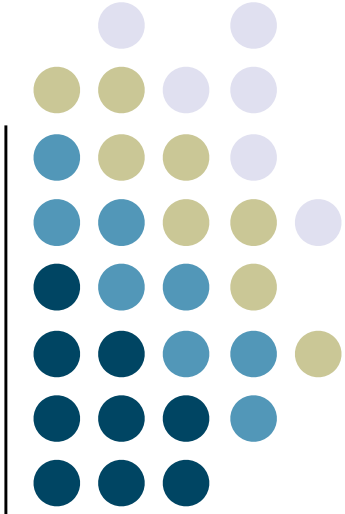
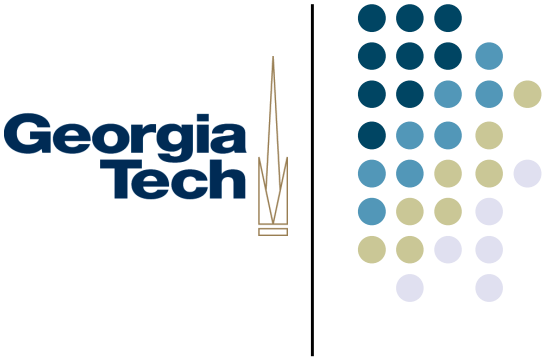
Accessing Files in Python



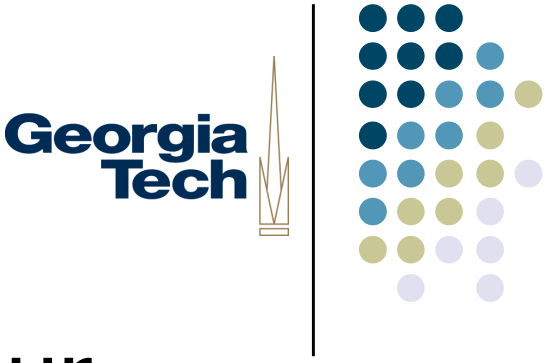
Georgia Tech

Learning Objectives



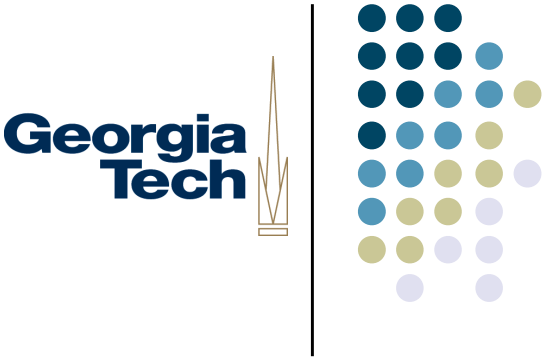
- Concepts about files in Python
- How to open files
- Different ways of reading files
- How to write out files
- How to read then search, count, etc. on files
- Concepts about exceptions and how to use try blocks in your code

Files



- Lots of data stored in files on your computer's disk
- Want to be able to open those files, read from them, and write to them

Types of Files



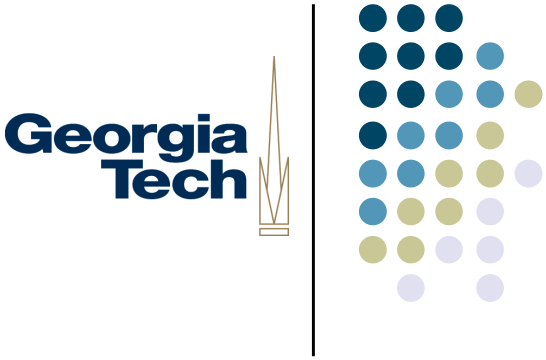
- Two main types
  - Text (ASCII or Unicode) We'll use
  - Binary

- Text files can be opened in Sublime or NotePad and it will make sense. Binary won't.

Sequential Access

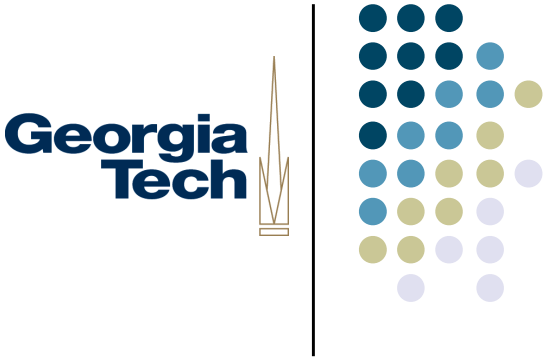
- We will work with files in a sequential access mode
  - Think of the file as one long sequence of characters
  - To read something at the end, must read all the prior stuff first
  - Like a cassette tape, not a CD

Main Process



- 1. Open the file
  - Output: Often are creating the file
  - Input: Reading data from it
- 2. Process the file
  - Either read or write
- 3. Close the file

Opening a file

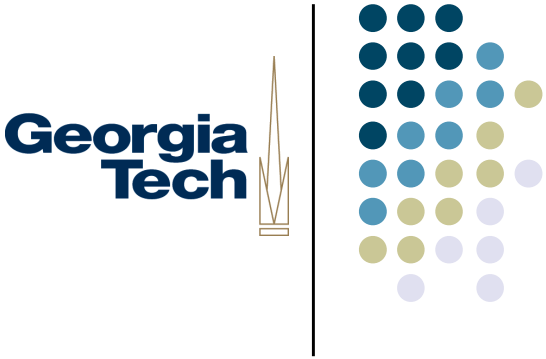



```
file_var = open(filename, mode)
```

`filename` – string specifying the name of the file
`mode` – string specifying mode in which file will be opened

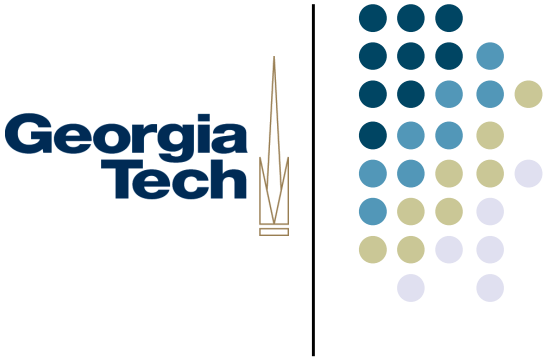
Modes
`'r'` – reading only
`'w'` – writing. If file already exists, erase it. If file doesn't exist, create it.
`'a'` – append style writing. If file already exists, all written data will be put at end. If doesn't exist, create it.

Examples



```
f1 = open("orders.txt", "r")
f2 = open("new_work.txt", "w")
```

File Functions/Methods



`file.read()` – Reads the whole file as one big string

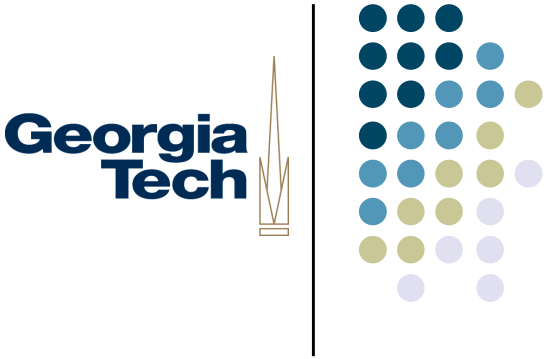
`file.readlines()` – Reads whole file into a list where each element is a single line

You can only use those two once for an open file

`file.write(somestring)` – Writes somestring to the file

`file.close()` – Closes the file. Good form to do it. If you were writing, it insures that all data does go out to the file.

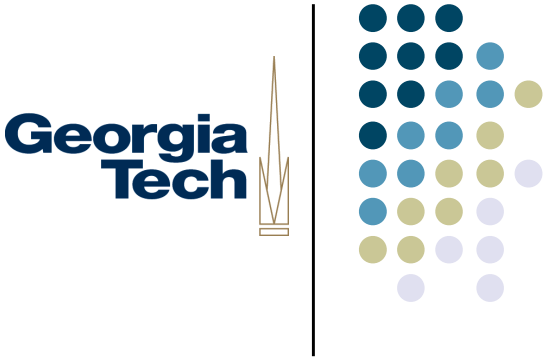
Reading a File



All at once method

```
def main():
    infile = open("example.txt", "r")
    contents = infile.read()
    infile.close()
    print(contents)

main()
```

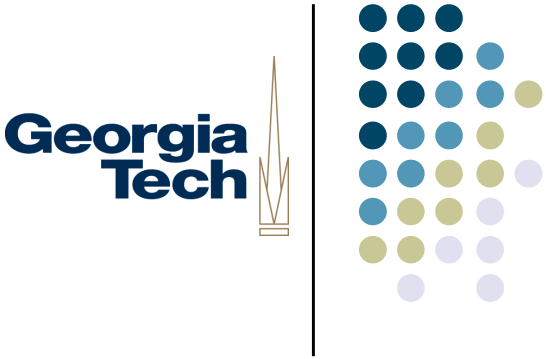
Reading a File



Line-by-line method 1

```
def main():
    infile = open("example.txt", "r")
    for line in infile:
        print(line)
    infile.close()

main()
```

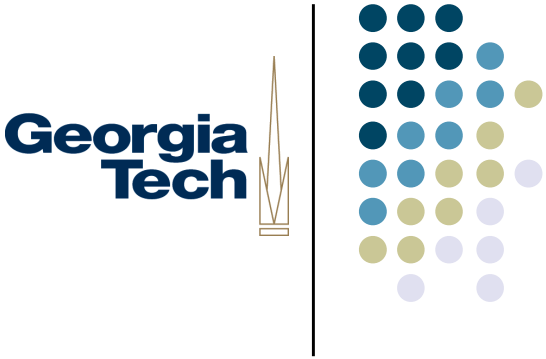
Reading a File



Line-by-line method 2

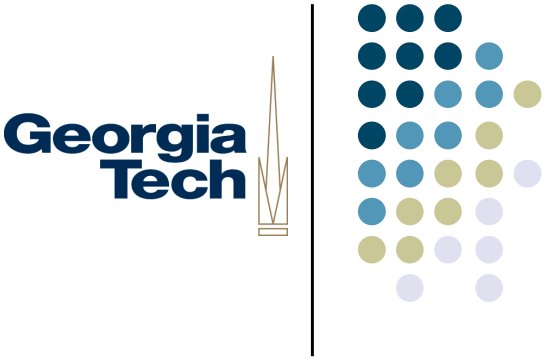
```
def main():
    infile = open("example.txt", "r")
    line = infile.readline()
    while line != '':
        print(line)
        line = infile.readline()
    infile.close()

main()
```

Writing a File



```
def main(somestring):
    outfile = open("example2.txt", "w")
    outfile.write("Line 1\n")
    outfile.write(somestring + '\n')
    outfile.close()

main("Woo hoo")
```

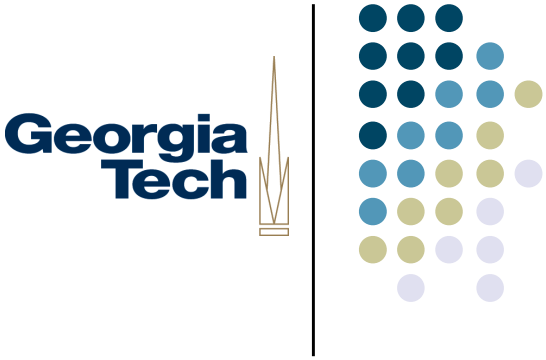
Searching a File



Print lines with "From:" and a gatech address

```
def main():
    infile = open("mailfile.txt", "r")
    for line in infile:
        line = line.lstrip()
        if not line.startswith("From:"):
            continue
        if line.find("@gatech.edu") != -1:
            print(line)
        elif line.find("@cc.gatech.edu") != -1:
            print(line)
    infile.close()

main()
```

Counting Things

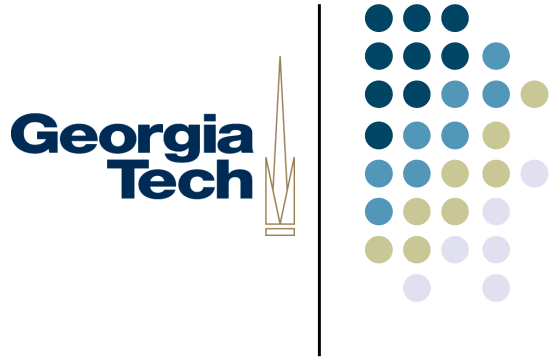


Count the number of characters, words, and lines in a file

```
def counter(filename):
    infile = open(filename, "r")
    data = infile.read()
    return len(data), len(data.split()), len(data.splitlines())
```

Not a good idea if the file is big

Counting Things



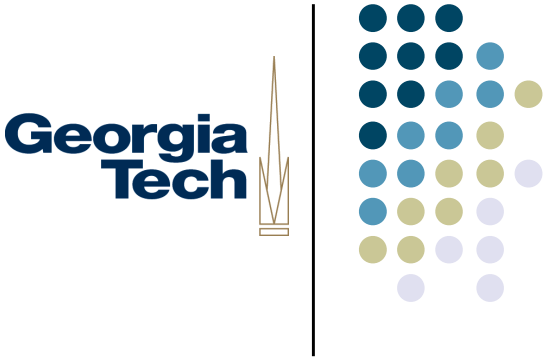
Count the number of characters, words, and lines in a file (OK if the file is big)

```
def counter(filename):
    infile = open(filename, "r")
    num_chars, num_words, num_lines = 0, 0, 0
    for line in infile:
        num_chars += len(line)
        num_words += len(line.split())
        num_lines += 1
    return num_chars, num_words, num_lines
```

Exceptions

- Error that occurs while a program is running, causing execution to halt
- Want to have some way of anticipating them in key "risky" spots in program, then recovering and continuing on if a suspected problem actually does arise

Example

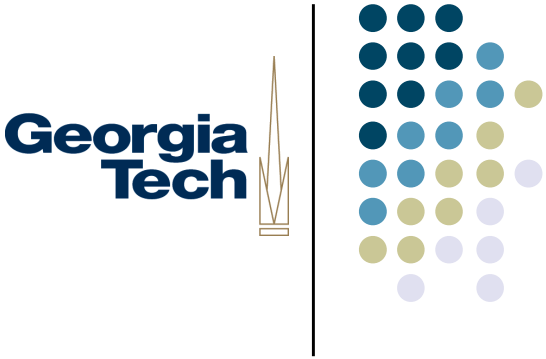


```
def main():
    num1 = eval(input('Enter a number: '))
    num2 = eval(input("Enter another number: '))
    result = num1/num2
    print(num1,"/",num2, " is ",result)

main()
```

How handle this?

if-then-else

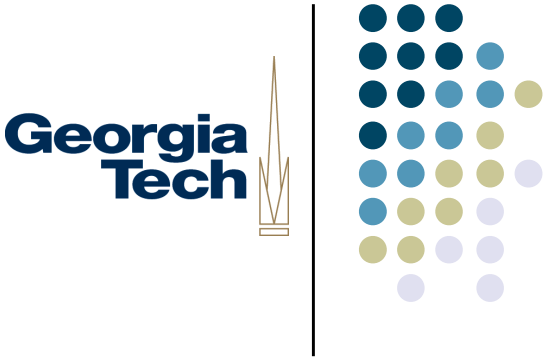
Another Example



```
def main():
    filename = input('Enter a filename: ')
    infile = filename.open(filename, 'r')
    contents = infile.read()
    print(contents)
    infile.close()


main()
```

What's the potential problem?

Handling Exceptions



- Use an exception handler
- Embed "risky" code in a special block
  - `try`
- Tell the system what to do in case a problem occurs
  - `except`

Details

```
try:
    stat1
    stat2
    …
except ExceptName:
    stat1
    stat2
    …
```

Semantics

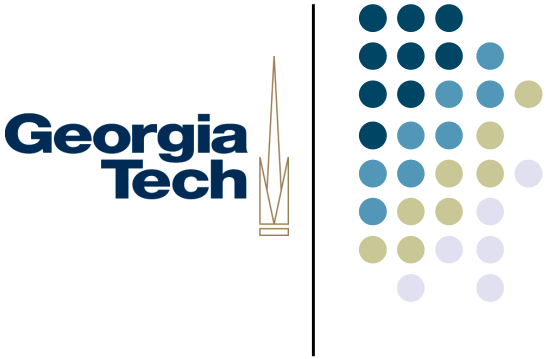
- If not exception occurs in try block, then resume execution after all statements in except block
- If a statement in the try block generates an exception specified by ExceptName, then the statements in the except block are executed. After they're done, execution goes to following code.
- If code in try block generates an exception not specified by the named exception, then the program halts with an error message

Sample ExceptName: `IOError`, `ValueError`, …
OK to have `except:` with no ExceptName (catch all)

Example

```
def main():
    total = 0.0

    try:
        infile = open('sales_data.txt', 'r')
        for line in infile:
            amount = float(line)
            total += amount
        infile.close()
        print('Total: $%.2f' % total)

    except IOError:
        print('An error occurred trying to read the file.')

    except ValueError:
        print('Non-numeric data found in the file.')

    except:
        print('An error occurred.')

    print("Code after try-except")
main()
```

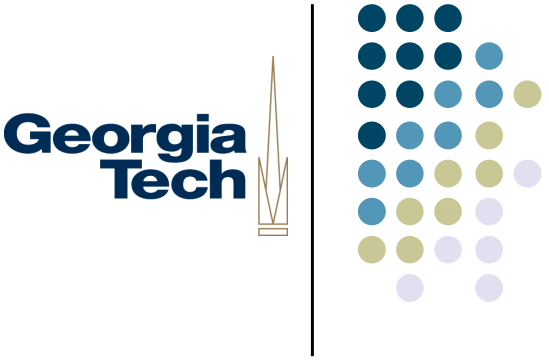
Programming Challenge



Get a filename from the user and read in that file.

Print out all the unique words (tokens) that appear in the file, in alphabetical order, with a count of how often each occurs.

Learning Objectives

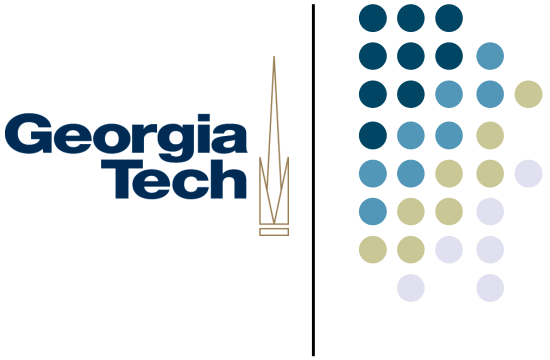


- Concepts about files in Python
- How to open files
- Different ways of reading files
- How to write out files
- How to read then search, count, etc. on files
- Concepts about exceptions and how to use try blocks in your code

Next Time

- Prototyping
  - All about prototyping, different methods, low vs. high, tools, etc.

Reading Summary



- Read article
- Write a one page (front, 2-3 paragraphs summary of paper)
- At bottom of page, write one "interesting quote" taken from paper

- Rudd, et al, "Prototyping Debate", interactions, Jan '96.