Files

We have a file "C:/CS101/planets.txt" on our hard disk, with this contents:

```
Mercury
Venus
Earth
Mars
Jupiter
Saturn
Uranus
Neptune
```

Files

We have a file "C:/CS101/planets.txt" on our hard disk, with this contents:

```
Mercury
Venus
Earth
Mars
Jupiter
Saturn
Uranus
Neptune
```

```
>>> f = open("C:/CS101/planets.txt", "r")
>>> s = f.readline()
>>> s, len(s)
('Mercury\n', 8)
```

Files

We have a file "C:/CS101/planets.txt" on our hard disk, with this contents:

```
Mercury
Venus
Earth
Mars
Jupiter
Saturn
Uranus
Neptune
```

f is a file object (of type <type 'file'>), not the contents of the file.

```
>>> f = open("C:/CS101/planets.txt", "r")
>>> s = f.readline()
>>> s, len(s)
('Mercury\n', 8)
```

Files

We have a file "C:/CS101/planets.txt" on our hard disk, with this contents:

```
Mercury
Venus
Earth
Mars
Jupiter
Saturn
Uranus
Neptune
```

f is a file object (of type <type 'file'>), not the contents of the file.

mode: "r" for reading

```
>>> f = open("C:/CS101/planets.txt", "r")
>>> s = f.readline()
>>> s, len(s)
('Mercury\n', 8)
```

Files

We have a file "C:/CS101/planets.txt" on our hard disk, with this contents:

```
Mercury
Venus
Earth
Mars
Jupiter
Saturn
Uranus
Neptune
```

f is a file object (of type <type 'file'>), not the contents of the file.

mode: "r" for reading

```
>>> f = open("C:/CS101/planets.txt", "r")
>>> s = f.readline()
>>> s, len(s)
('Mercury\n', 8)
```

line separator in file

Reading strings from a file

We typically use `strip()` or at least `rstrip()` for the lines we read from a file to get read of white space.

Reading strings from a file

We typically use `strip()` or at least `rstrip()` for the lines we read from a file to get read of white space.

```
>>> for l in f:
...     s = l.strip()
...     print s,

Venus Earth Mars Jupiter Saturn Uranus Neptune
```

Reading strings from a file

We typically use `strip()` or at least `rstrip()` for the lines we read from a file to get read of white space.

```
>>> for l in f:
...     s = l.strip()
...     print s,
```

```
Venus Earth Mars Jupiter Saturn Uranus Neptune
```

`for`-loop with a file object calls `readline()` automatically for each element, and stops after reading the last line.

Reading strings from a file

We typically use `strip()` or at least `rstrip()` for the lines we read from a file to get read of white space.

```
>>> for l in f:
...     s = l.strip()
...     print s,
```

```
Venus Earth Mars Jupiter Saturn Uranus Neptune
```

`for`-loop with a file object calls `readline()` automatically for each element, and stops after reading the last line.

Call `f.close()` when finished with the file object.

Reading a file

A typical program for reading the contents of an entire file and storing it in a list:

```
planets = []

f = open("C:/CS101/planets.txt", "r")
for line in f:
  planets.append(line.strip())
f.close()

print planets
```

Reading a file

A typical program for reading the contents of an entire file and storing it in a list:

```
planets = []

f = open("C:/CS101/planets.txt", "r")
for line in f:
  planets.append(line.strip())
f.close()

print planets
```

In fact file objects provide a method to do this (but then you get all the white space):

```
planets = f.readlines()
```

Finding earth

We want to find the line in the file containing earth:

```
f = open("C:/CS101/planets.txt", "r")
current = 0
earth = 0
for line in f:
  current += 1
  planet = line.strip().lower()
  if planet == "earth":
    earth = current

print "Earth is planet #%d" % earth
```

Finding earth

We want to find the line in the file containing earth:

```
f = open("C:/CS101/planets.txt", "r")
current = 0
earth = 0
for line in f:
  current += 1
  planet = line.strip().lower()
  if planet == "earth":
    earth = current

print "Earth is planet #%d" % earth
```

The program reads the entire file, even if earth is right at the beginning. After having found earth, there is no need to continue the loop.

Finding earth faster

The keyword break terminates the current loop:

```
f = open("C:/CS101/planets.txt", "r")
earth = 0
for line in f:
  earth += 1
  planet = line.strip().lower()
  if planet == "earth":
    break

print "Earth is planet #%d" % earth
```

Finding earth faster

The keyword break terminates the current loop:

```
f = open("C:/CS101/planets.txt", "r")
earth = 0
for line in f:
  earth += 1
  planet = line.strip().lower()
  if planet == "earth":
    break

print "Earth is planet #%d" % earth
```

break breaks out of the innermost loop only:

```
>>> for x in range(10):
...   for y in range(10):
...     print y,
...     if y == 5: break
```

Commented planets

Some data files contain useful comments, let's say starting with a # sign.

Commented planets

Some data files contain useful comments, let's say starting with a # sign.

```
f = open("C:/CS101/planetsc.txt", "r")
earth = 0
for line in f:
  planet = line.strip().lower()
  if planet[0] == "#":
    continue
  earth += 1
  if planet == "earth":
    break

print "Earth is planet #%d" % earth
```

Commented planets

Some data files contain useful comments, let's say starting with a # sign.

```
f = open("C:/CS101/planetsc.txt", "r")
earth = 0
for line in f:
  planet = line.strip().lower()
  if planet[0] == "#":
    continue
  earth += 1
  if planet == "earth":
    break

print "Earth is planet #%d" % earth
```

`continue` makes the loop move to the next element immediately

A long file

Let's do some word games. We use a file `words.txt` with 113809 English words (http://icon.shef.ac.uk/Moby/).

A long file

Let's do some word games. We use a file `words.txt` with 113809 English words (http://icon.shef.ac.uk/Moby/).

Let's print all English words longer than 18 letters:

```
f = open("C:/CS101/words.txt", "r")

for line in f:
  word = line.strip()
  if len(word) > 18:
    print word

f.close()
```

Word games

Count all the words without the letter 'e':

```
f = open("C:/CS101/words.txt", "r")

count = 0
for line in f:
  word = line.strip()
  if not "e" in word:
    count += 1

print "%d words have no 'e'" % count
f.close()
```

Abecedarian words

Let's find all words whose letters are sorted:

```
def is_abecedarian(word):
  for i in range(1,len(word)):
    if word[i-1] > word[i]:
      return False
  return True

f = open("C:/CS101/words.txt", "r")

for line in f:
  word = line.strip()
  if is_abecedarian(word):
    print word

f.close()
```

Three double letters in a row?

Is there a word that has three double letters in a row?

Committee and Mississippi are close...

Three double letters in a row?

Is there a word that has three double letters in a row?
Committee and Mississippi are close...

```
def three_doubles(word):
  s = ""
  for i in range(1,len(word)):
    if word[i-1] == word[i]:
      s = s + "*"
    else:
      s = s + " "
  return "* * *" in s
```

Writing files

We can also create and write to files:

```
f = open("C:/CS101/test.txt", "w")
f.write("CS101 is fantastic\n")
f.close()
```

Writing files

We can also create and write to files:

```
f = open("C:/CS101/test.txt", "w")
f.write("CS101 is fantastic\n")
f.close()
```

Use mode `"w"` to open a file for writing.

Writing files

We can also create and write to files:

```
f = open("C:/CS101/test.txt", "w")
f.write("CS101 is fantastic\n")
f.close()
```

Use mode `"w"` to open a file for writing.

The file object has a method `write(text)` to write to the file. Unlike `print`, this does not start a new line after the `text`, not even a single space. Use `\n` to include a line break.

Writing files

We can also create and write to files:

```
f = open("C:/CS101/test.txt", "w")
f.write("CS101 is fantastic\n")
f.close()
```

Use mode `"w"` to open a file for writing.

The file object has a method `write(text)` to write to the file. Unlike `print`, this does not start a new line after the `text`, not even a single space. Use `\n` to include a line break.

Do not forget to `close()` the file—otherwise, the file contents may be incomplete.

Financial data

Let's exercise with a currency exchange rate data set. We use files `1994.txt` ... `2009.txt` with the KRW-USD exchange rate for every day. (www.oanda.com)

```
2009/05/11              0.00080110
```

Financial data

Let's exercise with a currency exchange rate data set. We use files `1994.txt` ... `2009.txt` with the KRW-USD exchange rate for every day. (www.oanda.com)

```
2009/05/11              0.00080110
```

We first read the entire data set (16 files) into a long list of pairs:

```
[... (20091227, 1154), (20091228, 1154),
(20091229, 1167), (20091230, 1167),
(20091231, 1163)]
```

Financial data

Let's exercise with a currency exchange rate data set. We use files `1994.txt` ... `2009.txt` with the KRW-USD exchange rate for every day. (www.oanda.com)

```
2009/05/11              0.00080110
```

We first read the entire data set (16 files) into a long list of pairs:

```
[... (20091227, 1154), (20091228, 1154),
(20091229, 1167), (20091230, 1167),
(20091231, 1163)]
```

Let's find the maximum, minimum, and average for each year.

Financial data

Let's exercise with a currency exchange rate data set. We use files `1994.txt` ... `2009.txt` with the KRW-USD exchange rate for every day. (www.oanda.com)

```
2009/05/11              0.00080110
```

We first read the entire data set (16 files) into a long list of pairs:

```
[... (20091227, 1154), (20091228, 1154),
(20091229, 1167), (20091230, 1167),
(20091231, 1163)]
```

Let's find the maximum, minimum, and average for each year.

```
Minimum: (19950705, 755)
Maximum: (19971223, 1960)
```

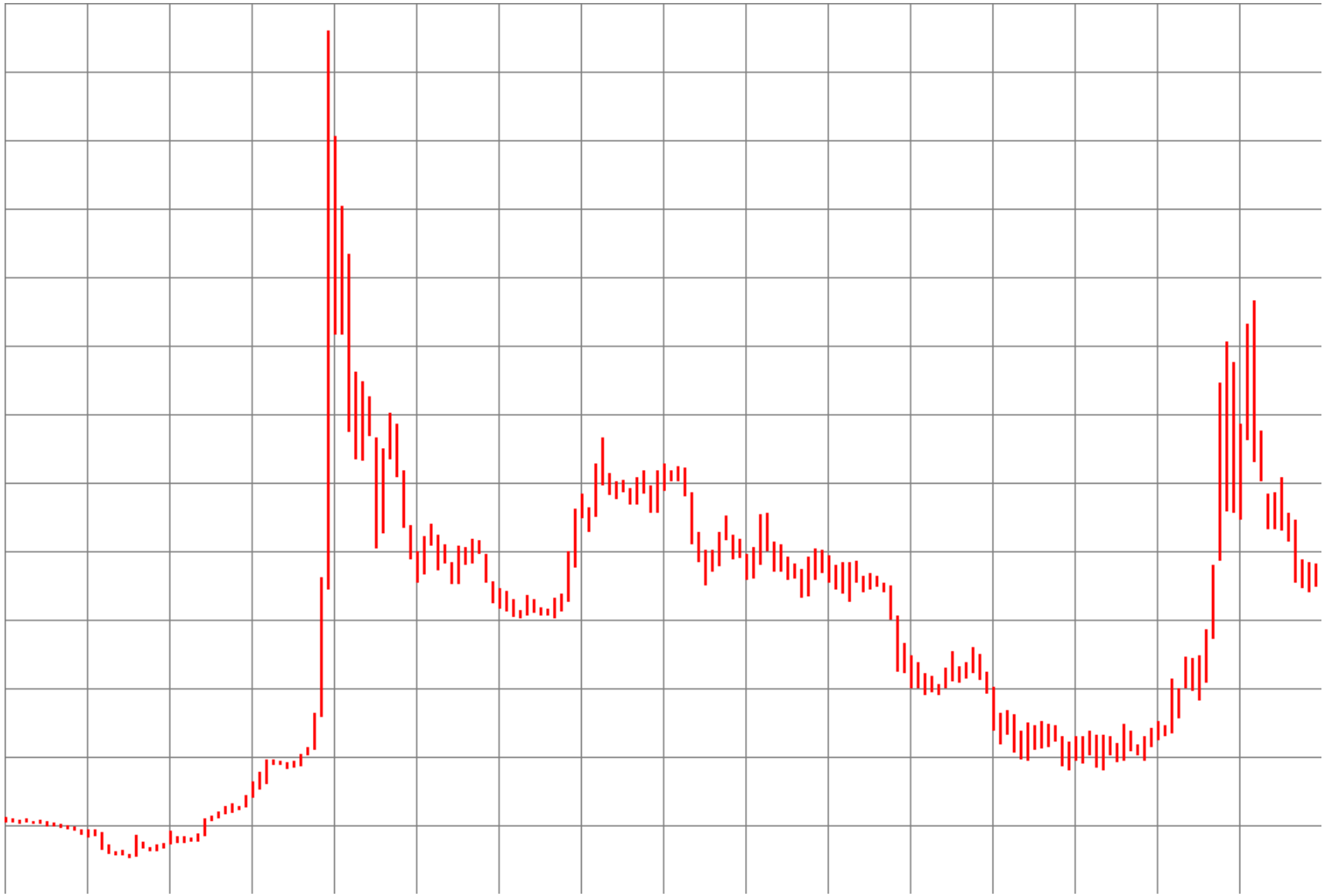
Financial data

Minimum and maximum for every month of a year:

```
def find_minmax(yr):
  minmax = [ (9999, 0) ] * 12
  data = read_year(yr)
  for d, v in data:
    # make month 0 .. 11
    month = (d / 100) % 100 - 1
    minr, maxr = minmax[month]
    if v < minr:
      minr = v
    if v > maxr:
      maxr = v
    minmax[month] = minr, maxr
  return minmax
```

Plot the data

Let's use `cs1media` to create a nice plot of the exchange rate.



Case study: maintaining stock

Let's write software for a vegetable and fruit shop. We keep a file with the current stock and prices. We should be able to sell items, list the current stock, and report all sales done during the day.

Case study: maintaining stock

Let's write software for a vegetable and fruit shop. We keep a file with the current stock and prices. We should be able to sell items, list the current stock, and report all sales done during the day.

File format `stock.txt`:

```
190,apple,900
30,orange,1300
```

Case study: maintaining stock

Let's write software for a vegetable and fruit shop. We keep a file with the current stock and prices. We should be able to sell items, list the current stock, and report all sales done during the day.

File format `stock.txt`:

```
190,apple,900
30,orange,1300
```

Main menu:

```
What would you like to do:
 S) Sell item
 P) Print stock
 R) Report sales
 E) Exit
Enter your choice>
```

Testing

If possible, we test each function after writing it. In the Wing IDE, pressing Run executes your current file. You can then use the functions inside the file from the Python shell. (From the command line, use `python -i script.py`)

Testing

If possible, we test each function after writing it. In the Wing IDE, pressing Run executes your current file. You can then use the functions inside the file from the Python shell. (From the command line, use `python -i script.py`)

Testing `load_stock("stock.txt")`:

```
>>> load_stock("stock.txt")
[(190, 'apple', 900), (30, 'orange', 1300),
(13, 'pineapple', 5500), (60, 'carrot', 600),
(30, 'cucumber', 900), (20, 'egg plant', 1100),
(10, 'zuccini', 1300), (70, 'garlic', 300)]
```

Top-down / bottom-up

We can test code before having written all the functions called:

```
def main():
  stock = load_stock(stock_file_name)
  sales = []
  while True:
    s = show_menu()
    if s == 'e':
      break
    elif s == 's':
      sell(stock, sales)
    elif s == 'p':
      print_stock(stock)
    elif s == 'r':
      print_sales(sales)
```

need these functions to test

needed only when selected