CS101 - Text Processing

Lecture 8

School of Computing
KAIST

Roadmap

Last week we learned

- Data structures
  - String
  - Set
  - Dictionary
- Image processing

Roadmap

Last week we learned

- Data structures
  - String
  - Set
  - Dictionary
- Image processing

This week we will learn

- Files
  - Reading from a file
  - Writing to a file
- break and continue

Files

We have a file `"planets.txt"` on our hard disk with this contents:

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

Files

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

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

```
>>> f = open("planets.txt", "r") # "r" for reading
>>> s = f.readline()
>>> s, len(s)
('Mercury\n', 8)
```

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

\n is a line separator read from the file.

Reading strings from a file

We typically use `strip()` or at least `rstrip()` for the lines we read from a file to get rid of white space. We can also add a parameter `end=" "` to the `print()` statement not to move to a newline after printing each content.

```
>>> for l in f:
...     s = l.strip()
...     print(s, end=" ")
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 rid of white space. We can also add a parameter `end=" "` to the `print()` statement not to move to a newline after printing each content.

```
>>> for l in f:
...     s = l.strip()
...     print(s, end=" ")
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 rid of white space. We can also add a parameter `end=" "` to the `print()` statement not to move to a newline after printing each content.

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

We should 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("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("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("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("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("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("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(2):
...   for y in range(5):
...     print(y, end=" ")
...     if y == 3:
...       break
0 1 2 3 0 1 2 3
```

Commented planets

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

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

Writing files

We can also create and write to files:

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

We use mode `"w"` to open a file for writing.

Writing files

We can also create and write to files:

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

We use mode `"w"` to open a file for writing.

The file object has a method `write(text)` to write to the file.
Unlike `print`, `write()` function 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("./test.txt", "w")
f.write("CS101 is fantastic\n")
f.close()
```

We use mode `"w"` to open a file for writing.

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

We should call `close()` 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.

```
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.