Text Data, File I/O, and Exceptions

- Strings, revisited
- Formatted output
- File Input/Output
- Errors and Exceptions
String representations

A string value is represented as a sequence of characters delimited by quotes

Quotes can be single ('') or double ("")

What if the string includes both characters?

Escape sequence \ or " is used to indicate that a quote is not the string delimiter but is part of the string value

Function print() interprets the escape sequence

Another example:
- \n is an escape sequence that represents a new line
Indexing operator, revisited

The indexing operator returns the character at index \(i\) (as a single character string). The indexing operator can also be used to obtain a slice of a string \(s[i:j]\) : the slice of \(s\) starting at index \(i\) and ending before index \(j\). 

\[
\begin{align*}
\text{s} & \quad = \quad 'A\ p\ p\ l\ e\ e' \\
\text{s[0:2]} & \quad = \quad 'A\ p' \\
\text{s[1:4]} & \quad = \quad 'p\ p\ l' \\
\text{s[2:5]} & \quad = \quad 'p\ l\ e' \\
\text{s[2]} & \quad = \quad 'p\ l\ e' \\
\text{s[4]} & \quad = \quad 'A\ p' \\
\text{s[-3:-1]} & \quad = \quad 'p\ l'
\end{align*}
\]

\[
\begin{align*}
\text{>>> s = 'Apple'} \\
\text{>>> s[0:2]} & \quad = \quad 'Ap' \\
\text{>>> s[1:4]} & \quad = \quad 'ppl' \\
\text{>>> s[2:5]} & \quad = \quad 'ple' \\
\text{>>> s[2]} & \quad = \quad 'ple' \\
\text{>>> s[2:]} & \quad = \quad 'p\ l\ e' \\
\text{>>> s[3:]} & \quad = \quad 'p\ l\ e' \\
\text{>>> s[-3:-1]} & \quad = \quad 'p\ l'
\end{align*}
\]
Exercise

The indexing operator can also be used to obtain slices of a list as well.
Let list \texttt{lst} refer to list

\[
['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h']
\]

Write Python expressions using list \texttt{lst} and the indexing operator that evaluate to:

a) \['a', 'b', 'c', 'd'\]
b) \['d', 'e', 'f'\]
c) \['d'\]
d) \['f', 'g'\]
e) \['d', 'e', 'f', 'g', 'h'\]
f) \['f', 'g', 'h'\]
String methods

Strings are immutable; none of the string methods modify string \texttt{link}

<table>
<thead>
<tr>
<th>Usage</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{s.capitalize()}</td>
<td>&gt;&gt;&gt; link = '<a href="http://www.main.com/smith/index.html">http://www.main.com/smith/index.html</a>'</td>
</tr>
<tr>
<td></td>
<td>&gt;&gt;&gt; link[:4]</td>
</tr>
<tr>
<td></td>
<td>'http'</td>
</tr>
<tr>
<td></td>
<td>&gt;&gt;&gt; link[:4].upper()</td>
</tr>
<tr>
<td></td>
<td>'HTTP'</td>
</tr>
<tr>
<td>\texttt{s.count(target)}</td>
<td>&gt;&gt;&gt; link.find('smith')</td>
</tr>
<tr>
<td></td>
<td>20</td>
</tr>
<tr>
<td>\texttt{s.find(target)}</td>
<td>&gt;&gt;&gt; link[20:25]</td>
</tr>
<tr>
<td></td>
<td>'smith'</td>
</tr>
<tr>
<td></td>
<td>&gt;&gt;&gt; link[20:25].capitalize()</td>
</tr>
<tr>
<td></td>
<td>'Smith'</td>
</tr>
<tr>
<td>\texttt{s.lower()}</td>
<td>&gt;&gt;&gt; link.replace('smith', 'jones')</td>
</tr>
<tr>
<td></td>
<td>'<a href="http://www.main.com/jones/index.html">http://www.main.com/jones/index.html</a>'</td>
</tr>
<tr>
<td>\texttt{s.replace(old, new)}</td>
<td>&gt;&gt;&gt; link</td>
</tr>
<tr>
<td></td>
<td>&gt;&gt;&gt; link.replace('smith', 'jones')</td>
</tr>
<tr>
<td></td>
<td>&gt;&gt;&gt; new = link.replace('smith', 'jones')</td>
</tr>
<tr>
<td></td>
<td>&gt;&gt;&gt; new</td>
</tr>
<tr>
<td></td>
<td>'<a href="http://www.main.com/jones/index.html">http://www.main.com/jones/index.html</a>'</td>
</tr>
<tr>
<td>\texttt{s.split(sep)}</td>
<td>&gt;&gt;&gt; link.count('/')</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>&gt;&gt;&gt; link.split('/')</td>
</tr>
<tr>
<td></td>
<td>['http:', '', 'www.main.com', 'smith', 'index.html']</td>
</tr>
</tbody>
</table>
String events describes the schedule of 4 events spread across 3 days

Write expressions that compute:

a) the number of events on 9/14

b) the index of the substring describing the 1st event on 9/14

c) the index just past the substring describing the last event on 9/14

d) the list of substrings describing the events on 9/14

>>> events.count('9/14')
2
>>> events.find('9/14')
13
>>> events.find('9/15')
40
>>> events[13:40]
'9/14 11:15 AM\n9/14 1:00 PM\n'
>>> lst = events[13:40].strip().split('\n')
>>> lst
['9/14 11:15 AM', '9/14 1:00 PM']
>>>
### String methods

**Suppose we need to pick up the date and time components of string `event`**

Punctuation makes it difficult to use method `split()`

```python
>>> event = "Tuesday, Feb 29, 2012 -- 3:35 PM"
>>> table = str.maketrans((':',-,',', 3*' ')
>>> event.translate(table)
'Tuesday  Feb 29  2012    3 35 PM'
>>> event.translate(table).split()
['Tuesday', 'Feb', '29', '2012', '3', '35', 'PM']
```

**Solution:** replace punctuation with blank spaces

<table>
<thead>
<tr>
<th>Usage</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>str.maketrans(old, new)</code></td>
<td>returns a table mapping characters in string <code>old</code> to characters in string <code>new</code></td>
</tr>
<tr>
<td><code>s.translate(table)</code></td>
<td>returns a copy of <code>s</code> in which the original characters are replaced using the mapping described by <code>table</code></td>
</tr>
</tbody>
</table>
Exercise

Read a line of text

Use translation table to: replace each ' ' with '\n', and capitalize every vowel (and then print)
Files and the file system

The file system is the Operating System (OS) component that organizes files and provides a way to create, access, and modify files.

Files are organized into a tree structure:
- folders (or directories)
- regular files

While every file and folder has a name, it is the file **pathname** that identifies the file.

**Absolute pathnames**
- `/var/poem.txt`
- `messi/poem.txt`
- `/Users/messi/poem.txt`
- `/Users/messi/image.jpg`
- `/Applications/Mail.app`

**Relative pathnames** (relative to current working directory `Users`)
- `messi/poem.txt`
- `messi/image.jpg`
Opening and closing a file

Processing a file consists of:
1. Opening the file
2. Reading from and/or writing to the file
3. Closing the file

Built-in function `open()` is used to open a file
- The first input argument is the file pathname, whether absolute or relative with respect to the current working directory
- The second (optional) argument is the file mode
- Returns a “file” object

File mode 'r' is used to open a file for reading (rather than, say, writing)

A “file” object is of a type that supports several “file” methods, including method `close()` that closes the file

```python
>>> infile = open('sample.txt')
Traceback (most recent call last):
  File "<pyshell#50>", line 1, in <module>
    infile = open('sample.txt')
IOError: [Errno 2] No such file or directory: 'sample.txt'
>>> infile = open('example.txt', 'r')
>>> infile.close()
```
Open file mode

The file mode defines how the file will be accessed

<table>
<thead>
<tr>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>Reading (default)</td>
</tr>
<tr>
<td>w</td>
<td>Writing (if file exists, content is wiped)</td>
</tr>
<tr>
<td>a</td>
<td>Append (if file exists, writes are appended)</td>
</tr>
<tr>
<td>r+</td>
<td>Reading and Writing</td>
</tr>
<tr>
<td>t</td>
<td>Text (default)</td>
</tr>
<tr>
<td>b</td>
<td>Binary</td>
</tr>
</tbody>
</table>

These are all equivalent

```python
>>> infile = open('example.txt', 'rt')
>>> infile = open('example.txt', 'r')
>>> infile = open('example.txt', 't')
>>> infile = open('example.txt')
```
File methods

There are several “file” types; they all support similar “file” methods
- Methods `read()` and `readline()` return the characters read as a string
- Methods `readlines()` returns the characters read as a list of lines
- Method `write()` returns the number of characters written

<table>
<thead>
<tr>
<th>Usage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>infile.read(n)</code></td>
<td>Read n characters starting from cursor; if fewer than n characters remain, read until the end of file</td>
</tr>
<tr>
<td><code>infile.read()</code></td>
<td>Read starting from cursor up to the end of the file</td>
</tr>
<tr>
<td><code>infile.readline()</code></td>
<td>Read starting from cursor up to, and including, the end of line character</td>
</tr>
<tr>
<td><code>infile.readlines()</code></td>
<td>Read starting from cursor up to the end of the file and return list of lines</td>
</tr>
<tr>
<td><code>outfile.write(s)</code></td>
<td>Write string s to file outfile starting from cursor</td>
</tr>
<tr>
<td><code>infile.close()</code></td>
<td>Close file infile</td>
</tr>
</tbody>
</table>
Reading a file

The 3 lines in this file end with the new line character.\n
There is a blank line above this line.\n
When the file is opened, a cursor is associated with the opened file

The initial position of the cursor is:
• at the beginning of the file, if file mode is r
• at the end of the file, if file mode is a or w

>>> infile = open('example.txt')
>>>infile.read(1)
'T'
>>> infile.read(5)
'he 3 '
>>> infile.readline()
'lines in this file end with the new line character.\n'
>>> infile.read()
'\nThere is a blank line above this line.\n'
>>> infile.close()

example.txt
Patterns for reading a text file

Common patterns for reading a file:

1. Read the file content into a string
2. Read the file content into a list of words
3. Read the file content into a list of lines

Example:

```python
def numLines(filename):
    'returns the number of lines in file filename'

    infile = open(filename, 'r')
    lineList = infile.readlines()
    infile.close()

    return len(lineList)
```

```python
def numWords(filename):
    'returns the number of words in file filename'

    infile = open(filename, 'r')
    content = infile.read()
    infile.close()

    wordList = content.split()

    return len(wordList)
```

```python
def numChars(filename):
    'returns the number of characters in file filename'

    infile = open(filename, 'r')
    content = infile.read()
    infile.close()

    return len(content)
```
Writing to a text file

```python
>>> outfile = open('test.txt', 'w')
>>> outfile.write('T')
1
>>> outfile.write('his is the first line.')
22
>>> outfile.write(' Still the first line...
')
25
>>> outfile.write('Now we are in the second line.
')
31
>>> outfile.write('Non string value like 5 must be converted first.
')
49
>>> outfile.write('Non string value like {} must be converted first.
'.format(5))
49
>>> outfile.close()
```
Built-in function `print()`, revisited

Function `print()` takes 0 or more arguments and prints them in the shell.

```
>>> prod = 'morels'
>>> cost = 139
>>> wght = 1/2
>>> total = cost * wght
>>> print(prod, cost, wght, total)
morels 139 0.5 69.5
>>> print(prod, cost, wght, total, sep='; ')
morels; 139; 0.5; 69.5
>>> print(prod, cost, wght, total, sep='::: ')
morels:::139:::0.5:::69.5
>>> 
```

A blank space separator is printed between the arguments.

The `sep` argument allows for customized separators.
Built-in function print(), revisited

Function `print()` prints, by default, a newline character after printing its arguments.

```python
>>> pets = ['boa', 'cat', 'dog']
>>> for pet in pets:
    print(pet)
boa
cat
dog

>>> for pet in pets:
    print(pet, end=', ')
boa, cat, dog,

>>> for pet in pets:
    print(pet, end='!!! ')
boa!!! cat!!! dog!!!
```

The `end` argument allows for customized end characters.
General output formatting

Suppose we have

```python
>>> day = 'Wednesday'
>>> month = 'March'
>>> weekday = 'Wednesday'
>>> month = 'March'
>>> day = 10
>>> year = 2010
>>> year = 2012
>>> hour = 11
>>> minute = 45
>>> second = 33
>>> print(hour+':'+minute+':'+second)
Traceback (most recent call last):
  File "<pyshell#113>", line 1, in <module>
    print(hour+':'+minute+':'+second)
TypeError: unsupported operand type(s) for +: 'int' and 'str'
>>> print(str(hour)+':'+str(minute)+':'+str(second))
11:45:33
```

and we want to print

Wednesday, March 10, 2010 at 11:45:33
Method format() of class str

```python
>>> day = 'Wednesday'
>>> month = 'March'
>>> weekday = 'Wednesday'
>>> month = 'March'
>>> day = 10
>>> year = 2010
>>> year = 2012
>>> hour = 11
>>> minute = 45
>>> second = 33
>>> print('{}:{}:{}'.format(hour, minute, second))
11:45:33
>>> print('{}, {} {}, {} at {}:{}:{}'.format(weekday, month, day, year, hour, minute, second))
Wednesday, March 10, 2012 at 11:45:33
```
Specifying field width

The `format()` method can be used to line up data in columns.

Numbers are aligned to the right.

```python
>>> for i in range(1, 8):
    print(i, i**2, 2**i)
1  1   2
2  4   4
3  9   8
4 16  16
5 25  32
6 36  64
7 49 128

>>> for i in range(1, 8):
    print('{} {:2} {:3}'.format(i, i**2, 2**i))
1  1   2
2  4   4
3  9   8
4 16  16
5 25  32
6 36  64
7 49 128
```
Specifying field width

The `format()` method can be used to line up data in columns.

```
>>> lst = ['Alan Turing', 'Ken Thompson', 'Vint Cerf']
>>> for name in lst:
    fl = name.split()
    print(fl[0], fl[1])
Alan Turing
Ken Thompson
Vint Cerf
```

Strings are aligned to the left.

```
>>> lst = ['Alan Turing', 'Ken Thompson', 'Vint Cerf']
>>> for name in lst:
    fl = name.split()
    print('{:5} {:10}'.format(fl[0], fl[1]))
Alan  Turing
Ken   Thompson
Vint  Cerf
```

Numbers are aligned to the right.
### Output format type

Inside the curly braces of a placeholder, we can specify the field width, the type of the output, and the decimal precision.

<table>
<thead>
<tr>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>binary</td>
</tr>
<tr>
<td>c</td>
<td>character</td>
</tr>
<tr>
<td>d</td>
<td>decimal</td>
</tr>
<tr>
<td>X</td>
<td>hexadecimal</td>
</tr>
<tr>
<td>e</td>
<td>scientific</td>
</tr>
<tr>
<td>f</td>
<td>fixed-point</td>
</tr>
</tbody>
</table>

```python
>>> n = 10
>>> '{:b}'.format(n)
'1010'
>>> '{:c}'.format(n)
'n'
>>> '{:d}'.format(n)
'10'
>>> '{:X}'.format(n)
'A'
>>> '{:e}'.format(n)
'1.000000e+01'
>>> '{:7.2f}'.format(n)
'  10.00'
```
Types of errors

We saw different types of errors in this chapter

There are basically two types of errors:

• syntax errors
• erroneous state errors

```python
>>> excuse = 'I\'m sick'
SyntaxError: invalid syntax
>>> print(hour+':'+minute+':'+second)
Traceback (most recent call last):
  File "<pyshell#113>" , line 1 , in <module>
    print(hour+':'+minute+':'+second)
TypeError: unsupported operand type(s) for +: 'int' and 'str'
>>> infile = open('sample.txt')
Traceback (most recent call last):
  File "<pyshell#50>" , line 1 , in <module>
    infile = open('sample.txt')
IOError: [Errno 2] No such file or directory: 'sample.txt'
```
Syntax errors

Syntax errors are errors that are due to the incorrect format of a Python statement

- They occur while the statement is being translated to machine language and before it is being executed.

```python
>>> (3+4]
SyntaxError: invalid syntax
>>> if x == 5
SyntaxError: invalid syntax
>>> print 'hello'
SyntaxError: invalid syntax
>>> lst = [4;5;6]
SyntaxError: invalid syntax
>>> for i in range(10):
  print(i)
SyntaxError: expected an indented block
```
Erroneous state errors

The program execution gets into an erroneous state

When an error occurs, an “error” object is created

• This object has a type that is related to the type of error
• The object contains information about the error
• The default behavior is to print this information and interrupt the execution of the statement.

The “error” object is called an exception; the creation of an exception due to an error is called the raising of an exception.
## Exception types

Some of the built-in exception classes:

<table>
<thead>
<tr>
<th>Exception</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>KeyboardInterrupt</td>
<td>Raised when user hits Ctrl-C, the interrupt key</td>
</tr>
<tr>
<td>OverflowError</td>
<td>Raised when a floating-point expression evaluates to a value that is too large</td>
</tr>
<tr>
<td>ZeroDivisionError</td>
<td>Raised when attempting to divide by 0</td>
</tr>
<tr>
<td>IOError</td>
<td>Raised when an I/O operation fails for an I/O-related reason</td>
</tr>
<tr>
<td>IndexError</td>
<td>Raised when a sequence index is outside the range of valid indexes</td>
</tr>
<tr>
<td>NameError</td>
<td>Raised when attempting to evaluate an unassigned identifier (name)</td>
</tr>
<tr>
<td>TypeError</td>
<td>Raised when an operation of function is applied to an object of the wrong type</td>
</tr>
<tr>
<td>ValueError</td>
<td>Raised when operation or function has an argument of the right type but incorrect value</td>
</tr>
</tbody>
</table>