Type Abstraction

• One of the most important advantages of the object-oriented paradigm is the idea of type abstraction, in which the goal is to think about types in terms of their high-level behavior rather than their low-level implementation.

• In computer science, types that are defined by their behavior are called abstract data types or ADTs.

• Python includes several built-in abstract types, and you have already seen a few implementations of abstract types, such as the **Rational** we just discussed.

• We’ll spend the rest of lecture discussing strategies on how to define your own abstract data types.
Remembering Pig Latin

• One of the largest examples we covered while teaching JavaScript strings was a program that translated text from English to Pig Latin. We revisited that same program when we discussed Python’s support for strings.

• Both Pig Latin translators decomposed the problem into two functions: a `toPigLatin` function that divides the input into words and a `wordToPigLatin` function that translates a single word to its Pig Latin equivalent. The first phase of this operation is completely independent of Pig Latin domain.

• It would be useful to have a package that divides input strings into individual units that have integrity as a unit, as words do in English. Since the same idea applies in contexts beyond human languages, computer scientists use the term token to define these units. A library that returns individual tokens from an input source is called a token scanner.
Designing a Token Scanner

- Section 12.2 in the Python reader describes a general library class called `TokenScanner`, which is implemented for several programming languages just as our graphics package is.
- The text also implements a small piece of that library that exports the following methods:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>scanner.setInput(str)</code></td>
<td>Sets the input for this scanner to the specified string or input stream.</td>
</tr>
<tr>
<td><code>scanner.hasMoreTokens()</code></td>
<td>Returns <code>true</code> if more tokens exist, and <code>false</code> at the end of the token stream.</td>
</tr>
<tr>
<td><code>scanner.nextToken()</code></td>
<td>Returns the next token from the token stream, and &quot;&quot; at the end.</td>
</tr>
<tr>
<td><code>scanner.ignoreWhitespace()</code></td>
<td>Tells the scanner to ignore whitespace characters.</td>
</tr>
</tbody>
</table>

- These methods are the only `TokenScanner` methods you need for your next assignment.
A Simple **TokenScanner** Class

```python
# File: tokenscanner.py

"""
This file implements a simple token scanner class.
"""

# A token scanner is an abstract data type that divides
# a string into tokens, which are strings of consecutive
# characters that form logical units. This simplified
# version recognizes two token types:
#
# 1. A string of consecutive letters and digits
# 2. A single character string
```

A Simple `TokenScanner` Class

class TokenScanner:

    def __init__(self, source=""):
        self._source = source
        self._nch = len(source)
        self._cp = 0
        self._ignoreWhitespaceFlag = False

    def setInput(self, source):
        self._source = source
        self._nch = len(source)
        self._cp = 0
A Simple **TokenScanner** Class

```python
def nextToken(self):
    if self._ignoreWhitespaceFlag:
        self._skipWhitespace()
    if self._cp == self._nch:
        return ""
    token = self._source[self._cp]
    self._cp += 1
    if token.isalnum():
        while (self._cp < self._nch and
               self._source[self._cp].isalnum()):
            token += self._source[self._cp]
            self._cp += 1
    return token

def hasMoreTokens(self):
    if self._ignoreWhitespaceFlag:
        self._skipWhitespace()
    return self._cp < self._nch
```
A Simple **TokenScanner** Class

```python
def ignoreWhitespace(self):
    self._ignoreWhitespaceFlag = True

# Private methods

def _skipWhitespace(self):
    while (self._cp < self._nch and
           self._source[self._cp].isspace()):
        self._cp += 1
```
Using **TokenScanner** in **PigLatin**

```python
# File: PigLatin.py

from tokenscanner import PigLatin

def toPigLatin(line):
    result = ""
    scanner = TokenScanner(line)
    while scanner.hasMoreTokens():
        token = scanner.nextToken()
        if token.isalpha():
            token = wordToPigLatin(token)
        result += token
    return result
```
The End