Objects and Abstraction

Jerry Cain
CS 106AX
November 7, 2022
slides leveraged from those constructed by Eric Roberts

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 last time.
• 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 into 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:

```python
scanner.setInput(str)  # Sets the input for this scanner to the specified string or input stream.
scanner.hasMoreTokens()  # Returns True if more tokens exist, and False at the end of the token stream.
scanner.nextToken()  # Returns the next token from the token stream, and """" at the end.
scanner.ignoreWhitespace()  # Tells the scanner to ignore whitespace characters.
```

• These methods are the primary TokenScanner methods you need for your next assignment.

A Simple TokenScanner Class

```python
# File: tokenscanner.py

class TokenScanner:
    def __init__(self, source):
        self._source = source
    def hasMoreTokens(self):
        return len(self._source) > 0
    def nextToken(self):
        return self._source[0]
    def ignoreWhitespace(self):
        self._source = self._source[1:]

scanner = TokenScanner("<your input here>")
```

A Simple TokenScanner Class
A Simple TokenScanner Class

def next_token(self):
    if self._ignore_whitespace_flag:
        self._skip_whitespace()
    if self._cp == self._nch:
        return ""
    token = self._source[self._cp]
    self._cp += 1
    if token.isspace():
        while self._cp < self._nch and
            self._source[self._cp].isspace():
            token = self._source[self._cp]
            self._cp += 1
        return token

def has_more_tokens(self):
    if self._ignore_whitespace_flag:
        self._skip_whitespace()
    return self._cp < self._nch

A Simple TokenScanner Class

def ignore_whitespace(self):
    self._ignore_whitespace_flag = True

# Private methods

def _skip_whitespace(self):
    while (self._cp < self._nch and
        self._source[self._cp].isspace()):
        self._cp += 1

Using TokenScanner in PigLatin

# File PigLatin.py
from tokenscanner import PigLatin

def PigLatin(line):
    result = ""
    scanner = TokenScanner(line)
    while scanner.has_more_tokens():
        token = scanner.next_token()
        if token.isalpha():
            token = wordPigLatin(token)
        result += token
    return result

The End