ECE12: Introduction to Programming
Lecture 22

Rainer Dömer

doemer@uci.edu

The Henry Samueli School of Engineering
Electrical Engineering and Computer Science
University of California, Irvine

Lecture 22: Overview

- String Manipulation
  - String Methods
    - Search and replace text
    - Examples
  - Regular Expressions
    - Introduction
    - Search and replace text
    - Examples

String Manipulation

- String Methods to Search and Replace Text
  - **`find(substring, start, end)`**
    - returns position where **`substring`** is found in the string, or -1, if **`substring`** is not found
    - **`start`** and **`end`** are optional search indices
      - **`"test string".find("st")`** returns **`2`**
      - **`"test string".find("ST")`** returns **`-1`**
  - **`replace(old, new, max)`**
    - returns the string with all occurrences of **`old`** replaced by **`new`**
    - **`max`** optionally indicates maximum number of replacements
      - **`"test string".replace("st","X")`** returns **`"teX Xring"`**
      - **`"test string".replace("st","X",1)`** returns **`"teX string"`**

String Manipulation

- String Methods to Search and Replace Text
  - Example
    - Text
      - **“The quick brown fox jumps over the lazy dog.”**
    - Tasks
      - Find the **“dog”**
      - Find the **“cat”**
      - Replace the **“dog”** by a **“cat”**
    - Interactive Python program

```
% python
>>> text = "The quick brown fox jumps over the lazy dog."
>>> text.find("dog")
40
>>> text.find("cat")
-1
>>> text.replace("dog", "cat")
'The quick brown fox jumps over the lazy cat.'
```

String Manipulation

- String Methods to Search and Replace Text
  - Example
    - Text
      - **`"The quick brown fox jumps over the lazy dog."`**
    - Advanced Tasks
      - Find all lower-case words
      - Find all three-letter words
      - Find all three-letter words with an **`"o"`** in the middle
      - Find the first **`"dog"`**, **`"cat"`**, or **`"fox"`**
      - Find any **`"quick" "fox"`**
      - Replace any **`"dog"`**, **`"cat"`**, or **`"fox"`** by an **`"animal"`**
      - Replace any **`"the"`** by an **`"a"`**, ignoring the case
      - etc.

➢ Possible, but difficult and cumbersome with string methods!

String Manipulation

- Regular Expressions
  - Introduction
    - Regular expressions describe a set of strings by use of string patterns
    - Regular expressions are a powerful tool for searching and replacing text
    - The module **`re`** provides regular expressions in Python
  - Example
    - The pattern **`"ab*c"`** matches all strings that
      - start with **`"a"`**,
      - followed by zero or more **`"b"`**,
      - and end with **`"c"`**
    - such as
      - **`"abc"`**, **`"ac"`**, or **`"abbbbbc"`**
    - but not
      - **`"Abc"`**, **`"aabc"`**, nor **`"cba"`**

String Manipulation

- Regular Expressions
  - Composing Regular Expressions
    - Standard characters
      - **“abc”** matches only **“abc”**
    - Meta characters
      - **“.”** matches any character except newline **“\n”**
      - **“x?”** matches zero or one occurrence of **“x”**
      - **“x+”** matches one or more occurrences of **“x”**
      - **“x*”** matches zero or more occurrences of **“x”**
      - **“^x”** matches **“x”** only at the beginning of the string
      - **“x$”** matches **“x”** only at the end of the string
      - **“x|y”** matches the occurrence of **“x”** or **“y”**
      - **“x{n}”** matches the occurrence of **“x”** **n** times
    - Character classes
      - **“[abc]”** matches the character **“a”**, **“b”**, or **“c”**
      - **“[a-d]”** matches the characters **“a”** through **“d”**
      - **“[^abc]”** matches any character except **“a”**, **“b”**, **“c”**

String Manipulation

- Regular Expressions
  - Composing Regular Expressions
    - Escape sequences (using *raw strings*)
      - **r“\d”** matches any digit (same as **“[0-9]”**)
      - **r“\D”** matches any non-digit (same as **“[^0-9]”**)
      - **r“\s”** matches any white space (same as **“[\n\f\r\t\v]”**)
      - **r“\S”** matches any non-white space (same as **“[^ \n\f\r\t\v]”**)
      - **r“\w”** matches any alphanumeric word (same as **“[a-zA-Z0-9_]”**)
      - **r“\w”** matches any non-alphanumeric word (same as **“[^a-zA-Z0-9_]”**)
      - **r“\\”** matches the backslash character

String Manipulation

- Regular Expressions
  - Examples
    - all lower-case word
      - **r" [a-z]+ "**
    - three letter word
      - **r" [a-zA-Z]{3} "**
    - three letter word with **"o"** in the middle
      - **r" [a-zA-Z]o[a-zA-Z] "**
    - **"dog"**, **"cat"** or **"fox"**
      - **r"dog|cat|fox"**
    - any quick fox
      - **r"quick.*fox"**
    - the word **"the"** in any case
      - **r"[tT][hH][eE]"**

String Manipulation

- Regular Expressions
  - Module **`re`** provides regular expression functions
    - **`search(regexp, string)`**
      - searches for an occurence of **`regexp`** in **`string`**
      - returns a match object, or **`None`** if no match is found
      - the match object provides methods
        - » **`group()`** the substring that matched the pattern
        - » **`start()`** the index where the match starts
        - » **`span()`** a tuple (start, stop) of the match
    - **`match(regexp, string)`**
      - matches **`regexp`** against the beginning of **`string`**
      - returns a match object, or **`None`** if no match is found
    - **`findall(regexp, string)`**
      - returns a list of substrings of **`string`** that match **`regexp`**

String Manipulation

- Regular Expressions
  - Module **`re`** provides regular expression functions
    - **`sub(regexp, replacement, string)`**
      - substitutes all occurrences of **`regexp`** in **`string`** by **`replacement`**
      - returns the new string
    - **`subn(regexp, replacement, string)`**
      - substitutes all occurrences of **`regexp`** in **`string`** by **`replacement`**
      - returns a tuple **`(s,n)`** with the new string **`s`** and the number of substitutions **`n`**
    - **`compile(regexp)`**
      - pre-compiles the **`regexp`** into a search pattern object for better performance
      - returns a pattern object that can be used as **`regexp`** in above functions

String Manipulation

- Regular Expressions
  - Interactive example

```
% python
>>> import re
>>> text = "The quick brown fox jumps over the lazy dog."
>>> re.findall(r" [a-z]+", text)
[' quick',' brown',' fox',' jumps',' over',' the',' lazy',' dog']
>>> re.findall(r" [a-z]{3}[ \.]", text)
[' fox ', ' the ', ' dog.']
>>> re.findall(r" [a-z]o[a-z][ \.]", text)
[' fox ', ' dog.']
>>> re.search(r"dog|cat|fox", text).group()
'fox'
>>> re.search(r"quick.*fox", text).group()
'quick brown fox'
>>> re.sub(r"dog|cat|fox", "animal", text)
'The quick brown animal jumps over the lazy animal.'
>>> re.sub(r"[Tt][Hh][Ee]", "a", text)
'a quick brown fox jumps over a lazy dog.'
```