

# HomeWork #2

## Internet Bots, Spiders, and Emotional Agents

Fall 2001  
Prof. Doug DeGroot <degroot@liacs.nl>

### Homework 2: WebSplit.r Improvement

**Goal:** Develop a Web-Page Splitter (to separate tags from text)

**Due:** Thursday, Sept 27, 2001 (submit via email)

### Description:

Shown below is the original code for a demonstration application from Rebol Technologies. This application is contained in the Rebol library file named *websplit.r*. It demonstrates a method for reading a specified web page and then splitting that page into the set of HTML tags and the text between the tags. Here is the program:

```
REBOL [  
  Title: "Web HTML Tag Extractor"  
  File: %websplit.r  
  Date: 20-May-1999  
  Purpose: "Separate the HTML tags from the body text of a document."  
  Category: [web net text 3]  
]  
  
tags: make block! 100  
text: make string! 8000  
html-code: [  
  copy tag ["<" thru ">"] (append tags tag) |  
  copy txt to "<" (append text txt)  
]  
page: read http://www.rebol.com  
parse page [to "<" some html-code]  
foreach tag tags [print tag]  
print text
```

As is typical of the demonstration programs in the Rebol library, the code is useful only for direct execution. It does not define any functions or data sets for reuse. This is not very useful to us in this class, as we need to build up a set of reusable components from these demo codes that will allow us to later on build complex, sophisticated bots but with speed. Accordingly, I have modified the program just a bit so that it defines a function and then runs the function with a test URL.

```

REBOL [
  Title: "Web HTML Tag Extractor (Revised, DD)"
  File: %websplitDD.r
  Purpose: "Separate the HTML tags from the body text of a document."
]

webSplit: func [theUrl
                /local page tags text html-code-rule
] [
  tags: make block! 100
  text: make string! 8000
  html-code-rule: [
    copy tag ["<" thru ">"] (append tags tag) |
    copy txt to "<" (append text txt)
  ]
  page: read theUrl
  parse page [to "<" some html-code-rule]
  return reduce [tags text]
]

;delete these lines when ready for production use
tags-and-text: webSplit http://www.google.com
foreach tag tags-and-text/1 [print tag]

```

The benefits of this approach are that:

- 1) the results of the program execution are saved in a variable (for further use, if desired), and
- 2) the function can be easily re-run with other URLs, from within the Rebol window (e.g., you can type something

Please try to adopt this method of improvement for each of the library programs you experiment with, although you should certainly adopt whatever specific methods of improvement you feel most valuable to you.

### **Problem:**

Actually, as delivered, the demonstration program does not always work as one would expect. It is advertised as a program that can split a web page's source code into text and tags. In fact, this is really how we are used to thinking of a web page — as some readable text marked up with various HTML tags.

Your assignment, should you choose to accept it, is to determine the ways the program can fail to perform in expected ways and to then fix the program.

1. Run the program on a number of different web pages. Examine the results of execution and look for problems in the output. Check both the *tags* output and the *text* output. If you run

the application on a sufficient number of web pages, you should begin to see one or more semi-obvious problems with this simple program.

2. You might at first be tempted to think that the inclusion of symbols like &nbsp; in the text output are a problem, but this is not necessarily so. Sometimes, these &nbsp; symbols are included to represent an actual space character that is required for proper formatting of the readable text; sometimes they are included only to make certain graphical elements appear properly (e.g., blank cells in tables), and thus they are not expected to be read. If you choose to do something about these symbols, it is up to you. But keep going and look for the “big” problems in the program and fix them.
3. You are encouraged to add whatever cute and clever post-processing of the program’s output that you wish, but your submitted program will be graded mostly upon its ability to properly perform the stated objective of the program. All submitted programs will be executed (by us) against a test-set of web sites to ensure correct functionality.

Enjoy!