**Writing URL classification patterns**

**&**

**Web scraping rules**

**Project**

**Document History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Revision** | **Date** | **Author** | **Notes** |
| 1.0 | 29-Sep-13 | Liran Sigalat | Initial version |
| 1.1 | 14-Oct-13 | Liran Sigalat | Adding the ‘fpattern’ to the URL pattern section.  Unifying the three different cart action into one ‘cart’ action.  Adding the reference to the url\_classification test application. |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table of Contents

[1 Project Goal 4](#_Toc369558880)

[2 Prerequisites 4](#_Toc369558881)

[3 Classifying a Domain 4](#_Toc369558882)

[3.1 Input 4](#_Toc369558883)

[3.2 Deciding which URLs to classify 5](#_Toc369558884)

[3.3 Classifying URLs 6](#_Toc369558885)

[3.3.1 URL classification rule structure 7](#_Toc369558886)

[3.3.2 Action 7](#_Toc369558887)

[3.3.3 URL Patterns 8](#_Toc369558888)

[3.3.3.1 Signals 9](#_Toc369558889)

[3.3.3.2 URL structure 9](#_Toc369558890)

[3.3.3.3 Pattern 10](#_Toc369558891)

[3.3.3.4 QPattern 10](#_Toc369558892)

[3.3.3.5 HPattern 10](#_Toc369558893)

[3.3.3.6 SPatterns 10](#_Toc369558894)

[3.3.3.7 FPatterns 10](#_Toc369558895)

[3.3.4 Norm 10](#_Toc369558896)

[3.4 Output File Structure 11](#_Toc369558897)

[3.4.1 The match(url) function 11](#_Toc369558898)

[3.4.2 The rules definition 11](#_Toc369558899)

[3.4.3 The params definition 12](#_Toc369558900)

[3.4.4 The examples definition 12](#_Toc369558901)

[3.5 Testing your code with the url\_parser application 13](#_Toc369558902)

[3.5.1 Adjusting the Unit Testing Application and running it 13](#_Toc369558903)

[3.5.2 Common problems when running the Unit Testing Application 14](#_Toc369558904)

[3.5.2.1 Example 1: 'XXX: not found in actual' 14](#_Toc369558905)

[3.5.2.2 Example 2: 'rule: expected ip1, found ip2' 14](#_Toc369558906)

[3.5.2.3 Example 3: 'XXX: not found in expected' 14](#_Toc369558907)

[3.6 Testing your code with the url\_classification application 15](#_Toc369558908)

[3.6.1 Adjusting the Application and running it 15](#_Toc369558909)

[3.6.2 The Application’s output 15](#_Toc369558910)

[4 Page Scraping 15](#_Toc369558911)

[4.1 Deciding which rules should have a scraping function 16](#_Toc369558912)

[4.2 The scraping function 16](#_Toc369558913)

[4.2.1 Locating the product names within the page 16](#_Toc369558914)

[4.2.2 Locating the product names within the page’s source code 17](#_Toc369558915)

[4.2.3 Writing the scraping function 18](#_Toc369558916)

[4.3 Output file structure 19](#_Toc369558917)

[4.3.1 The scraping functions 19](#_Toc369558918)

[4.3.2 The parsers definition 19](#_Toc369558919)

[4.3.3 The examples definition 19](#_Toc369558920)

[4.4 Testing your code with the scraper\_tests application 20](#_Toc369558921)

[4.4.1 Adjusting the Unit Testing Application and running it 20](#_Toc369558922)

[4.4.2 Common problems when running the Unit Testing Application 20](#_Toc369558923)

[4.4.2.1 Example 1: ‘AssertionError’ 20](#_Toc369558924)

[4.4.2.2 Example 2: ‘KeyError: 'scrap\_pn'’ 21](#_Toc369558925)

[4.5 Testing your code with the scraper application 21](#_Toc369558926)

[4.5.1 Adjusting the scraper Application and running it 22](#_Toc369558927)

[4.5.2 The run’s result 22](#_Toc369558928)

[5 Summary 23](#_Toc369558929)

# Project Goal

The goal of this project is to classify and categorize domains related to purchasing, such as: Walmart, Sears, Samsung, Apple, etc.

The target is to capture at least 90% of the URLs related to purchasing for each domain, with URL patterns, while the rest of the URLs, which are not relevant to purchasing can be ignored.

In addition, for each URL pattern that doesn’t include a ‘product name’ in the pattern itself a scraping function should be written to capture the ‘product name’ from within the page.

# Prerequisites

* Python version 2.7.5 – <http://www.python.org/download/releases/2.7.5/>.
* Scrapy – The scraper we’re using and all its dependencies:
  + If you’re using a Win64 machine use this: <http://steamforge.net/wiki/index.php/How_to_Install_Scrapy_in_64-bit_Windows_7> (Don’t miss any step and do things exactly as defined in this manual, besides choosing the newer versions for some packages instead of the older ones specified in the manual).
  + Otherwise use: <https://scrapy.readthedocs.org/en/latest/intro/install.html>.
* url\_parser.py – The unit testing application we’re using to test URL pattern rules.
* scraper\_tests.py – The unit testing application we’re using to test scraping rules.
* scraper.py – The scraper application we’re using (for testing as well) which actually retrieves the relevant data from the URL itself.
* {optional} <http://gskinner.com/RegExr/> - This site can be used to quickly test regular expressions (check the global / ignoreCase / dotall checkboxes).
* {optional} <http://videlibri.sourceforge.net/cgi-bin/xidelcgi> - This site can be used to quickly test scraping rules (we’re using the ‘XPath 2.0’ option).

# Classifying a Domain

The following steps should be followed in order to classify a domain properly:

## Input

For each domain that needs to be classified a file would be supplied. The file will contain up to 1 million rows (but usually much less) representing real user traffic to the domain (to both purchase related pages and non-purchase related pages). Each row in the file contains 2 columns separated by a Tab sign. The first column is the URL itself and the second column is the number of occurrences of this URL in our data set. The file will be sorted according to the URL column in a descending order.

See an example below:

|  |  |
| --- | --- |
| http://www.sears.com/4.6-cu-ft-top-load-washer-and/p-026CO54236212B | 1 |
| http://www.sears.com/48-8221-foosball-table/p-00643965000P?prdNo=4&amp;amp;blockNo=4&amp;amp;blockType=G4 | 1 |
| http://www.sears.com/5.1-cu-ft-front-load-washer-and-9.0/p-026CO55946112B?prdNo=1&blockNo=1&blockType=G1 | 4 |
| http://www.sears.com/54inch-foosball-table/p-00601232000P?prdNo=1&amp;amp;blockNo=1&amp;amp;blockType=G1#reviewsWrap | 3 |
| http://www.sears.com/5pc-dining-set-with-storage/p-00856170000P?prdNo=2&amp;amp;blockNo=2&amp;amp;blockType=G2 | 3 |
| http://www.sears.com/9-drawer-red-ball-bearing-griplatch-combo-limited/p-009CO52821012B?&amp;amp;adCell=W21 | 16 |
| http://www.sears.com/?adcell=W1 | 18 |
| http://www.sears.com/?affsrc=1&amp;amp;sid=IAx20050830x001277&amp;amp;PID=3211374&amp;amp;aff=Y | 7 |

## Deciding which URLs to classify

As stated in the project goal the idea is to classify only purchase related URLs and to capture at least 90% of the purchase related traffic (there are some exceptions to this definition that would be described later on in details). The best way to analyze the data and to decide on which URL patterns you should focus is by loading all the URLs into a SQL database and performing a simple query that would count the number of occurrences per hostname in the domain.

I created the following table (MySQL v5.6):

CREATE TABLE urls.walmart (

`URL` varchar(1000) NOT NULL,

`Count` int(10) unsigned NOT NULL

) ENGINE=InnoDB DEFAULT ROW\_FORMAT=COMPRESSED KEY\_BLOCK\_SIZE=2;

Ran the following query and received the below results:

SELECT SUBSTRING\_INDEX(URL, '/', 3) AS Domain, SUM(Count) AS NumOfOccurrences

FROM urls.walmart

GROUP BY Domain

ORDER BY NumOfOccurrences DESC;

|  |  |  |  |
| --- | --- | --- | --- |
| **Total** | **2,039,484** | **100.00%** | **Explanation - What to focus on?** |
| http://www.walmart.com | 1,598,746 | 78.39% | Major focus on this since this is the vast majority of the traffic |
| https://www.walmart.com | 213,306 | 10.46% | Focus on this as well since a lot of the traffic is here too |
| http://photos2.walmart.com | 40,663 | 1.99% | Check these 2 along with the other blue ones, since together they're 5% of the traffic, something that can't be ignored |
| http://photos1.walmart.com | 34,737 | 1.70% |
| http://careers.walmart.com | 33,231 | 1.63% | Can obviously be ignored, since careers and jobs aren't related to purchasing activity |
| http://jobs.walmart.com | 23,654 | 1.16% |
| http://photos.walmart.com | 21,287 | 1.04% | See above comment |
| http://help.walmart.com | 8,986 | 0.44% | A low amount of traffic therefore can be ignored |
| http://see.walmart.com | 8,383 | 0.41% |
| https://photos.walmart.com | 6,604 | 0.32% | See above comment |
| http://wireless.walmart.com | 5,567 | 0.27% | A low amount of traffic therefore can be ignored |
| http://tracking01.walmart.com | 5,548 | 0.27% |
| http://localad.walmart.com | 5,308 | 0.26% |
| http://corporate.walmart.com | 4,496 | 0.22% |
| http://instoresnow.walmart.com | 3,293 | 0.16% | Although the portion of the traffic to these 2 hostnames is very low, both coupons and in-store-now might be related to special purchasing activity (more explanation to follow) |
| http://coupons.walmart.com | 2,816 | 0.14% |
| https://avocado.walmart.com | 2,435 | 0.12% | A low amount of traffic therefore can be ignored |
| https://survey.walmart.com | 2,248 | 0.11% |
| http://reviews.walmart.com | 1,614 | 0.08% |
| http://soundcheck.walmart.com | 1,468 | 0.07% |
| http://foundation.walmart.com | 1,122 | 0.06% |
| … | … | … | … |

In addition I ran the following query to better understand the breakdown of the hostname ‘http://www.walmart.com’ which is the vast majority of the traffic and received the below results:

SELECT SUBSTRING\_INDEX(URL, '/', 3) AS Domain, SUM(Count) AS NumOfOccurrences

FROM urls.walmart

WHERE URL LIKE 'http://www.walmart.com%'

GROUP BY Domain

ORDER BY NumOfOccurrences DESC;

|  |  |  |  |
| --- | --- | --- | --- |
| **Total** | **1,578,151** | **100.00%** | **Explanation - What to focus on?** |
| http://www.walmart.com/search | 498,595 | 31.59% | Focus on all these patterns since each one represents a significant percentage of the traffic to the domain |
| http://www.walmart.com/ip | 313,373 | 19.86% |
| http://www.walmart.com/browse | 300,305 | 19.03% |
| http://www.walmart.com/cp | 243,529 | 15.43% |
| http://www.walmart.com/ | 152,841 | 9.68% |
| http://www.walmart.com/storeLocator | 15,862 | 1.01% | This events is related to special purchasing activity (more explanation to follow) |
| http://www.walmart.com/cservice | 8,877 | 0.56% |  |
| http://www.walmart.com/catalog | 6,097 | 0.39% |  |
| http://www.walmart.com/c | 4,890 | 0.31% |  |
| … | … | … | … |

## Classifying URLs

Once deciding on the URLs that should be classified, the next step would be to look at the actual URLs and identify patterns within them. It is also important to actually open several URLs from each type in order to understand the exact content of the pages you’re trying to classify.

The sections below explain in detail the structure of rules that need to be written for each identified pattern as well as how to test the code and the expected output.

### URL classification rule structure

Below is an example for a URL rule written to match many of the URLs for the Walmart domain:

{'name' : 'ip1',

'pattern' : 'ip/(?P<pn>.\*?)/(?P<pid>.\*?)([#?]|$)',

'action' : 'product',

'norm' : 'http://www.walmart.com/ip/{pn}/{pid}'}

* name – The rule name (in this case ‘ip’ since this is the most distinguishable part of the pattern and the ‘1’ was added since there’re several rules that contain ‘ip’ within them.
* pattern – The regular expression capturing all URLs from this pattern. This is an example of a URL matching the pattern: <http://www.walmart.com/ip/Dyson-DC24-Animal-Ultra-Lightweight-Bagless-Upright-Vacuum-Cleaner/15173329>, while 'pn': 'Dyson-DC24-Animal-Ultra-Lightweight-Bagless-Upright-Vacuum-Cleaner' and 'pid': '15173329'. Full explanation on the meaning of ‘pn’ and ‘pid’ below.
* action – The action expressed in all URLs matching this pattern, in case a person was looking at this page. Full explanation on all possible actions below.
* norm - This is the normalized URL. The shortest possible subset of the original URL that would lead to the same page. Full explanation to follow.

### Action

The table below details all the actions that can be reflected in pages. Each URL pattern is assigned a single action (in the above example it was ‘product’).

|  |  |  |
| --- | --- | --- |
| **Action Name** | **Action Description** | **URL Example** |
| List | Several products are presented in the page | <http://www.walmart.com/browse/electronics/scanners/3944_3951_37807_4439/> |
| Product | A single product is presented in the page | <http://accessories.us.dell.com/sna/productdetail.aspx?sku=A6280934> |
| Compare | Usually a page in which the user chose to compare multiple products | <http://store.apple.com/us/iphone/family/iphone/compare> |
| Review | Usually a page containing customer reviews and ratings for products | <http://store.apple.com/us/reviews/H7890ZM/A/smart-baby-monitor-by-withings> |
| cart (\*) | Adding an item to the shopping cart or viewing the shopping cart (as in some cases the same URL is used for both actions) | <http://store.apple.com/us/cart/shared_cart/5041e79b-0169-4220-b02e-001018ddf8b0> |
| Checkout (\*) | The user is in the checkout process (if this process ends the user purchases a product) | <https://www.walmart.com/subflow/CheckoutFlowContext/1000193623/webflow/co_regular> |
| Purchase (\*) | Completion of the checkout process, a purchased product (A Thank-You page in many domains) | https://secure1.store.apple.com/us/checkout/thankyou |
| Store\_locator (\*) | Store Locator - Usually pages in which a person looks for an online store close to his location | <http://www.apple.com/retail/locator/index.html?q=Nashville,%20TN> |
| Irrelevant | A URL under a the domain that is irrelevant to purchasing | <http://www.apple.com/jobs/us/> |

Several important notes:

* The vast majority of the actions on most of the classified domains are either ‘list’ or ‘product’. ‘compare’ and ‘review’ are pretty rare and in most cases capture less than 1% of the total traffic to the domain, therefore can be ignored in case they’re indeed capturing a low volume of the traffic.
* ‘irrelevant’ – In some domains that are not only related to purchasing activity (such as dell), the non-purchasing related traffic to the domain can be a major part of the traffic. In any case we’re interested in mapping the majority of the irrelevant URL patterns.
* Special actions – Special actions are all the actions with a green background, also marked with (\*). All those special actions are related to special purchasing activity showing that the person viewing this page is doing something that is beyond viewing a list of products or a single product, meaning the person is actually interested in buying the product or has already bought it in the case of the ‘purchase’ action). It is very important to capture **ALL** the patterns relating to these special actions, even if they present only a very low percentage of the entire traffic to the domain (which is usually the case).

In order to better capture these special actions, the best way (in addition to looking at the actual URLs) is to enter the actual domain you’re classifying add several items to the shopping cart and to start the checkout process, of course, without actually buying the product.

* Cart related URLs (‘View cart’ / ‘Add cart’ / ‘Add/view cart’) – usually contain one of the following words within the URL: cart / basket / buy-box, but not necessarily (both URLs are cart related examples: [http://store.sony.com/webapp/wcs/stores/servlet/OrderItemDisplay?...](http://store.sony.com/webapp/wcs/stores/servlet/OrderItemDisplay) or <http://www.bestbuy.com/site/olspage.jsp?id=pcat17005&type=page&_requestid=240281>)
* Checkout URLs start in all cases with ‘https://’ and in most case contain the word ‘checkout’ within the URL, but not necessarily (both URLs are checkout related examples: <https://www-ssl.bestbuy.com/site/olspage.jsp?id=pcat17002&type=page&_requestid=23765> or <https://ecomm2.dell.com/dellstore/basket.aspx?c=us&cs=19&l=en&s=dhs&itemtype=CFG&oid=b269472a-c9f9-4618-a65e-2e6c404c6534#signin~0>
* Purchase URLs contain in many cases the phrase ‘thankyou’, but not necessarily. It is of course impossible to recognize the actual pattern without completing a purchase on the relevant domain, but this isn’t expected, so see what you can identify within the list of URLs.
* store\_locator URLs contain in many cases one of the following words within the URL: store / locate / location / retailer, but not necessarily.

### URL Patterns

The purpose of writing the URL patterns is on the one hand to capture as much URLs as possible within a single pattern, but on the other hand to be accurate in the written pattern and to capture properly the relevant signals within the URL (see following section).

#### Signals

The table below details all the signals that might be extracted from the URL pattern matching rules. The most important one is the ‘product name’ (pn), if it exists in the URL itself.

|  |  |  |  |
| --- | --- | --- | --- |
| **Signal Name** | **Signal Description** | **Example URL** | **Example Pattern** |
| pn | Product name | http://www.walmart.com/ip/Dyson-DC24-Animal-Ultra-Lightweight-Bagless-Upright-Vacuum-Cleaner/15173329 | [http://www.walmart.com/ip/{pn}/{pid}](http://www.walmart.com/ip/%7bpn%7d/%7bpid%7d) 'pn': 'Dyson-DC24-Animal-Ultra-Lightweight-Bagless-Upright-Vacuum-Cleaner', ‘pid’: ‘15173329’ |
| pid | Product ID as defined by the domain |
| c1, c2, c3 | Category level 1, 2 and 3 as defined by the domain | http://www.walmart.com/browse/ipad-ereaders/tablet-pcs/apple/3944\_1078524\_1078084/YnJhbmQ6QXBwbGUie | http://www.walmart.com/browse/{c1}/{c2}/{c3}/{c2id}/{words}  'c1': 'ipad-ereaders', 'c2': 'tablet-pcs', 'c3': 'apple', ‘c2id’: ‘3944\_1078524\_1078084’ |
| c1id, c2id | Category level 1 and 2 IDs as defined by the domain | http://www.walmart.com/cp/iPad-eReaders-Tablets/1078524 | http://www.walmart.com/cp/{c2}/{c2id}  'c2': 'iPad-eReaders-Tablets', 'c2id': '1078524' |
| brand | The brand of the product | - | - |
| mid | Manufacturer ID as defined by the domain | http://accessories.us.dell.com/sna/Category.aspx?c=us&l=en&cs=19&s=dhs&category\_id=2999&mfgpid=202709&Tab=Parts&stype=2 | http://accessories.us.dell.com/sna/productdetail.aspx?category\_id={c2id}& mfgpid={mid}  'c2id': ‘2999’, ‘mid’: ‘202709’ |
| sku | The SKU of the product as defined by the domain | http://accessories.dell.com/sna/PopupProductDetail.aspx?c=us&amp;amp;l=en&amp;amp;cs=19&amp;amp;sku=332-0335&amp;amp;price=39.99&amp;amp;client=config | http://accessories.us.dell.com/sna/productdetail.aspx?sku={sku}&price={price}  'sku': '332-0335', 'price' : '39.99' |
| price | The price |
| search | The search term used within the domain’s search engine | http://www.walmart.com/search/search-ng.do?search\_query=windows+7+computers | http://www.walmart.com/search/search-ng.do?search\_query={search}  'search': 'windows+7+computers' |
| location | In case of a store locator | http://www.apple.com/retail/southdale | [http://www.apple.com/retail/{location](http://www.apple.com/retail/%7blocation)} 'location' : 'southdale' |

#### URL structure

In order to capture different parts of the URL, if needed, we divided the URL into 5 different parts.

* ‘spattern’ – Relates to the URL scheme, meaning whether the URL starts with HTTP (http://) or with HTTPS (https://).
* ‘hpattern’ – Relates to the hostname part of the URL, meaning everything after the ‘http://’ or the ‘https://’ and before the first ‘/’.
* ‘pattern’ – Relates to the URL path, meaning everything after the ‘hpattern’ and before the ‘?’
* ‘qpattern’ – Relates to the URL query part and doesn’t necessarily have to appear, meaning everything after the ‘?’.
* ‘fpattern’ – Relates to the URL fragment part and doesn’t necessarily have to appear, meaning everything after the ‘#’.
* Examples (different parts of the URL expressed in the different colors):
  + <http://www.walmart.com/ip/9853395#Book+Information>
  + <http://www.walmart.com/ip/Astroglide-Sensual-Strawberry-Personal-Lubricant-2.5-oz/19416634?findingMethod=rr>
  + <https://www.walmart.com/a_d_registration_flow/landing.do;jsessionid=0000000c2bc749b43fd784161f12c8103fd6eab96c3d4cd6>
  + <http://careers.walmart.com/career-areas/retail-operations/store-club-hourly-jobs/>
  + [https://ecomm2.dell.com/dellstore/basket.aspx?c=us&amp;amp;cs=19&amp;amp;l=en&amp;amp;s=dhs&amp;amp;itemtype=SNA&amp;amp;oid=6f15ff24-8541-4657-a1e8-7ae2ef0a3f1f#thankyou~6](https://ecomm2.dell.com/dellstore/basket.aspx?c=us&amp;amp;cs=19&amp;amp;l=en&amp;amp;s=dhs&amp;amp;itemtype=SNA&amp;amp;oid=6f15ff24-8541-4657-a1e8-7ae2ef0a3f1f%23thankyou~6)
  + [https://www.walmart.com/cart2/cart.do#gberr](https://www.walmart.com/cart2/cart.do%23gberr)

#### Pattern

The most important part of the rule and exists in every rule. In some cases it may contain signals such as: 'pattern': 'ip/(?P<pn>.\*?)/(?P<pid>.\*?)([#?]|$)'and in some cases it doesn’t, such as: 'pattern' : 'browse-ng.do'

#### QPattern

In some cases can be part of a specific rule and in some cases when the same query is used by multiple patterns the same qpattern can be used in common for several rules

#### HPattern

Usually used to define rules with an ‘irrelevant’ action. Such as:

{'name' : 'irrelevant\_domain',

'hpattern': '(answers|avocado|brands|careers|wireless|wm|wwwndc)',

'action' : 'irrelevant'}

#### SPatterns

Usually used to define rules with a ‘checkout’ or ‘purchase’ action. Such as:

{'name' : 'checkout2',

'spattern' : 'https',

'pattern' : '^/(wmflows|subflow)/checkout',

'action' : 'checkout',

'norm' : 'https://www.walmart.com/wmflows/checkout'}

#### FPatterns

Usually used to define rules with a ‘checkout’ or ‘purchase’ action. Such as:

{'name' : 'checkout1',

'spattern' : 'https',

'pattern' : '^/cart',

'fpattern' : 'gberr',

'action' : 'checkout',

'norm' : 'https://www.walmart.com/cart2/cart.do#gberr'}

### Norm

The ‘norm’ is the normalized URL, meaning the shortest possible subset of the original URL that would lead to the same page. The URL is cleaned up from all the parameters that don’t affect the content of the displayed page. In many cases everything after the ‘?’ can be dropped, but in some cases there’re some search parameters that can’t be dropped and need to be included. In addition there’re cases in which nothing can be dropped from the original URL and therefore there’s no need to define a normalized URL for such a rule.

The examples below will assist in understanding the concept (the highlighted part in the ‘original URL’ column is the ‘normalized URL.

|  |  |  |
| --- | --- | --- |
| **Original URL** | **Normalized URL** | **Normalized Pattern** |
| http://www.walmart.com/cp/iPad-eReaders-Tablets/1078524?povid=P1262-C1110.2784+1455.2776+1115.2956-L21 | http://www.walmart.com/cp/iPad-eReaders-Tablets/1078524 | http://www.walmart.com/cp/{c2}/{c2id} |
| http://www.walmart.com/browse/electronics/scanners/3944\_3951\_37807\_4439/?tab\_value=all&amp;amp;pref\_store=1576&amp;amp;ss=false&amp;amp;ic=32\_0 | http://www.walmart.com/browse/electronics/scanners/3944\_3951\_37807\_4439/ | http://www.walmart.com/browse/{c1}/{c2}/{c2id}/ |
| https://www.walmart.com/cart2/cart.do?webflowforward=true&amp;amp;webflowforward=true | https://www.walmart.com/cart2/cart.do | https://www.walmart.com/cart2/cart.do |
| http://www.walmart.com/search/browse-ng.do?\_refineresult=true&amp;amp;povid=cat1070145-env172199-moduleA080112-lLinkGNAV\_Electronics\_Electronics\_GPS\_Navigation&amp;amp;facet=brand%3AGarmin&amp;amp;cat\_id=3944&amp;amp;fromPageCatId=538883 | http://www.walmart.com/search/browse-ng.do?facet=brand%3AGarmin | http://www.walmart.com/search/browse-ng.do?facet={facet} |
| http://www.walmart.com/search/search-ng.do?tab\_value=all&search\_query=floor+lamp&search\_constraint=4044&Find=Find&cat\_id=4044&facet=price%3a%2420+-+%2450&pref\_store=5129&ss=false&ic=16\_32&\_be\_related=2 | http://www.walmart.com/search/search-ng.do?search\_query=floor+lamp&facet=price%3a%2420+-+%2450 | http://www.walmart.com/search/search-ng.do?search\_query={search}&facet={facet} |
| http://www.sears.com/shc/s/clothing-women-s-dresses&amp;amp;Sears\_Next/s-1023574?filter=storeOrigin\_storeOrigin&amp;amp;viewItems=50&amp;amp;i\_cntr=1366336840304 | http://www.sears.com/shc/s/clothing-women-s-dresses&amp;amp;Sears\_Next/s-1023574 | http://www.sears.com/shc/s/{c2}/{letter}-{c2id} |
| http://store.apple.com/us/search?find=usb+adapter+for+ipad+that+you+can+use+for+a+fash+drive | Same as original URL | None |

## Output File Structure

The expected output for each classified domain is a python file in the format of ‘domainName.py’, for example: walmart.py or sears.py. This section will explain the file’s format.

### The match(url) function

This function defines the domain we’re working on in this file, as the example below shows:

def match(url):

return url.hostname.endswith('walmart.com')

### The rules definition

The previous sections described in details how to write rules to classify URLs. This section will focus on some general guidelines regarding the rules that should be written.

* You should surely have a rule with an ‘irrelevant’ action for all the hostnames under the domain that are irrelevant to purchasing as shown in the [‘HPattern’ section](#_HPattern).
* In many domains you will have a rule with an ‘irrelevant’ action to capture all the paths that not related to purchasing, such as this example:

{'name' : 'irrelevant1',

'pattern' : '(support|downloads|api)',

'action' : 'irrelevant'}

* You should surely have at least one rule with a ‘product’ action and one rule with a ‘list’ action. As these are the most popular actions, for most domains it is expected to have several rules for these actions in order to capture the different URL patterns.
* You will most likely have at least one rule that captures a ‘search’ signal for searches that are run in the domain, since all sites have a built in search engine. The action of this rule is most likely to be with a ‘list’ action. For example:

{'name' : 'search',

'pattern' : '^/(?P<c2>.\*?)/(?P<letter>[svc]?)-(?P<c2id>.\*?)([&#?]|$)',

'action' : 'list',

'norm' : 'http://www.sears.com/{c2}/{letter}-{c2id}?**keyword={search}**',

'qpattern': '**keyword=(?P<search>.\*?)**([&]|$)'}

* You should surely have a rule with an ‘irrelevant’ action for all the hostnames under the domain that are irrelevant to purchasing as shown in the [‘HPattern’ section](#_HPattern).
* In most domains you should have at least one rule with each one of the following actions: ‘checkout’, ‘store\_locator’, ‘purchase’ and one of the cart related actions (‘View cart’ / ‘Add cart’ / ‘Add/view cart’).
* You should note that in some cases the order of the rules matters, since some rules may override others. In the example below ‘ip1’ must appear before ‘ip2’ otherwise all URLs that can match the ‘ip1’ rule will match the ‘ip2’ rule if it comes before (as it is more general):

{'name' : 'ip1',

'pattern' : 'ip/(?P<pn>.\*?)/(?P<pid>.\*?)([#?]|$)',

'action' : 'product',

'norm' : 'http://www.walmart.com/ip/{pn}/{pid}'}

{'name' : 'ip2',

'pattern' : 'ip/(?P<pid>.\*?)([#?]|$)',

'action' : 'product',

'norm' : 'http://www.walmart.com/ip/{pid}'}

### The params definition

Here you should define all the patterns to match the query part of the URL, which are common for several rules. For example:

params = [

{'qpattern': 'search\_query=(?P<search>.\*?)([&]|$)'},

{'qpattern': 'facet=(?P<facet>.\*?)([&]|$)'},

{'qpattern': 'sfsearch\_zip=(?P<location>.\*?)([&]|$)'},

]

If no such patterns exists than ‘params’ should be empty like this:

params = [

]

### The examples definition

This section serves as unit testing of the defined rules. Here you should add **AT LEAST** one example per each rule that was defined. For trivial rules, one example is enough. An example of a trivial rule is:

{'name' : 'storeLocator',

'pattern' : '^/storeLocator',

'action' : 'store\_locator'}

For more complex rules add two or three examples. An example of a more complex rule is:

{'name' : 'search',

'pattern' : '^/(?P<c2>.\*?)/(?P<letter>[svc]?)-(?P<c2id>.\*?)([&#?]|$)',

'action' : 'list',

'norm' : 'http://www.sears.com/{c2}/{letter}-{c2id}?keyword={search}',

'qpattern': 'keyword=(?P<search>.\*?)([&]|$)'}

An example is constituted from the following parts:

* URL – An example URL that should match the rule.
* ‘norm’ – The normalized URL, if one was defined for the rule.
* ‘action’ – The action reflected by the rule.
* Signals – All the signals that are extracted according to the rule definition, if defined.
* ‘rule’ – The rule name that is being tested.

Here are some examples of examples:

['<http://www.sears.com/appliances-accessories-washer-dryers/s-1023537?keyword=ge+washing+machines+on+sale&amp;amp;autoRedirect=true&amp;amp;viewItems=25&amp;amp;redirectType=CAT_REC_PRED'>,

{'norm': '<http://www.sears.com/appliances-accessories-washer-dryers/s-1023537?keyword=ge+washing+machines+on+sale>', 'action': 'list', 'c2': 'appliances-accessories-washer-dryers', 'letter': 's', 'c2id': '1023537', 'search': 'ge+washing+machines+on+sale', 'rule': 'search'}]

['<http://www.sears.com/acer-aspire-as5749z-4809-notebook-pc/p-020W004451989000P?prdNo=13&amp;amp;blockNo=13&amp;amp;blockType=G13>',

{'norm': '<http://www.sears.com/acer-aspire-as5749z-4809-notebook-pc/p-020W004451989000P>', 'action': 'product', 'pn': 'acer-aspire-as5749z-4809-notebook-pc', 'pid': '020W004451989000P', 'rule': 'product'}]

['<http://www.dell.com/support/drivers/us/en/04/ProductSelector/Select/Progress?DownloadClient=False&amp;amp;rquery=~srd-e-true-a-sk-e-st%20micro%20screen%20detection%20sensor-a-scat-e-prod>',

{'action': 'irrelevant', 'rule': 'irrelevant1'}]

## Testing your code with the url\_parser application

This section will explain how to easily test your code using our unit testing (UT) application. This application will run your examples one by one and test the actual results you defined (as described in the previous section) to the planned (or expected) results as defined by the examples. From our experience is it best to write down a rule, immediately after add the examples to test it, then run the test application and see that it completes correctly and only then move to writing to the next rule.

If all your examples match the rules properly the unit testing application will just return ‘True’, otherwise the application will report the problem (see example in following section) and return ‘False’. During the rule development it might be useful to use the RegEx site mentioned in the Prerequisites section.

### Adjusting the Unit Testing Application and running it

In the end of the supplied file ‘url\_parser.py’ you have the following lines:

if \_\_name\_\_ == "\_\_main\_\_":

print test\_examples('walmart')

All you need to do is replace ‘walmart’ with the name of the file, which is the name of the domain you’re classifying. Your file should be placed in the same directory as the ‘url\_parser.py’ file.

Running the test application is done by running the ‘url\_parser.py’ code (as any other Python code).

### Common problems when running the Unit Testing Application

#### Example 1: 'XXX: not found in actual'

The rule:

{'name' : 'shc\_search\_only',

'pattern' : '^/shc/s/search\_(?P<numbers>.\*?)([?]|$)',

'action' : 'list',

'norm' : 'http://www.sears.com/shc/s/search\_{numbers}?keyword={search}',

#'qpattern': 'keyword=(?P<search>.\*?)([&]|$)' # In comment on purpose to fail the test

}

The test example:

['http://www.sears.com/shc/s/search\_10153\_12605?vName=Appliances&amp;amp;cName=Microwaves&amp;amp;keyword=whirlpool+microwave&amp;amp;sid=ISx20070515x00001d&amp;amp;psid=15x1222646&amp;amp;knshCrid=1728848975&amp;amp;k\_clickID=06860883-73e1-c2c9-d27c-00002c7e8150',

{'norm': 'http://www.sears.com/shc/s/search\_10153\_12605?keyword=whirlpool+microwave', 'action': 'list', 'numbers': '10153\_12605', 'search': 'whirlpool+microwave', 'rule': 'shc\_search\_only'}],

The UT response:

http://www.sears.com/shc/s/search\_10153\_12605?vName=Appliances&amp;amp;cName=Microwaves&amp;amp;keyword=whirlpool+microwave&amp;amp;sid=ISx20070515x00001d&amp;amp;psid=15x1222646&amp;amp;knshCrid=1728848975&amp;amp;k\_clickID=06860883-73e1-c2c9-d27c-00002c7e8150 ['search: not found in actual', 'norm: not found in actual']

The Explanation:

There’s no ‘qpattern’ rule that can extract the search term from the query part of the URL, therefore the response of the UT application is that the ‘search’ wasn’t found in actual (while expected in the planned results) and same for the ‘norm’ that doesn’t match.

#### Example 2: 'rule: expected ip1, found ip2'

The UT response:

http://www.walmart.com/ip/Dyson-DC24-Animal-Ultra-Lightweight-Bagless-Upright-Vacuum-Cleaner/15173329?wmlspartner=HBLvzQS2RdU&amp;amp;sourceid=00051088942570117644&amp;amp;affillinktype=3&amp;amp;veh=aff ['pid: expected 15173329, found Dyson-DC24-Animal-Ultra-Lightweight-Bagless-Upright-Vacuum-Cleaner/15173329', 'pn: not found in actual', 'rule: expected ip1, found ip2']

The Explanation:

The example didn’t match the rule ‘ip1’ as expected instead it matched ‘ip2’. This problem usually happens because the rules weren’t put in the correct order (or the patterns weren’t defined properly), therefore the example matched the incorrect rule. The rest of the reported problem (incorrect ‘pid’ and ‘pn’ are as a result of the rule mismatch).

#### Example 3: 'XXX: not found in expected'

The rule:

{'name' : 'cp1',

'pattern' : 'cp/(?P<c2>.\*?)/(?P<c2id>.\*?)([#?]|$)',

'action' : 'list',

'norm' : 'http://www.walmart.com/cp/{c2}/{c2id}'}

The test example:

['http://www.walmart.com/cp/iPad-eReaders-Tablets/1078524?povid=P1262-C1110.2784+1455.2776+1115.2956-L21',

{'norm': 'http://www.walmart.com/cp/iPad-eReaders-Tablets/1078524', 'action': 'list', 'c2': 'iPad-eReaders-Tablets', 'rule': 'cp1'}]

The UT response:

http://www.walmart.com/cp/iPad-eReaders-Tablets/1078524?povid=P1262-C1110.2784+1455.2776+1115.2956-L21 ['c2id: not found in expected']

The Explanation:

The UT Application extracted the ‘c2id’ signal from the URL, while this signal wasn’t found in the expected results. In order to fix it the test example should look as following:

['http://www.walmart.com/cp/iPad-eReaders-Tablets/1078524?povid=P1262-C1110.2784+1455.2776+1115.2956-L21',

{'norm': 'http://www.walmart.com/cp/iPad-eReaders-Tablets/1078524', 'action': 'list', 'c2': 'iPad-eReaders-Tablets', 'c2id': '1078524', 'rule': 'cp1'}],

## Testing your code with the url\_classification application

This section will explain how to easily test your code using our url\_classification application. This application receives as an input a file with URLs from one domain in the format that is described in the [Input](#_Input) section. This application runs over all the URLs in the input file and classifies them according to the classification rules defined for the domain. The output of this application (to stdout) is the breakdown of all URLs in the input file to the URL classification rule that matched them. If no rule was found then the URL will be under the 'unclassified' category (see example in following section). In addition the application writes all the unclassified URLs to the output file in the same format as the input file. The goal of running this application is to evaluate the percentage of URLs that hasn’t been classified and if needed to work on the unclassified URLs to reach a higher classification percentage.

### Adjusting the Application and running it

In the beginning of the supplied file ‘url\_classification.py’ you have the following lines:

mod = \_\_import\_\_('walmart', globals(), locals(), ['\*'], -1)

And in the end of the file you have the following line:

process\_file('../Domains/walmart.txt', '../Domains/walmart\_unclassified.txt')

All you need to do is replace ‘walmart’ with the name of the file, which is the name of the domain you’re classifying. The first parameter to the process\_file function is the input file and the second is the output file (for the unclassified URLs).

### The Application’s output

The application will output (to stdout) the list of rules, where each line contains the rule name, the number of occurrences of this rule in the input file (summing the number of occurrences column in the input file) and the percentage of this rule from the total number of occurrences. Below is an example of the output:

list 10 29.4117647059%

search 9 26.4705882353%

product 8 23.5294117647%

unclassified 7 20.5882352941%

The application may take several minutes to run depending on the size of the input file.

# Page Scraping

After completing the domain classification phase, the next step is writing a page scraping function for each one of the relevant rules defined in the domain classification phase. The sections below explain in detail when to write a scraping function, what should be scraped as well as how to test the code and the expected output.

## Deciding which rules should have a scraping function

For each rule from the domain classification phase with one of the following actions: ‘List’, ‘Product’, ‘Compare’ and ‘Review’ that doesn’t include a ‘product name’ signal in the pattern itself a scraping function should be written to capture the ‘product name’ (pn) from within the page. If multiple products are presented within the page (this is surely the situation in rules that have an action of ‘List’ and ‘Compare’ and most likely the case for a ‘Review’ action as well) only the first product name should be returned by the function.

Several examples:

|  |  |
| --- | --- |
| **URL classification rule** | **Should have a scraping function?** |
| {'name' : 'search\_browse',  'pattern' : 'browse-ng.do',  'action' : 'list',  'norm' : 'http://www.walmart.com/search/browse-ng.do?facet={facet}'} | Yes. Action is ‘list’ and pn isn’t read from the URL |
| {'name' : 'ip1',  'pattern' : 'ip/(?P<pn>.\*?)/(?P<pid>.\*?)([#?]|$)',  'action' : 'product',  'norm' : 'http://www.walmart.com/ip/{pn}/{pid}'} | No. pn exists in the URL |
| {'name' : 'ip2',  'pattern' : 'ip/(?P<pid>.\*?)([#?]|$)',  'action' : 'product',  'norm' : 'http://www.walmart.com/ip/{pid}'} | Yes. Action is ‘product’ and pn isn’t read from the URL |
| {'name' : 'checkout2',  'spattern': 'https',  'pattern' : '^/(wmflows|subflow)/checkout',  'action' : 'checkout',  'norm' : 'https://www.walmart.com/wmflows/checkout'} | No. Although pn isn’t read from the URL, the action is ‘checkout’ |

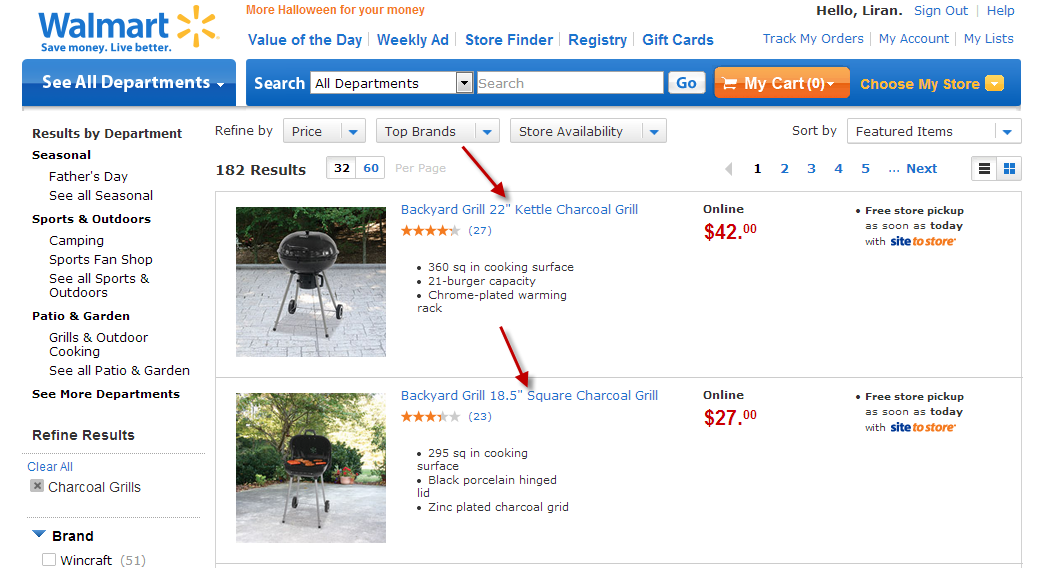
## The scraping function

The easiest way to write the scraping function is by following these steps:

### Locating the product names within the page

Open 2-3 pages that are classified by the rule you wish to write a scraping function for and locate the product name within the page (you should have those pages from the examples you wrote for the domain classification page phase and of course you can keep more URL examples aside than used in the examples). See examples below for locating the product name within the page. Product names are pointed by a red arrow.

<http://www.walmart.com/search/browse-ng.do?facet=category%3ACharcoal+Grills>:



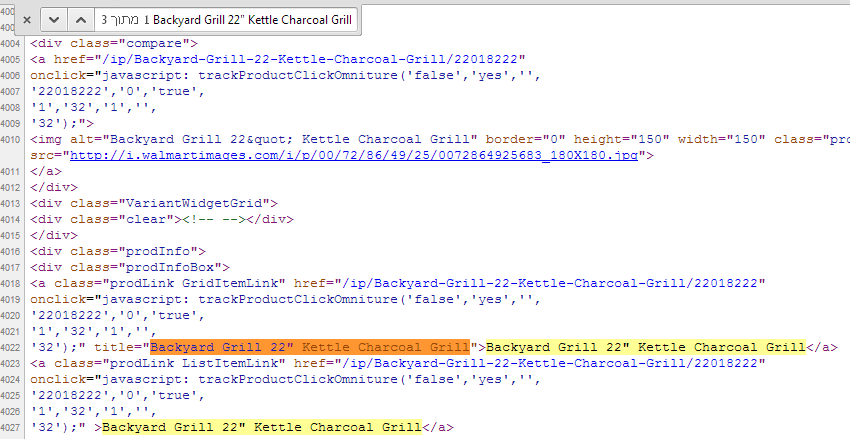
<http://www.walmart.com/ip/9863255>:

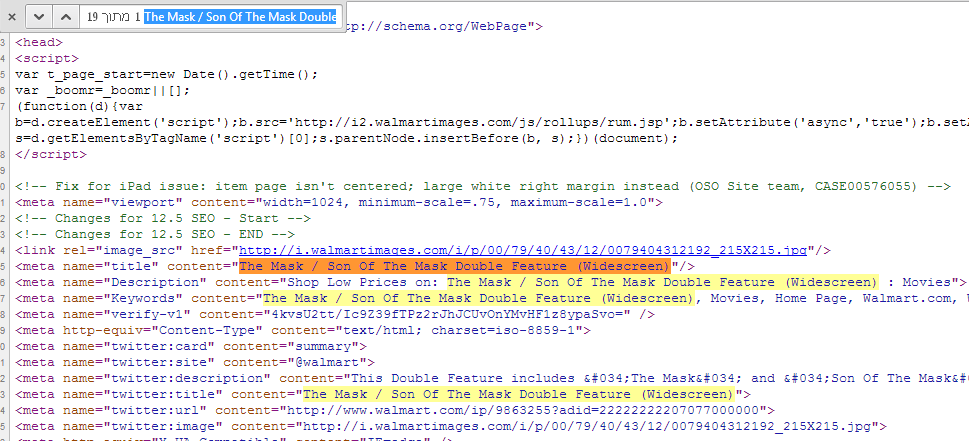


### Locating the product names within the page’s source code

Next step would be to open the source code of the page and locating the same product name within the source code. You should find the best reference (easiest to extract) and one that is consistent across all pages that are classified by the same rule. If you found more than one good references in the source code to the product name, add them several of them to the scraping function.

See examples below (respectively to the above pages):





### Writing the scraping function

The simplest way of understanding the scraping functions is by looking at examples (respectively to the above pages):

‘names’ always collects all the matches of this pattern within the page, then we check if names contains anything and if so return the first entrance.

def parse\_prodInfoBox(x, item):

names = x.select('//div[@class="prodInfoBox"]/a[contains(@class,"prodLink")]/@title').extract()

if names:

item['scrap\_pn'] = names[0].strip()

return item

In this example we try to capture the first reference to the product name within the page, if not found we try the second and if not found too try the third.

def parse\_ip2(x, item):

names = x.select('//meta[@name="title"]/@content').extract()

if names:

item['scrap\_pn'] = names[0].strip()

else:

names = x.select('//h1[@class="productTitle"]/text()').extract()

if names:

item['scrap\_pn'] = names[0].strip()

else:

names = x.select('//span[@id="SAC\_prodName"]/text()').extract()

if names:

item['scrap\_pn'] = names[0].strip()

return item

## Output file structure

The expected output for each domain is a python file in the format of ‘domainName.py’, for example: walmart.py or sears.py. This section will explain the file’s format.

### The scraping functions

This section will include all the scraping functions that are required to scrape the domain. For consistency all function name would be parse\_XXX, when XXX is a meaningful name (either the tag that is used to scrape the page as shown in the first example above or the name of the URL classification rule in case many tags are used as shown in the second example above.

### The parsers definition

Here we match between a URL classification rule and a ‘parse’ function. Each rule should have only one ‘parse’ function correlated to it (so if the product name can be scraped from several places within the page they should all be incorporated into one function as can be seen in my example), while a parsing function can be used by several rules. Below is an example:

parsers = {

'ip2': parse\_ip2,

'cp1': parse\_seo\_h1,

'cp2': parse\_seo\_h1,

'browse1': parse\_seo\_h1\_or\_SRNode\_selected,

'browse2': parse\_seo\_h1\_or\_SRNode\_selected,

'browse3': parse\_seo\_h1\_or\_SRNode\_selected,

'search\_browse': parse\_prodInfoBox,

'search': parse\_prodInfoBox,

}

### The examples definition

This section serves as unit testing of the ‘parse’ functions. Here you should add **AT LEAST** one example per each ‘parse’ function that was defined. For trivial functions, one example is enough. An example of a trivial function is the above parse\_prodInfoBox function. For more complex functions add an example for each case. An example for a more complex function is the above parse\_ip2 function.

An example is constituted from the following parts:

* The ‘parse’ function.
* URL – A URL that can be scraped and contains the relevant tags.
* Html – The content of the page specified by the URL, from which the ‘product name’ would be extracted.
* pn – The ‘product name’ extracted from the page.

[parse\_ip2,

{

'url' : 'http://www.walmart.com/ip/9863255',

'html' : '''

<meta name="title" content="The Mask / Son Of The Mask Double Feature (Widescreen)"/>

''',

'pn' : 'The Mask / Son Of The Mask Double Feature (Widescreen)'

}]

## Testing your code with the scraper\_tests application

This section will explain how to easily test your code using our unit testing (UT) application. This application will run your examples one by one and test the actual results you defined (as described in the previous section) to the planned (or expected) results as defined by the examples. From our experience is it best to write down a ‘parse’ function, immediately after add the examples to test it, then run the test application and see that it completes correctly and only then move to writing to the next function.

If all your examples match the rules properly the unit testing application will just return ‘OK (and the time it ran), otherwise the application will report the problem (see example in following section) and return ‘Failed’ with the number of failures. During the ‘parse’ function development it might be useful to use the sourceforge site mentioned in the Prerequisites section.

### Adjusting the Unit Testing Application and running it

In the end of the supplied file ‘scraper\_tests.py’ you have the test\_domain function which has the following lines:

def test\_domain(self):

domain = 'walmart'

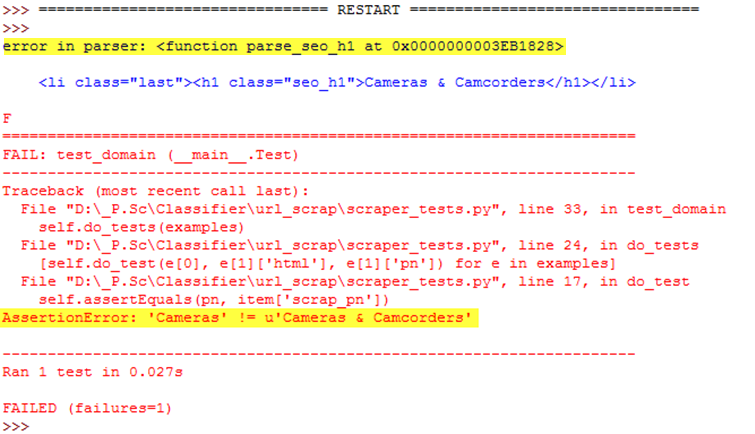
All you need to do is replace ‘walmart’ with the name of the file, which is the name of the domain you’re scraping. Your file should be placed in the same directory as the ‘scraper\_tests.py’ file.

Running the test application is done by running the ‘scraper\_tests.py’ code (as any other Python code).

### Common problems when running the Unit Testing Application

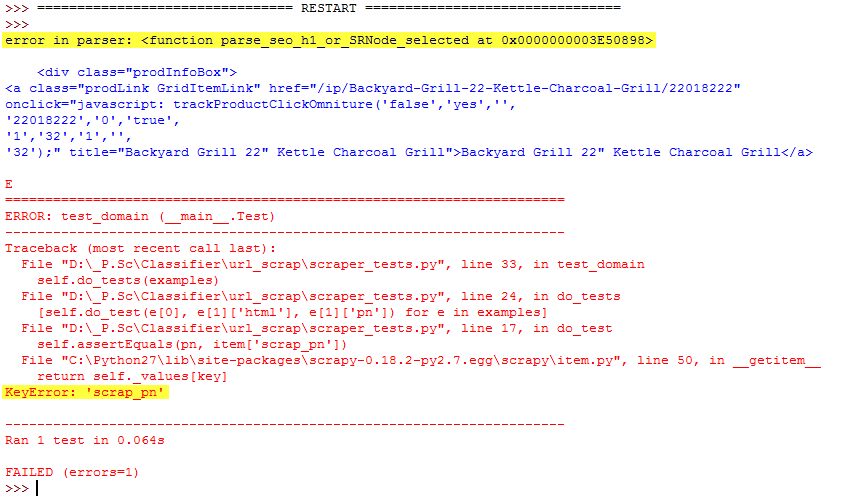
#### Example 1: ‘AssertionError’

The highlighted error happens when there’s a mismatch between the expected result and the actual result that was scraped. To fix this problem you should check your example or the parsing function.



#### Example 2: ‘KeyError: 'scrap\_pn'’

The highlighted error happens when you’re trying to scrape an example with the wrong scraping rule. To fix this problem you should check your example code.



## Testing your code with the scraper application

This section will explain how to easily test your code using the scraper application. Unlike the previous application, this application will actually scrape the defined URL using the define URLs, so it is good to run it to see that the example URL can actually be scraped. This application will print as an output what it scraped from the URL with some debug info.

### Adjusting the scraper Application and running it

In the end of the supplied file ‘scraper.py’ you need to define the page scraping requests in the following manner:

requests = (request(URL1, domain.parse\_function1, None),

request(URL2, domain.parse\_function2, None),

request(URL3, domain.parse\_function3, None),

…)

For Example:

requests = (request("http://www.walmart.com/browse/computers/laptop-computers/toshiba/3944\_3951\_132960/YnJhbmQ6VG9zaGliiYQieie", walmart.parse\_seo\_h1\_or\_SRNode\_selected, None),

request("http://www.walmart.com/search/browse-ng.do?facet=category%3ACharcoal+Grills", walmart.parse\_prodInfoBox, None)

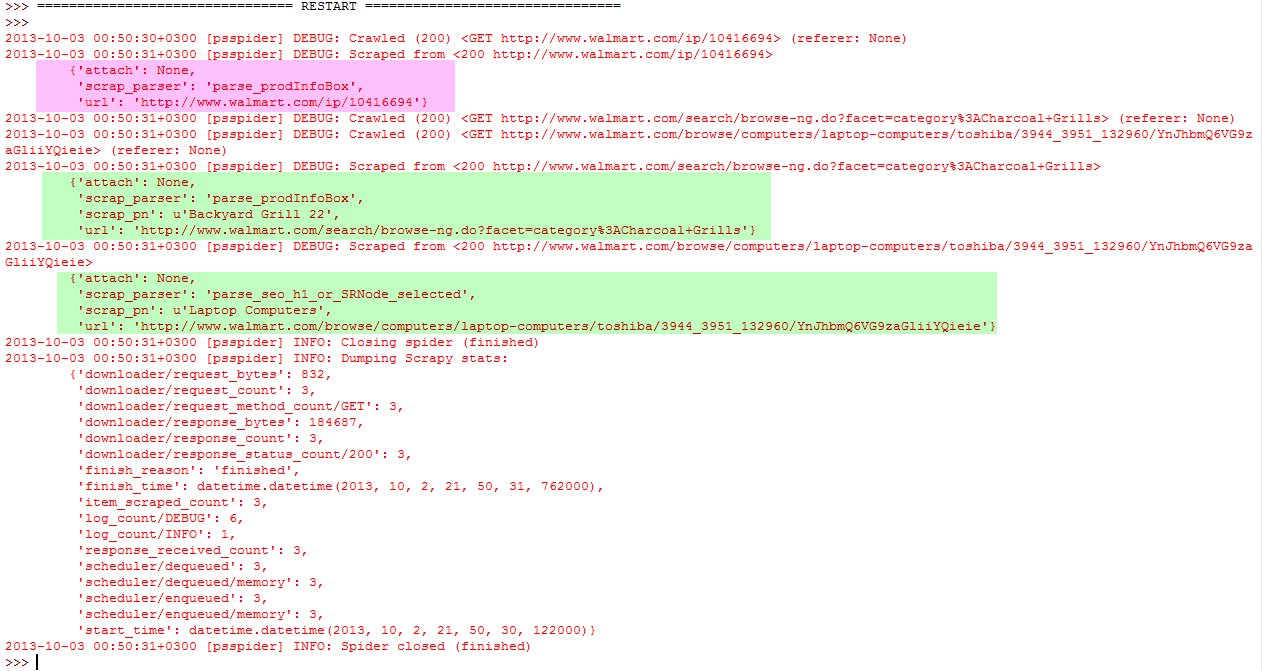
request("http://www.walmart.com/ip/10416694", walmart.parse\_ip2, None))

In addition in the following import line at the beginning of the file you should add the domain you’re working on:

import walmart, sears

### The run’s result

When there are no scraping issues the expected result includes the ‘scrap\_pn’ as shown below highlighted in green. When there are scraping issues the expected result doesn’t include the ‘scrap\_pn’ as shown below highlighted in pink.



# Summary

Once you’ve finished writing both python files and they successfully pass all the testing examples you can submit the files by email along with the tables (Excel or text file) leading you to the decision which URLs to classify as described in the [Deciding which URLs to classify](#_Deciding_which_URLs) section.

**Good Luck!**