Scraping and Extracting Links from any major Search Engine like Google, Yandex, Baidu, Bing and Duckduckgo

BY nikolai (http://incolumitas.com/author/admin/) - POSTED ON November 12, 2014 (http://incolumitas.com/2014/11/12/)

Hey dear readership!

Prelude

It’s been quite a while since I worked on my projects. But recently I had some motivation and energy left, which is quite nice considering my full time university week and a programming job besides.

I have a little project on GitHub (https://github.com/NikolaiT/GoogleScraper) that I worked on every now and again in the last year or so. Recently it got a little bit bigger (I have 115 github stars now, would’ve never imagined that I ever achieve this) and I receive up to 2 mails with job offers every week (Sorry if I cannot accept any request :( ).

But unfortunately my progress with this project is not as good as I want it to be (that’s probably a quite common feeling under us programmers). It’s not a problem of missing ideas and features that I want to implement, the hard part is to extend the project without blowing legacy code up. GoogleScraper has grown evolutionary and I am waisting **a lot** of time to understand my old code. Mostly it’s much better to just erease whole modules and reimplement things completely anew. This is essentially what I made with the parsing module.

Parsing SERP pages with many search engines

So I rewrote the parsing.py module of GoogleScraper. From now on, parsing happens much more stable and is more extendable. In fact, everyone can add their own CSS selectors while subclassing the abstract `Parser` class. For now, parsing.py support the following search engines:

- Google (https://google.com) (as before)
- Yandex (http://yandex.ru/) (quite a nice search engine)
- Bing (http://www.bing.com) (pretty mature by now)
- Yahoo (http://https://search.yahoo.com) (good old google competitor)
- Baidu (http://www.baidu.com/) (let’s dive into the asian market ;))
- Duckduckgo (https://duckduckgo.com) (I am very excited about duckduck.go, because the results are clean any very easily parsable)

This means that GoogleScraper now support **6 search engines**. So you can scale your scraping and compare the results between search engines. This means much more output and statistical data for your analysis. You can also divide your scrape jobs on the different search engines. A few people might still say that Google is the only usable search engine. Have you actually used an alternative recently or are you just suffering from the locked in effect (http://en.wikipedia.org/wiki/Lock-in_%28decision-making%29)?

Let’s play with it

Well, to give you some first insight in the new functionality, lets dig some code and see it in action:

To run it download the code below, save it as parsing.py and just install the modules:

- lxml
- cssselect
- beautifulsoup4

You can do so with `sudo pip3 install modulename` .

Now when you are ready, you can easily test the new parsing functionality with firing such an example command in the command line:

```
python3 parsing.py 'http://yandex.ru/yandsearch?text=GoogleScraper&lr=178'
```

This will scrape the results from Yandex with the search query **GoogleScraper**. You can try it out with the other search engines: Just search in your browser, than copy and paste the url from the address bar in the command!

Please note: Using this module directly makes little sense, because requesting such urls directly without imitating a real browser (which is done in my GoogleScraper module with faking User Agent, using selenium, PhantomJS, …) makes the search engines sometimes return crippled html, which makes it hard to parse.

But for some engines it nevertheless works quite well (for example: yandex, google, …).

Please note, the most actual version of the code can be found here: parsing.py at GoogleScraper (https://github.com/NikolaiT/GoogleScraper/blob/master/GoogleScraper/parsing.py)

```python
# -*- coding: utf-8 -*-

"""
author: Nikolai Tschacher
date: 11.11.2014
home: incolumitas.com
"""

# TODO: Implement alternatate selectors for different SERP formats (just use a list in the CSS selector datatypes)

import sys
import re
import lxml.html
import logging
import urllib
import pprint

try:
    from cssselect import HTMLTranslator, SelectorError
    from bs4 import UnicodeDammit
except ImportError as ie:
    if hasattr(ie, 'name') and ie.name == 'bs4' or hasattr(ie, 'args') and 'bs4' in str(ie):
        sys.exit('Install bs4 with the command "sudo pip3 install beautifulsoup4"')

logger = logging.getLogger('GoogleScraper')

class InvalidSearchTypeExcpetion(Exception):
    pass

class Parser():
    """Parses SERP pages.

    Each search engine results page (SERP) has a similar layout:

    The main search results are usually in a html container element (#main, .results, #leftSide).
    There might be separate columns for other search results (like ads for example). Then each
    result contains basically a link, a snippet and a description (usually some text on the
    target site). It's really astonishing how similar other search engines are to Google.

    Each child class (that can actual parse a concrete search engine results page) needs
    to specify css selectors for the different search types (Like normal search, news search, video search, ...).

    Attributes:
        search_results: The results after parsing.
    """

    # The supported search types. For instance, Google supports Video Search, Image Search, News search
    search_types = []

    def __init__(self, html, searchtype='normal'):
        """Create new Parser instance and parse all information.

        Args:
            html: The raw html from the search engine search
            searchtype: The search type. By default "normal"

        Raises:
            Assertion error if the subclassed
            specific parser cannot handle the the settings.
        """
        assert searchtype in self.search_types

        self.html = html
        self.searchtype = searchtype
        self.dom = None

        self.search_results = {}

        # Try to parse the provided HTML string using lxml
        doc = UnicodeDammit(self.html, is_html=True)
        parser = lxml.html.HTMLParser(encoding=doc.declared_html_encoding)
        self.dom = lxml.html.document_fromstring(self.html, parser=parser)
        self.dom.resolve_base_href()

        # lets do the actual parsing
        self._parse()

        # Apply sublcass specific behaviour after parsing has happened
        self.after_parsing()

    def _parse(self):
        """Parse the dom according to the provided css selectors.

        Raises: InvalidSearchTypeExcpetion if no css selectors for the searchtype could be found.
        """
        # try to parse the number of results.
        attr_name = self.searchtype + '_search_selectors'
        selector_dict = getattr(self, attr_name, None)

        # short alias because we use it so extensively
        css_to_xpath = HTMLTranslator().css_to_xpath

        # get the appropriate css selectors for the num_results for the keyword
        num_results_selector = getattr(self, 'num_results_search_selectors', None)
        if num_results_selector:
            self.search_results['num_results'] = self.dom.xpath(css_to_xpath(num_results_selector))[0].text_content()

        if not selector_dict:
            raise InvalidSearchTypeExcpetion('There is no such attribute: {}. No selectors found'.format(attr_name))

        for result_type, selectors in selector_dict.items():
            self.search_results[result_type] = []

            results = self.dom.xpath(
                css_to_xpath('{container} {result_container}'.format(**selectors))
            )

            to_extract = set(selectors.keys()) - {'container', 'result_container'}
            selectors_to_use = dict(((key, selectors[key]) for key in to_extract if key in selectors.keys()))

            for index, result in enumerate(results):
                # Let's add primitve support for CSS3 pseudo selectors
                # We just need two of them
                # ::text
                # ::attr(someattribute)

                # You say we should use xpath expresssions instead?
                # Maybe you're right, but they are complicated when it comes to classes,
                # have a look here: http://doc.scrapy.org/en/latest/topics/selectors.html
                serp_result = {}
                for key, selector in selectors_to_use.items():
                    value = None
                    if selector.endswith('::text'):
                        try:
                            value = result.xpath(css_to_xpath(selector.split('::')[0]))[0].text_content()
                        except IndexError as e:
                            pass
                    else:
                        attr = re.search(r'::attr\((?P<attr>.*)\)$', selector).group('attr')
                        if attr:
                            try:
                                value = result.xpath(css_to_xpath(selector.split('::')[0]))[0].get(attr)
                            except IndexError as e:
                                pass
                        else:
                            try:
                                value = result.xpath(css_to_xpath(selector))[0].text_content()
                            except IndexError as e:
                                pass
                    serp_result[key] = value
                if serp_result:
                    self.search_results[result_type].append(serp_result)

    def after_parsing(self):
        """Sublcass specific behaviour after parsing happened.

        Override in subclass to add search engine specific behaviour.
        Commonly used to clean the results.
        """

    def __str__(self):
        """Return a nicely formated overview of the results."""
        return pprint.pformat(self.search_results)

"""
Here follow the different classes that provide CSS selectors
for different types of SERP pages of several common search engines.

Just look at them and add your own selectors in a new class if you
want the Scraper to support them.

You can easily just add new selectors to a search engine. Just follow
the attribute naming convention and the parser will recognize them:

If you provide a dict with a name like finance_search_selectors,
then you're adding a new search type with the name finance.

Each class needs a attribute called num_results_search_selectors, that
extracts the number of searches that were found by the keyword.
"""


class GoogleParser(Parser):
    """Parses SERP pages of the Google search engine."""

    search_types = ['normal', 'image']

    num_results_search_selectors = 'div#resultStats'

    normal_search_selectors = {
        'results': {
            'container': '#center_col',
            'result_container': 'li.g ',
            'link': 'h3.r > a:first-child::attr(href)',
            'snippet': 'div.s span.st::text',
            'title': 'h3.r > a:first-child::text',
            'visible_link': 'cite::text'
        },
        'ads_main' : {
            'container': '#center_col',
            'result_container': 'li.ads-ad',
            'link': 'h3.r > a:first-child::attr(href)',
            'snippet': 'div.s span.st::text',
            'title': 'h3.r > a:first-child::text',
            'visible_link': '.ads-visurl cite::text',
        }
    }

    image_search_selectors = {
        'results': {
            'container': 'li#isr_mc',
            'result_container': 'div.rg_di',
            'imgurl': 'a.rg_l::attr(href)'
        }
    }

    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)

    def after_parsing(self):
        """Clean the urls.

        A typical scraped results looks like the following:

        '/url?q=http://www.youtube.com/user/Apple&sa=U&ei=lntiVN7JDsTfPZCMgKAO&ved=0CFQQFjAO&usg=AFQjCNGkX65O-hKLmyq1FX9HQqbb9iYn9A'

        Clean with a short regex.
        """
        super().after_parsing()
        for key, value in self.search_results.items():
            if isinstance(value, list):
                for i, item in enumerate(value):
                    if isinstance(item, dict) and item['link']:
                        result = re.search(r'/url\?q=(?P<url>.*?)&sa=U&ei=', item['link'])
                        if result:
                            self.search_results[key][i]['link'] = result.group('url')


class YandexParser(Parser):
    """Parses SERP pages of the Yandex search engine."""

    search_types = ['normal']

    num_results_search_selectors = None

    normal_search_selectors = {
        'results': {
            'container': 'div.serp-list',
            'result_container': 'div.serp-item__wrap ',
            'link': 'a.serp-item__title-link::attr(href)',
            'snippet': 'div.serp-item__text::text',
            'title': 'a.serp-item__title-link::text',
            'visible_link': 'a.serp-url__link::attr(href)'
        },
    }


class BingParser(Parser):
    """Parses SERP pages of the Bing search engine."""

    search_types = ['normal']

    num_results_search_selectors = '.sb_count'

    normal_search_selectors = {
        'results': {
            'container': 'ol#b_results',
            'result_container': 'li.b_algo',
            'link': '.b_title > h2 > a::attr(href)',
            'snippet': '.b_snippet > p::text',
            'title': '.b_title > h2 > a::text',
            'visible_link': 'cite::text'
        },
        'ads_main' : {
            'container': 'ol#b_results',
            'result_container': 'li.b_ad',
            'link': '.sb_add > h2 > a::attr(href)',
            'snippet': '.b_caption::text',
            'title': '.sb_add > h2 > a::text',
            'visible_link': 'cite::text'
        }
    }


class YahooParser(Parser):
    """Parses SERP pages of the Yahoo search engine."""

    search_types = ['normal']

    num_results_search_selectors = '#pg > span:last-child'

    normal_search_selectors = {
        'results': {
            'container': '#main',
            'result_container': '.res',
            'link': 'div > h3 > a::attr(href)',
            'snippet': 'div.abstr::text',
            'title': 'div > h3 > a::text',
            'visible_link': 'span.url::text'
        },
    }


class BaiduParser(Parser):
    """Parses SERP pages of the Baidu search engine."""

    search_types = ['normal']

    num_results_search_selectors = '#container .nums'

    normal_search_selectors = {
        'results': {
            'container': '#content_left',
            'result_container': '.result-op',
            'link': 'h3 > a.t::attr(href)',
            'snippet': '.c-abstract::text',
            'title': 'h3 > a.t::text',
            'visible_link': 'span.c-showurl::text'
        },
    }


class DuckduckgoParser(Parser):
    """Parses SERP pages of the Duckduckgo search engine."""

    search_types = ['normal']

    num_results_search_selectors = None

    normal_search_selectors = {
        'results': {
            'container': '#links',
            'result_container': '.result',
            'link': '.result__title > a::attr(href)',
            'snippet': 'result__snippet::text',
            'title': '.result__title > a::text',
            'visible_link': '.result__url__domain::text'
        },
    }


if __name__ == '__main__':
    """Originally part of https://github.com/NikolaiT/GoogleScraper.

    Only for testing purposes: May be called directly with an search engine
    search url. For example:

    python3 parsing.py 'http://yandex.ru/yandsearch?text=GoogleScraper&lr=178&csg=82%2C4317%2C20%2C20%2C0%2C0%2C0'

    Please note: Using this module directly makes little sense, because requesting such urls
    directly without imitating a real browser (which is done in my GoogleScraper module) makes
    the search engines return crippled html, which makes it impossible to parse.
    But for some engines it nevertheless works (for example: yandex, google, ...).
    """
    import requests
    assert len(sys.argv) == 2, 'Usage: {} url'.format(sys.argv[0])
    url = sys.argv[1]
    raw_html = requests.get(url).text
    parser = None

    if re.search(r'^http[s]?://www\.google', url):
        parser = GoogleParser(raw_html)
    elif re.search(r'^http://yandex\.ru', url):
        parser = YandexParser(raw_html)
    elif re.search(r'^http://www\.bing\.', url):
        parser = BingParser(raw_html)
    elif re.search(r'^http[s]?://search\.yahoo.', url):
        parser = YahooParser(raw_html)
    elif re.search(r'^http://www\.baidu\.com', url):
        parser = BaiduParser(raw_html)
    elif re.search(r'^https://duckduckgo\.com', url):
        parser = DuckduckgoParser(raw_html)

    print(parser)

    with open('/tmp/testhtml.html', 'w') as of:
        of.write(raw_html)
```

What you can expect in the near future from GoogleScaper?

I am quite excited to develop some new features for GoogleScraper:

1. Comple documentation and hosting it on readthedocs (https://readthedocs.org/).
2. Asynchroneous support for massive parallel scraping with 1000 proxies and up. I don't know yet what framework to use. Maybe Twisted or something more low level (libevent, epoll, ...)
3. SqlAlchemy integration. I am really excited about that.

4. Cleaner API.
5. Complete configuration for all search engine parameters.
6. Many examples that show how to use GoogleScraper effectively

Many thanks for your patience and time!

Nikolai

This entry was posted in Meta (http://incolumitas.com/category/meta/), Programming (http://incolumitas.com/category/programming/), Python (http://incolumitas.com/category/python/) and tagged baidu (http://incolumitas.com/tag/baidu/), Bing (http://incolumitas.com/tag/bing/), Extracting (http://incolumitas.com/tag/extracting/), google (http://incolumitas.com/tag/google/), scraping (http://incolumitas.com/tag/scraping/), search engine (http://incolumitas.com/tag/search-engine/). Bookmark the permalink (http://incolumitas.com/2014/11/12/scraping-and-extracting-links-from-any-major-search-engine-like-google-yandex-baidu-bing-and-duckduckgo/).

← Using the Python cryptography module with custom passwords (http://incolumitas.com/2014/10/19/using-the-python-cryptography-module-with-custom-passwords/)

Very good program to record audio and desktop on Linux! → (http://incolumitas.com/2015/01/18/very-good-program-to-record-audio-and-desktop-on-linux/)

5 thoughts on “Scraping and Extracting Links from any major Search Engine like Google, Yandex, Baidu, Bing and Duckduckgo”

baishuguoguo **says:**

January 2, 2015 at 12:56 pm (http://incolumitas.com/2014/11/12/scraping-and-extracting-links-from-any-major-search-engine-like-google-yandex-baidu-bing-and-duckduckgo/#comment-581)

Hi nikolai,
Firstly I would like to thank you for sharing this module on github. I am from China and is a beginner of Linux and python, but i am really looking for a tool to make my search work more efficient. So i installed GoogleScraper on my virtual machine.

GoogleScraper v.0.1.14 installed

problem: no data was saved to .db or .csv file.

Details:
1. what i run: GoogleScraper -m http -s “baidu” –keyword “k2 mountain” -o- “test” -f “csv” -v2

2015-01-02 07:47:51,216 – GoogleScraper – INFO – 0 cache files found in .scrapecache/
2015-01-02 07:47:51,216 – GoogleScraper – INFO – 0/1 keywords have been cached and are ready to get parsed. 1 remain to get scraped.
2015-01-02 07:47:51,223 – GoogleScraper – INFO – Going to scrape 1 keywords with 1 proxies by using 1 threads.
2015-01-02 07:47:51,282 – GoogleScraper – INFO – [+] HttpScrape[localhost][search-type:normal] created using the search engine baidu. Number of keywords to scrape=1, using proxy=None, number of pages per keyword=1
2015-01-02 07:47:55,292 – GoogleScraper – INFO – [Thread-1][localhost][baidu]Keyword: “k2 mountain” with 1 pages, slept 4 seconds before scraping. 1/1 already scraped.
2015-01-02 07:47:55,619 – GoogleScraper – INFO – {‘num_results': ‘百度为您找到相关结果约449,000个’, ‘results': []}

***** the chinese means: num_results':’baidu found 449,000 results for you’, ‘results':[]}

the files created:
4.0K -rw-r–r– 1 root root 121 Jan 2 07:47 test.csv
8.0K -rw-r–r– 1 root root 7.0K Jan 2 07:47 test.db

when i open it, it only has heading/1st line. not results stored.

Can you please take some time and help me out. Thank you very much!

Reply to baishuguoguo (/2014/11/12/scraping-and-extracting-links-from-any-major-search-engine-like-google-yandex-baidu-bing-and-duckduckgo/?replytocom=581#respond)

nikolai says:

January 18, 2015 at 2:50 pm (http://incolumitas.com/2014/11/12/scraping-and-extracting-links-from-any-major-search-engine-like-google-yandex-baidu-bing-and-duckduckgo/#comment-584)

This was a bug. It should be long fixed by now. Just always try with the newest GoogleScraper version :)

Reply to nikolai (/2014/11/12/scraping-and-extracting-links-from-any-major-search-engine-like-google-yandex-baidu-bing-and-duckduckgo/?replytocom=584#respond)

Jerry says:

January 8, 2015 at 5:02 pm (http://incolumitas.com/2014/11/12/scraping-and-extracting-links-from-any-major-search-engine-like-google-yandex-baidu-bing-and-duckduckgo/#comment-582)

Hi Nikolai,

I really like the GoogleScraper, such a clever set of workarounds for all kinds of little search engine problems.
However, I do have a question for you: how can I access the number of results from a script? (Or else from the command line?)
When I turn the verbosity up to 4 the num_results is displayed along with a lot of other information, but I'd like to know how to access it more directly, as I would like to do some research on the frequency of certain phrases in search engine results.

I'm sure there is an easy way to do this and I am just overlooking it; I hope you can help.

Reply to Jerry (/2014/11/12/scraping-and-extracting-links-from-any-major-search-engine-like-google-yandex-baidu-bing-and-duckduckgo/?replytocom=582#respond)

nikolai says:

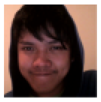
January 18, 2015 at 2:47 pm (http://incolumitas.com/2014/11/12/scraping-and-extracting-links-from-any-major-search-engine-like-google-yandex-baidu-bing-and-duckduckgo/#comment-583)

Hey

I hope you still see this, I am sorry I didn't answer before, stress kills me :D

Look at the this (https://github.com/NikolaiT/GoogleScraper/blob/master/examples/basic.py) example, it shows how to access the number of results of the search query :)

Reply to nikolai (/2014/11/12/scraping-and-extracting-links-from-any-major-search-engine-like-google-yandex-baidu-bing-and-duckduckgo/?replytocom=583#respond)


Cyrus David (https://jcyr.us) says:

February 8, 2015 at 1:17 am (http://incolumitas.com/2014/11/12/scraping-and-extracting-links-from-any-major-search-engine-like-google-yandex-baidu-bing-and-duckduckgo/#comment-595)

Which of the other search engines are much more lenient than google? Sometimes even when I'm not doing something fishy Google detects me doing automated queries even when I'm not.

Reply to Cyrus (/2014/11/12/scraping-and-extracting-links-from-any-major-search-engine-like-google-yandex-baidu-bing-and-duckduckgo/?replytocom=595#respond)

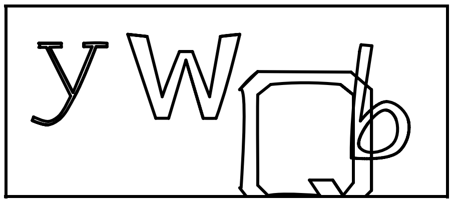
Leave a Reply

Your email address will not be published. Required fields are marked *

Name *

Email *

Captcha *



reload captcha

Website

Comment

You may use these HTML (HyperText Markup Language) tags and attributes:

```
<a href="" title=""> <abbr title=""> <acronym title=""> <b> <blockquote cite=""> <cite> <code class="" title="" data-url=""> <del datet
ime=""> <em> <i> <q cite=""> <strike> <strong> <pre class="" title="" data-url=""> <span class="" title="" data-url="">
```

Post Comment

Search …

RECENT POSTS

- A lot of work to do for GoogleScraper in the future and request for comments! (http://incolumitas.com/2015/03/01/a-lot-of-work-to-do-for-googlescraper-in-the-future-and-request-for-comments/)
- Implementing two Graph traversal algorithms in Python: Depth First Search and Breadth First Search (http://incolumitas.com/2015/01/24/implementing-two-graph-traversal-algorithms-in-python-depth-first-search-and-breadth-first-search/)
- Very good program to record audio and desktop on Linux! (http://incolumitas.com/2015/01/18/very-good-program-to-record-audio-and-desktop-on-linux/)
- Scraping and Extracting Links from any major Search Engine like Google, Yandex, Baidu, Bing and Duckduckgo (http://incolumitas.com/2014/11/12/scraping-and-extracting-links-from-any-major-search-engine-like-google-yandex-baidu-bing-and-duckduckgo/)
- Using the Python cryptography module with custom passwords (http://incolumitas.com/2014/10/19/using-the-python-cryptography-module-with-custom-passwords/)
- Beautiful, beautiful python (http://incolumitas.com/2014/07/11/beautiful-beautiful-python/)
- Lichess.org chess bot! (http://incolumitas.com/2014/04/23/lichess-org-chess-bot/)

THE LATEST THOUGHTS

- Stas on Contact (http://incolumitas.com/about/contact/#comment-600)
- Stas on Contact (http://incolumitas.com/about/contact/#comment-597)
- Mahesh on Contact (http://incolumitas.com/about/contact/#comment-596)
- Cyrus David (https://jcyr.us) on Scraping and Extracting Links from any major Search Engine like Google, Yandex, Baidu, Bing and Duckduckgo (http://incolumitas.com/2014/11/12/scraping-and-extracting-links-from-any-major-search-engine-like-google-yandex-baidu-bing-and-duckduckgo/#comment-595)
- max on Contact (http://incolumitas.com/about/contact/#comment-594)

ARCHIVES

- March 2015 (http://incolumitas.com/2015/03/) (1)
- January 2015 (http://incolumitas.com/2015/01/) (2)
- November 2014 (http://incolumitas.com/2014/11/) (1)
- October 2014 (http://incolumitas.com/2014/10/) (1)
- July 2014 (http://incolumitas.com/2014/07/) (1)
- April 2014 (http://incolumitas.com/2014/04/) (1)
- February 2014 (http://incolumitas.com/2014/02/) (1)
- January 2014 (http://incolumitas.com/2014/01/) (1)
- December 2013 (http://incolumitas.com/2013/12/) (2)
- November 2013 (http://incolumitas.com/2013/11/) (4)
- October 2013 (http://incolumitas.com/2013/10/) (2)
- July 2013 (http://incolumitas.com/2013/07/) (2)
- June 2013 (http://incolumitas.com/2013/06/) (1)
- May 2013 (http://incolumitas.com/2013/05/) (1)
- March 2013 (http://incolumitas.com/2013/03/) (1)
- January 2013 (http://incolumitas.com/2013/01/) (2)
- December 2012 (http://incolumitas.com/2012/12/) (1)
- November 2012 (http://incolumitas.com/2012/11/) (2)
- October 2012 (http://incolumitas.com/2012/10/) (2)
- July 2012 (http://incolumitas.com/2012/07/) (2)

CATEGORIES

- C (http://incolumitas.com/category/c/) (2)
- Chess (http://incolumitas.com/category/chess/) (3)
- Cryptography (http://incolumitas.com/category/cryptography/) (2)
- GoogleScraper (http://incolumitas.com/category/googlescraper/) (1)
- Learning (http://incolumitas.com/category/learn/) (8)
- linux (http://incolumitas.com/category/linux/) (1)
- Meta (http://incolumitas.com/category/meta/) (3)
- Philosophical (http://incolumitas.com/category/philosophical/) (1)
- PHP (http://incolumitas.com/category/php/) (3)
- Programming (http://incolumitas.com/category/programming/) (20)
- Python (http://incolumitas.com/category/python/) (2)
- Security (http://incolumitas.com/category/security/) (10)
- software (http://incolumitas.com/category/software/) (2)
- Uncategorized (http://incolumitas.com/category/uncategorized/) (6)

- University (http://incolumitas.com/category/university/) (1)
- Wordpress (http://incolumitas.com/category/wordpress/) (2)

META

- Log in (http://incolumitas.com/wp-login.php)
- Entries RSS (Really Simple Syndication) (http://incolumitas.com/feed/)
- Comments RSS (Really Simple Syndication) (http://incolumitas.com/comments/feed/)
- WordPress.org (https://wordpress.org/)

META ACTIONS

- Log in (http://incolumitas.com/wp-login.php)
- Entries RSS (Really Simple Syndication) (http://incolumitas.com/feed/)
- Comments RSS (Really Simple Syndication) (http://incolumitas.com/comments/feed/)
- WordPress.org (https://wordpress.org/)

CATEGORIES

- C (http://incolumitas.com/category/c/) (2)
- Chess (http://incolumitas.com/category/chess/) (3)
- Cryptography (http://incolumitas.com/category/cryptography/) (2)
- GoogleScraper (http://incolumitas.com/category/googlescraper/) (1)
- Learning (http://incolumitas.com/category/learn/) (8)
- linux (http://incolumitas.com/category/linux/) (1)
- Meta (http://incolumitas.com/category/meta/) (3)
- Philosophical (http://incolumitas.com/category/philosophical/) (1)
- PHP (http://incolumitas.com/category/php/) (3)
- Programming (http://incolumitas.com/category/programming/) (20)
- Python (http://incolumitas.com/category/python/) (2)
- Security (http://incolumitas.com/category/security/) (10)
- software (http://incolumitas.com/category/software/) (2)
- Uncategorized (http://incolumitas.com/category/uncategorized/) (6)
- University (http://incolumitas.com/category/university/) (1)
- Wordpress (http://incolumitas.com/category/wordpress/) (2)

November 2014

M	T	W	T	F	S	S
					1	2
3	4	5	6	7	8	9
10	11	12 (http://incolumitas.com/2014/11/12/)	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

« Oct (http://incolumitas.com/2014/10/) Jan » (http://incolumitas.com/2015/01/)

Proudly powered by WordPress (http://wordpress.org/) | Theme: clearcontent by incolumitas.com (http://incolumitas.com/).