

Introduction to Whoosh — Whoosh 2.6.0 documentation - Mozilla Firefox

Arquivo Editar Exibir Histórico Favoritos Ferramentas Ajuda

Introduction to Whoosh — Whoosh 2.6.0 do... +

pythonhosted.org/Whoosh/intro.html#about-whoosh

python whoosh

Whoosh 2.6.0 documentation » previous | next | modules | index

## Table Of Contents

- Introduction to Whoosh
  - About Whoosh
  - What is Whoosh?
  - What can Whoosh do for you?
  - Getting help with Whoosh

### Previous topic

Quick start

### Next topic

Glossary

### This Page

Show Source

### Quick search

Enter search terms or a module, class or function name. Go

## Introduction to Whoosh

### About Whoosh

Whoosh was created by [Matt Chaput](#). It started as a quick and dirty search server for the online documentation of the [Houdini](#) 3D animation software package. Side Effects Software generously allowed Matt to open source the code in case it might be useful to anyone else who needs a very flexible or pure-Python search engine (or both!).

- Whoosh is fast, but uses only pure Python, so it will run anywhere Python runs, without requiring a compiler.
- By default, Whoosh uses the [Okapi BM25F](#) ranking function, but like most things the ranking function can be easily customized.
- Whoosh creates fairly small indexes compared to many other search libraries.
- All indexed text in Whoosh must be *unicode*.
- Whoosh lets you store arbitrary Python objects with indexed documents.

### What is Whoosh?

Whoosh is a fast, pure Python search engine library.

The primary design impetus of Whoosh is that it is pure Python. You should be able to use Whoosh anywhere you can use Python, no compiler or Java required.

Like one of its ancestors, Lucene, Whoosh is not really a search engine, it's a programmer library for creating a search engine [1].

Practically no important behavior of Whoosh is hard-coded. Indexing of text, the level of information stored for each term in each field, parsing of search queries, the types of queries allowed, scoring algorithms, etc. are all customizable, replaceable, and extensible.

[1] It would of course be possible to build a turnkey search engine on top of Whoosh, like Nutch and Solr use Lucene.

### What can Whoosh do for you?

Whoosh lets you index free-form or structured text and then quickly find matching documents based on simple or complex search criteria.

### Getting help with Whoosh

## Whoosh – “ranking function”

Okapi BM25 - Wikipedia, the free encyclopedia - Mozilla Firefox

Arquivo Editar Exibir Histórico Favoritos Ferramentas Ajuda

W Okapi BM25 - Wikipedia, the free encyclopedia +

en.wikipedia.org/wiki/Okapi\_BM25 python whoosh

Article Talk Read Edit View history Search

# Okapi BM25

From Wikipedia, the free encyclopedia

In [information retrieval](#), **Okapi BM25** is a [ranking function](#) used by [search engines](#) to rank matching documents according to their [relevance](#) to a given search query. It is based on the [probabilistic retrieval framework](#) developed in the 1970s and 1980s by [Stephen E. Robertson](#), [Karen Spärck Jones](#), and others.

The name of the actual ranking function is BM25. To set the right context, however, it usually referred to as "Okapi BM25", since the Okapi information retrieval system, implemented at [London's City University](#) in the 1980s and 1990s, was the first system to implement this function.

BM25, and its newer variants, e.g. BM25F (a version of BM25 that can take document structure and anchor text into account), represent state-of-the-art [TF-IDF-like](#) retrieval functions used in document retrieval, such as Web search.

## Contents

- The ranking function
- IDF Information Theoretic Interpretation
- Modifications
- Footnotes
- References
- External links

## The ranking function

BM25 is a [bag-of-words](#) retrieval function that ranks a set of documents based on the query terms appearing in each document, regardless of the inter-relationship between the query terms within a document (e.g., their relative proximity). It is not a single function, but actually a whole family of scoring functions, with slightly different components and parameters. One of the most prominent instantiations of the function is as follows.

Given a query  $Q$ , containing keywords  $q_1, \dots, q_n$ , the BM25 score of a document  $D$  is:

$$\text{score}(D, Q) = \sum_{i=1}^n \text{IDF}(q_i) \cdot \frac{f(q_i, D) \cdot (k_1 + 1)}{f(q_i, D) + k_1 \cdot (1 - b + b \cdot \frac{|D|}{\text{avgdl}})},$$

where  $f(q_i, D)$  is  $q_i$ 's [term frequency](#) in the document  $D$ ,  $|D|$  is the length of the document  $D$  in words, and  $\text{avgdl}$  is the average document length in the text collection from which documents are drawn.  $k_1$  and  $b$  are free parameters, usually chosen, in absence of an advanced optimization, as  $k_1 \in [1.2, 2.0]$  and  $b = 0.75$ .<sup>[1]</sup>  $\text{IDF}(q_i)$  is the [IDF \(inverse document frequency\)](#) weight of the query term  $q_i$ . It is usually computed as:

$$\text{IDF}(q_i) = \log \frac{N - n(q_i) + 0.5}{n(q_i) + 0.5},$$

## Whoosh – “ranking function” (cont.)

How to search — Whoosh 2.6.0 documentation - Mozilla Firefox

Arquivo Editar Exibir Histórico Favoritos Ferramentas Ajuda

How to search — Whoosh 2.6.0 documentat... +

whoosh.readthedocs.org/en/latest/searching.html python whoosh

## Scoring and sorting

### Scoring

Normally the list of result documents is sorted by `score`. The `whoosh.scoring` module contains implementations of various scoring algorithms. The default is `BM25F`.

You can set the scoring object to use when you create the searcher using the `weighting` keyword argument:

```
from whoosh import scoring

with myindex.searcher(weighting=scoring.TF_IDF()) as s:
    ...
```

A weighting model is a `WeightingModel` subclass with a `scorer()` method that produces a "scorer" instance. This instance has a method that takes the current matcher and returns a floating point score.

### Sorting

See [Sorting and faceting](#).

## Highlighting snippets and More Like This

See [How to create highlighted search result excerpts](#) and [Query expansion and Key word extraction](#) for information on these topics.

### Filtering results

You can use the `filter` keyword argument to `search()` to specify a set of documents to permit in the results. The argument can be a `whoosh.query.Query` object, a `whoosh.searching.Results` object, or a set-like object containing document numbers. The searcher caches filters so if for example you use the same query filter with a searcher multiple times, the additional searches will be faster because the searcher will cache the results of running the filter query.

You can also specify a `mask` keyword argument to specify a set of documents that are not permitted in the results.

```
with myindex.searcher() as s:
```

## Whoosh – “ranking function” (cont.)

scoring module — Whoosh 2.6.0 documentation - Mozilla Firefox

Arquivo Editar Exibir Histórico Favoritos Ferramentas Ajuda

scoring module — Whoosh 2.6.0 documenta... +

whoosh.readthedocs.org/en/latest/api/scoring.html#module-whoosh.scoring python whoosh

## Scoring algorithm classes

```
class whoosh.scoring.BM25F(B=0.75, K1=1.2, **kwargs)
```

Implements the BM25F scoring algorithm.

```
>>> from whoosh import scoring
>>> # Set a custom B value for the "content" field
>>> w = scoring.BM25F(B=0.75, content_B=1.0, K1=1.5)
```

**Parameters:**

- **B** – free parameter, see the BM25 literature. Keyword arguments of the form `fieldname_B` (for example, `body_B`) set field- specific values for B.
- **K1** – free parameter, see the BM25 literature.

```
class whoosh.scoring.TF_IDF
```

```
class whoosh.scoring.Frequency
```

```
class whoosh.scoring.MultiWeighting(default, **weightings)
```

Chooses from multiple scoring algorithms based on the field.

The only non-keyword argument specifies the default `Weighting` instance to use. Keyword arguments specify `Weighting` instances for specific fields.

For example, to use BM25 for most fields, but `Frequency` for the `id` field and `TF_IDF` for the `keys` field:

```
mw = MultiWeighting(BM25(), id=Frequency(), keys=TF_IDF())
```

**Parameters:** `default` – the `Weighting` instance to use for fields not specified in the keyword arguments.

## Whoosh – “ranking function” (cont.)

The default query language — Whoosh 2.6.0 documentation - Mozilla Firefox

Arquivo Editor Exibir Histórico Favoritos Ferramentas Ajuda

The default query language — Whoosh 2.6... +

pythonhosted.org/Whoosh/querylang.html

python whoosh

Whoosh 2.6.0 documentation » previous | next | modules | index

## Table Of Contents

- The default query language
  - Overview
  - Individual terms and phrases
  - Boolean operators
  - Fields
  - Inexact terms
  - Ranges
  - Boosting query elements
  - Making a term from literal text

### Previous topic

Parsing user queries

### Next topic

Indexing and parsing dates/times

### This Page

Show Source

### Quick search

Enter search terms or a module, class or function name.

## The default query language

### Overview

A query consists of *terms* and *operators*. There are two types of terms: single terms and *phrases*. Multiple terms can be combined with operators such as *AND* and *OR*.

Whoosh supports indexing text in different *fields*. You must specify the *default field* when you create the `whoosh.qparser.QueryParser` object. This is the field in which any terms the user does not explicitly specify a field for will be searched.

Whoosh's query parser is capable of parsing different and/or additional syntax through the use of plug-ins. See [Parsing user queries](#).

### Individual terms and phrases

Find documents containing the term `render`:

```
render
```

Find documents containing the phrase `all was well`:

```
"all was well"
```

Note that a field must store Position information for phrase searching to work in that field.

Normally when you specify a phrase, the maximum difference in position between each word in the phrase is 1 (that is, the words must be right next to each other in the document). For example, the following matches if a document has `library` within 5 words after `whoosh`:

```
"whoosh library"~5
```

## Whoosh – “query language”

## Boolean operators

Find documents containing `render` *and* `shading`:

```
render AND shading
```

Note that AND is the default relation between terms, so this is the same as:

```
render shading
```

Find documents containing `render`, *and* also either `shading` *OR* `modeling`:

```
render AND shading OR modeling
```

Find documents containing `render` but *not* `modeling`:

```
render NOT modeling
```

Find documents containing `alpha` but not either `beta` *OR* `gamma`:

```
alpha NOT (beta OR gamma)
```

Note that when no boolean operator is specified between terms, the parser will insert one, by default AND. So this query:

```
render shading modeling
```

is equivalent (by default) to:

```
render AND shading AND modeling
```

See [customizing the default parser](#) for information on how to change the default operator to OR.

Group operators together with parentheses. For example to find documents that contain both `render` and `shading`, or contain `modeling`:

```
(render AND shading) OR modeling
```

---

## Whoosh – “query language” (cont.)

## Fields

Find the term `ivan` in the `name` field:

```
name:ivan
```

The `field:` prefix only sets the field for the term it directly precedes, so the query:

```
title:open sesame
```

Will search for `open` in the `title` field and `sesame` in the *default* field.

To apply a field prefix to multiple terms, group them with parentheses:

```
title:(open sesame)
```

This is the same as:

```
title:open title:sesame
```

Of course you can specify a field for phrases too:

```
title:"open sesame"
```

## Inexact terms

Use “globs” (wildcard expressions using `?` to represent a single character and `*` to represent any number of characters) to match terms:

```
te?t test* *b?g*
```

Note that a wildcard starting with `?` or `*` is very slow. Note also that these wildcards only match *individual terms*. For example, the query:

```
my*life
```

will **not** match an indexed phrase like:

```
my so called life
```

because those are four separate terms.

---

## Whoosh – “query language” (cont.)

## Ranges

You can match a range of terms. For example, the following query will match documents containing terms in the lexical range from `apple` to `bear` *inclusive*. For example, it will match documents containing `azores` and `be` but not `blur`:

```
[apple TO bear]
```

This is very useful when you've stored, for example, dates in a lexically sorted format (i.e. YYYYMMDD):

```
date:[20050101 TO 20090715]
```

The range is normally *inclusive* (that is, the range will match all terms between the start and end term, *as well* as the start and end terms themselves). You can specify that one or both ends of the range are *exclusive* by using the `{` and/or `}` characters:

```
{0000 TO 0025}  
{prefix TO suffix}
```

You can also specify *open-ended* ranges by leaving out the start or end term:

```
[0025 TO]  
{TO suffix}
```

## Boosting query elements

You can specify that certain parts of a query are more important for calculating the score of a matched document than others. For example, to specify that `ninja` is twice as important as other words, and `bear` is half as important:

```
ninja^2 cowboy bear^0.5
```

You can apply a boost to several terms using grouping parentheses:

```
(open sesame)^2.5 roc
```

## Making a term from literal text

If you need to include characters in a term that are normally treated specially by the parser, such as spaces, colons, or brackets, you can enclose the term in single quotes:

```
path:'MacHD:My Documents'  
'term with spaces'  
title:'function()'
```

## Whoosh – “query language” (cont.)



Parsing user queries — Whoosh 2.6.0 documentation - Mozilla Firefox

Arquivo Editar Exibir Histórico Favoritos Ferramentas Ajuda

Parsing user queries — Whoosh 2.6.0 docu... +

pythonhosted.org/Whoosh/parsing.html

python whoosh

Whoosh 2.6.0 documentation » previous | next | modules | index

## Table Of Contents

- Parsing user queries
  - Overview
  - Using the default parser
  - Common customizations
    - Searching for any terms instead of all terms by default
    - Letting the user search multiple fields by default
    - Simplifying the query language
    - Changing the AND, OR, ANDNOT, ANDMAYBE, and NOT syntax
    - Adding less-than, greater-than, etc.
    - Adding fuzzy term queries
    - Allowing complex phrase queries
  - Advanced customization
    - QueryParser arguments
    - Configuring plugins
    - Creating custom operators

## Parsing user queries

### Overview

The job of a query parser is to convert a *query string* submitted by a user into *query objects* (objects from the `whoosh.query` module).

For example, the user query:

```
rendering shading
```

might be parsed into query objects like this:

```
And([Term("content", u"rendering"), Term("content", u"shading")])
```

Whoosh includes a powerful, modular parser for user queries in the `whoosh.qparser` module. The default parser implements a query language similar to the one that ships with **Lucene**. However, by changing plugins or using functions such as `whoosh.qparser.MultifieldParser()`, `whoosh.qparser.SimpleParser()` or `whoosh.qparser.DisMaxParser()`, you can change how the parser works, get a simpler parser or change the query language syntax.

(In previous versions of Whoosh, the query parser was based on `pyparsing`. The new hand-written parser is less brittle and more flexible.)

**Note:** Remember that you can directly create query objects programmatically using the objects in the `whoosh.query` module. If you are not processing actual user queries, this is preferable to building a query string just to parse it.

## Common customizations

<http://pythonhosted.org/Whoosh/parsing.html>

## Whoosh – “query parser”