WikipediaBase

Chris Perivolaropoulos

Sunday 21 February 2016

Contents

WikipediaBase is a backend to START responsible for providing access to wikipedia related information, that mimics the API interface provided by the Omnibase. Wikipediabase has gone through a couple of rewrites. The initial version was written in Java. It was then rewritten in Ruby copying the original architecture and design and now it is rewritten in python and rearchitectured from scratch. There are two main reasons for this: python is taught as a pre-graduate course in MiT, and therefore a Python codebase will make initiation of new MiT students smoother and, more importantly, while the initial design of the previous WikipediaBase should have been adequate, it grew to a point where the code was ad-hoc and hard to understand, reason about and extend.

The python implementation was initially written by Chris Perivolaropoulos in close association with Dr Sue Felshin and was eventually handed over to Sue Felshin, Alvaro Morales and Michael Silver. Other students have also joined the project shortly after.

1 Functionality

As far as the ontology that START assumes for WikipediaBase, each infobox type corresponds to a START class and each valid infobox attribute is a START class attribute. Furthermore all such classes inherit from the `wikipediabase-term` START class which supports the following attributes:

- `IMAGE-DATA` : The infobox image
- `SHORT-ARTICLE` : A short version of the article, typically the first paragraph

- `URL` : The url of the article
- `COORDINATES` : Wherever it makes sense, the coordinates of the concept of the article
- `PROPER` : Whether the article refers to a proper noun (eg The Beatles, United States etc)
- `NUMBER` : `#t` if the concept of the article refers to many things, `#f` if it refers to one.

All commands and return values are encoded into s-expressions.

1.1 get

Given a class, object name, and typed attribute, return the value as a lisp-readable form.

Valid attribute typecodes are

- `:code` for an attribute name as in infobox wiki markup format
- and `:rendered` for an attribute name in the rendered form of the infobox.

1. Types

   All return values of `get` are typed. Below is a comprehensive list of all the supported types.

   (a) `:HTML`

   A string suitable for rendering as paragraph-level HTML. The string must be escaped for lisp, meaning double quoted, and with double quotes and backslashes escaped with backslashes. For example:

```
(get "wikipedia-sea" "Black␣Sea" (:code "AREA
   "))
=> ((:html "436,402␣km2␣(168,500␣sq␣mi)"))

(get "wikipedia-president" "Bill␣Clinton" (:
   code "SUCCESSOR"))
=> ((:html "George␣W.␣Bush"))
```

```
(get "wikipedia-president" "Bill␣Clinton" (:
   rendered "Succeeded␣by"))
=> ((:html "George␣W.␣Bush"))
```

(b) `:YYYYMMDD`

Parsed dates are represented in the format `[-]<4 digit year><2 digit month><2 digit day>`. Unparsable dates are represented as `:html` types

```
(get "wikibase-person" "Barack␣Obama" (:ID "
   BIRTH-DATE"))
=> ((:yyyymmdd 19610804))

(get "wikibase-person" "Julius␣Caesar" (:ID "
   BIRTH-DATE"))
=> ((:YYYYMMDD -1000713))
```

(c) `:CALCULATED`

The type of calculated properties based on characteristics of the article, e.g., *GENDER* and *NUMBER*. See below under Special Attributes for a complete list of calculated attributes.

(d) `:CODE`

Deprecated, old synonym for `:HTML`.

(e) `:STRING`

Deprecated, old synonym for `:HTML`.

2. Special Attributes

Besides the attributes that are fetched as attributes of the infobox, the rest of the available attributes are special in that they are calculated from the contents of the article. They are also special in that they are hardocoded, ie the value of the attribute is calculated, not the attribute itself. These attributes should be specific to `wikibase-term`, `wikibase-person`, and `wikipedia-paragraphs`.

(a) `SHORT-ARTICLE`, `wikibase-term`

The first paragraph of the article, or if the first paragraph is shorter than 350 characters, then the value of `short-article` is the the first paragraphs such that the sum of the rendered characters is at least 350.

(b) `URL`, `wikibase-term`

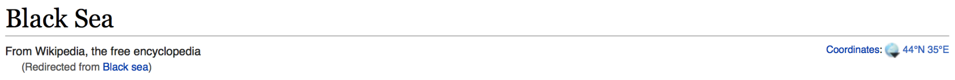
   The URL of the article as `((:url URL))`

(c) `IMAGE-DATA`, `wikibase-term`

   A list of URLs for images in the article content (excludes images that are in the page but outside of the article content). The "best" image should be the first URL in the list; if there is a picture at the top of the infobox, this is considered to be the best image, or otherwise the first image that appears anywhere in the article. If there is no caption, the caption value should be omitted, e.g., `((0 "Harimau_Harimau_cover.jpg"))` rather than `((0 "Harimau_Harimau_cover.jpg" ""))`.

(d) `COORDINATES`, `wikibase-term`

   Computed from latitude and longitude attributes given in the article header or, if none can be found, the infobox. The value is a list of the latitude and longitude, e.g., ((`:coordinates` latitude longitude))


Black Sea

From Wikipedia, the free encyclopedia

(Redirected from Black sea)

Coordinates: 44°N 35°E

Figure 1: An example of coordinates in the header

(e) `BIRTH-DATE`, `wikibase-person`

   Searches in order using `dateparser` and type `:yyyymmdd` and relying on the first valid occurrence of a date in the following:

   - The infobox attribute `birth date`
   - In the first sentence of the article look for `born {date}`
   - In the first parentheses of the article look for a date range and use the lower bound of the range.

   If a date is detected but cannot be parsed then the attribute's value has type `:html`

(f) `DEATH-DATE`, `wikibase-person`

   Fetched similarly to `BIRTH-DATE`. Returns the same value types as BIRTH-DATE, except if the person is still alive, throws an error with the reply "Currently alive".

(g) `GENDER`, `wikibase-person`
Computed from the page content based on heuristics such as the number of times that masculine vs. feminine pronouns appear. Valid values are `:masculine` and `:feminine`.

(h) `NUMBER`, `wikibase-term`
Computed from the page content based on heuristics such as number of times the page's title appears plural. Valid for all objects. Returns `#t` if many, `#f` if one.

(i) `PROPER`, `wikibase-term`
Computed from the page content based on heuristics such as number of times the page's title appears capitalized when not at the start of a sentence. Valid for all objects. Returns `#t` if proper and `#f` if not.

1.2 get-classes

Given an object name, return a list of all classes to which the object belongs. Class names are conventionally given in lower case, but this is not an absolute requirement. E.g.,

```
(get-classes "Cardinal␣(bird)")
=> ("wikibase-term" "wikipedia-paragraphs" "
   wikipedia-taxobox")

(get-classes "Hillary␣Rodham␣Clinton")
=> ("wikibase-term"
"wikipedia-paragraphs"
"wikibase-person"
"wikipedia-officeholder"
"wikipedia-person")
```

1.3 get-attributes

Given a class name, return a list of all attributes that the class implements. If possible also provide the typecode of the value type and the human readable form, ie the rendered attribute from the wikipedia infobox

```
(get-attributes "wikipedia-officeholder" "Barack␣
   Obama")
=> ((:CODE "TERM_END3" :VALUE :YYYYMMDD) ...)
```

1.4 `sort-symbols`

Given any number of symbols `sort-symbols` will sort them into subsets by the length of the associated article. E.g.,

```
(sort-symbols  "Obama␣(surname)" "Barack␣Obama")
=> (("Barack␣Obama") ("Obama␣(surname)"))
```

1.5 `sort-symbols-named`

`sort-symbols-named` takes a synonym and any number of symbols and sorts the symbols into subsets; if any symbol name is the same as the synonym, it and its subset are sorted to the front. E.g.

```
(sort-symbols-named
 "cake"
 "Cake␣(TV␣series)"
 "Cake␣(firework)"
 "Cake␣(film)"
 "Cake␣(drug)"
 "Cake"
 "Cake␣(band)"
 "Cake␣(advertisement)"
 "The␣Cake")
=> (("Cake")
("Cake␣(band)")
("Cake␣(advertisement)")
("Cake␣(TV␣series)")
("The␣Cake")
("Cake␣(film)")
("Cake␣(firework)")
("Cake␣(drug)"))
```

2 Getting started

The entire WikipediaBase resides in a git repository in infolab's github orginization page

```
git clone git@github.com:infolab-csail/WikipediaBase
```

WikipediaBase depends on multiple other python packages. Fortunately, python is shipped not only with a great package manager, but also with a mechanism called virtualenv that isolates installations of a project's dependencies from the rest of the system, thus avoiding problems like version or namespace collisions. The way this effectively works is that the global python installation is half copied half symlinked to a local directory and the dependencies are installed only in the local sandbox. To create and activate a python virtualenv:

```
$ virtualenv --no-site-packages py
$ . py/bin/activate
$ which python
/the/local/directory/py/bin/python
```

Now that we can safely install anything we want without breaking any global installation

```
pip install -r requirements.txt
```

We will need some extra stuff for WikipediaBase to work:

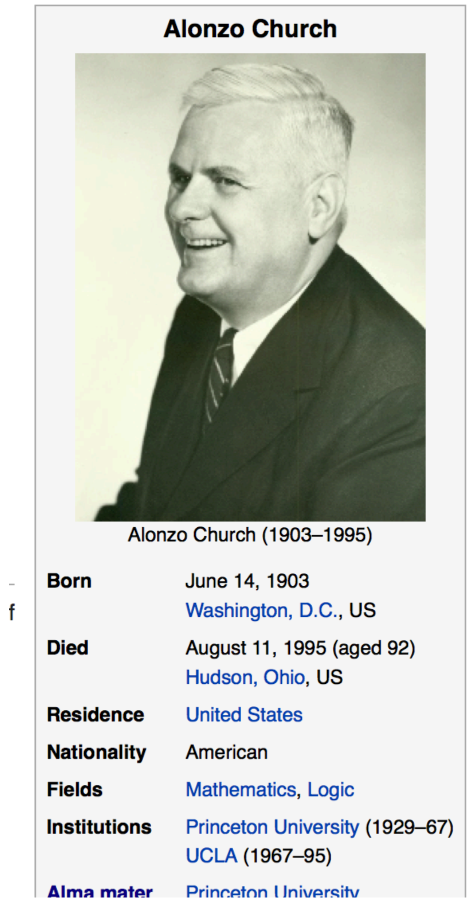
- Postresql
- Redis

The installation process of these packages varies across platforms. Both are databases. Their purpose is for caching repeated computations and for storing ahead-of-time computation like infobox markup, name to rendered name maps, and synonyms.

3 Architecture

3.1 Infobox

Infoboxes are tables that are commonly used in wikipedia to provide an overview of the information in an article in a semi structured way. Infoboxes are the main source of information for WikipediaBase.



Alonzo Church

Alonzo Church (1903–1995)

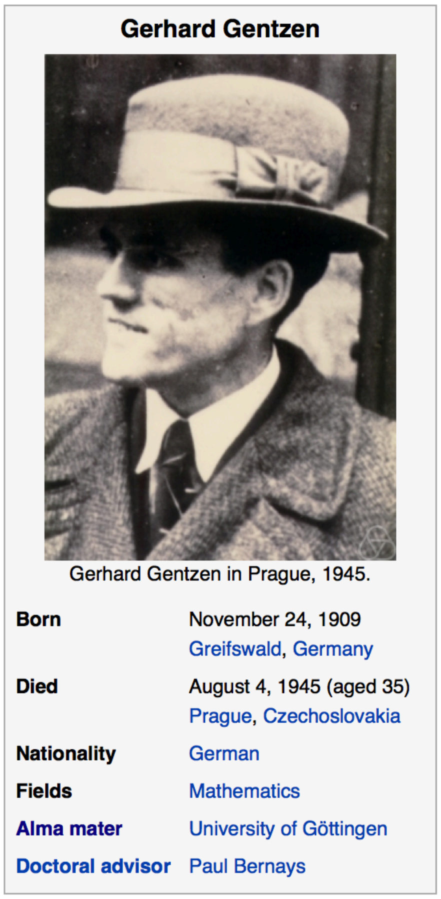
Born	June 14, 1903 Washington, D.C., US
Died	August 11, 1995 (aged 92) Hudson, Ohio, US
Residence	United States
Nationality	American
Fields	Mathematics, Logic
Institutions	Princeton University (1929–67) UCLA (1967–95)
Alma mater	Princeton University

Figure 2: An example of an infobox

In mediawiki markup terms an infobox is a markup template with a type that gets rendered into html so that the provided information makes sense in the context that it is provided. For example:

```
{{Infobox scientist
| name                = Gerhard Gentzen
| image               = Gerhard Gentzen.jpg
| image_size          =
| alt                 =
| caption             = Gerhard Gentzen in Prague,
  1945.
| birth_date          = {{Birth date|1909|11|24}}
| birth_place         = [[Greifswald]], [[Germany]]
| death_date         = {{Death date and age
   |1945|8|4|1909|11|24}}
| death_place        = [[Prague]], [[Czechoslovakia]]
| nationality        = [[Germany|German]]
| fields             = [[Mathematics]]
| workplaces         =
| alma_mater         = [[University of Gottingen]]
| doctoral_advisor   = [[Paul Bernays]]
| doctoral_students =
| known_for          =
| awards             =
}}
```

will yield:



Gerhard Gentzen

Gerhard Gentzen in Prague, 1945.

Born	November 24, 1909 Greifswald, Germany
Died	August 4, 1945 (aged 35) Prague, Czechoslovakia
Nationality	German
Fields	Mathematics
Alma mater	University of Göttingen
Doctoral advisor	Paul Bernays

Figure 3: An example of an infobox

Infobox types are organized into a fairly wide hierarchy. For example Template:Infobox Austrian district is a special case of a Template:Infobox settlement and each is rendered differently. For our purposes, and to mirror the markup definition of infoboxes, an infobox I with attributes a_i and values v_i is a set of pairs (a_i, v_i) together with a infobox type t. Each attribute a_i and value v_i have two forms:

- a rendered form, a_i^r and v_i^r respectively, which is the rendered HTML representation and
- a markup form, a_i^m and v_i^m which is the mediawiki markup code that corresponds to them.

An article may have more than one infoboxes, for example Bill Clinton article has both Infobox Officeholder and Infobox President infoboxes.

The `Infobox` class is the basic data type for accessing information from the infobox of an article. `Infobox`, as well as `Article`, are what one would use were they to use wikipediabase as a python library. The methods provided by an infobox are:

types Because we retrieve an infobox based on a symbol name (ie page name), a single `Infobox` may actually be an interface for multiple infoboxes. There is a separate method, based on this one, for getting types in a format suitable for START.

Value access is possible provided either a_i^r or a_i^m.

Rendered keys are provided using the `MetaInfobox` (see below).

Infobox export to python types, namely:

- dict for $a_i^r \to v_i^r$ or $a_i^m \to v_i^m$
- the entire infobox rendered, or in markup form.

Infoboxes are organized in a wide hierarchy that in the WikiepdiaBase codebase is refered to as infobox tree. The infobox tree is retrieved from the list of infoboxes wikipedia page and is used to deduce the ontology of wikipedia terms.

3.2 MetaInfobox

The `MetaInfobox` is implemented as a subclass of the `Infobox` that provides information about the infobox, most importantly a map between markup attributes and the rendered counterparts. Say we have an infobox of type I which has attributes $a_1, ..., a_n$. Each attribute has two representations:

- a markup representation that is the key used in the infobox template.
- The HTML rendered representation, that is the left-column text of the rendered infobox table.

For example in the `officeholder` infobox there is an attribute that has markup representation `predecessor` and rendered representation `Preceded by`.

To do this the `MetaInfobox` uses the template documentation page to find the markup representation of all valid attributes of an infobox type. It then creates an infobox where each attribute has value its markup attribute name wrapped int `!!!`. (for example `predecessor = !!!predecessor!!!`). It then renders the created infobox and looks for `!!!predecessor!!!` in the rendered values. The corresponding rendered attribute names also correspond to the markup attribute names. Note that the correspondence of rendered - markup attributes is not bijective, that is to say each markup attribute may correspond to zero or more rendered attributes and vice versa.

For example, an infobox of type `Foo` has valid attributes *A*, *B*, *C* and *D*. The generated infobox markup would be:

```
{{Infobox Foo
| A = !!!A!!!
| B = !!!B!!!
| C = !!!C!!!
| D = !!!D!!!
}}
```

And the rendered version could be, depending on the implementation of the `Foo` infobox.

Attribute	Value
A	!!!A!!! !!!B!!! !!!C!!!
B	!!!A!!! !!!B!!! !!!C!!!
C	!!!A!!! !!!B!!! !!!C!!!
D	!!!D!!!

Which makes the mapping fairly obvious.

3.3 Article

The `Article` data structure is responsible for accessing any resource relevant to the article at large. This includes paragraphs, headings, markup source and the mediawiki categores.

3.4 Fetcher

The fetcher is an abstraction over the communication of WikipediaBase with the outside world. It is a singleton object that implements a specific interface.

Fetchers are organized in an inheriting hierarchiy

BaseFetcher The baseclass for fetchers, it will return the symbol instead of trying to resolve it in any way

Fetcher contains the core functionality of a a fetcher. It will fetch articles from *wikipedia.org*. It is possible to direct it to a mirror but wikipedia-mirror's runtime performance turned out to be prohibitive.

CachingFetcher inherits fetcher and retains its functionality, only it uses Redis to cache the fetched symbols. It is the default fetcher for wikipediabase.

StaticFetcher is a class that implements the `BaseFetcher` interface but instead of reaching out to some data source for the data the return values are statically defined. It is used most notably by `MetaInfobox` to use the `Infobox` functionality to convey arbitrary information.

By default, markup is fetched from the backend. If $\text{force}_{\text{live}}$ is set to True, the markup will be fetched from live wikipedia.org

When tests are ran on TravisCI, we always want to use live data. We check if Travis is running tests by looking at the $\text{WIKIPEDIABASE}_{\text{FORCELIVE}}$ env variable.

3.5 Renderer

Renderers are singleton classes that are useful for rendering mediawiki markup into HTML. Originally the wikiepedia sandbox was used by wikipediabase for rendering pages because it is slightly faster than the API, but the wikipedia-mirror was really slow at this and wikipedia.org would consider it an abuse of the service and block our IP. For that reason we eventually switched to the API with Redis caching, which works out pretty well because `Renderer` objects end up being used only by `MetaInfobox` which has quite a limited scope, making thus cache misses rarely.

An interesting anecdote about the `Renderer` class was that it was the reason for a couple of CSAIL IPs to get temporarily banned from editing wikipedia. While wikipedia.org has a very lenient policy when it comes to banning people who are spamming their servers, repeated testing of the `Renderer` class targeting wikipedia's sandbox caused the testing machine's ip to be temporarily banned on the grounds that "its activity does not promote the imporovement of wikipedia". We reimplemented the `Renderer` to use the wikipedia API and we never had a problem with wikipedia moderation again.

3.6 Pipeline

When resolving a query WikipediaBase employs a pipeline of modules to figure out what the best way to respond would be.

1. Frontend

   WikipediaBase can be used as a library but its primary function is as a backend to START. The communication between START and WikipediaBase is carried out over a plaintext telnet connection on port 8023 using s-expressions. The frontend handles the network connection with START, translates the received queries into calls to knowledgebase and then translates the knowledgebase response into properly formulated s-expressions that it sends back over the telnet connection.

2. Knowledgebase

   The knowledgebase is the entry point to the rest of wikipediabase. It uses the Provider/Acquirer pattern to transparently provide the frontend with arbitrary methods. Those methods are responsible for chosing whether we are to resort to classifiers or resolvers (or any other mechanism) for answering the query. Available classifiers and resolvers become accessible to the knowledgebase automatically using their base class.

3. Classifiers

   Each classifier is a singleton that implements a heuristic for deducing a set of classes of an object. An object may inhibit zero or more classes. There are a couple classifiers available at the moment. Typically a classifier will only deduce whether an object actually inhibits a specific class or not but that is not necessary.

   (a) Term

   The `TermClassifier` simply assigns the `wikipedia-term` class. Wikipediabase only deals with wikipedia related information.

   (b) Infobox

   The `InfoboxClassifier` assigns to a term the classes of the infobox. For example Bill Clinton's page contains the infobox:

   ```
   {{Infobox president
   |name          = Bill Clinton
   |image         = 44 Bill Clinton 3x4.jpg{{!}}border
   [...]
   }}
   ```

   And therefore gets the class `wikipedia-president`.

(c) Person

`PersonClassifier` assigns the class `wikibase-person` using a few heuristics in the order they are described:

i. Category regexes

   Use the following regular expressions to match categories of an article.

   - `.* person`
   - `^\d+ deaths.*`
   - `^\d+ births.*`
   - `.* actors`
   - `.* deities`
   - `.* gods`
   - `.* goddesses`
   - `.* musicians`
   - `.* players`
   - `.* singers`

ii. Category exclude regexes

   Exclude categories matching the following regexes.

   - `\sbased on\s`
   - `\sabout\s`
   - `lists of\s`
   - `animal\`

iii. Category matches

   We know an article refers to a person if the page is in one or more of the following mediawiki categories :

   - `american actors`
   - `american television actor stubs`
   - `american television actors`
   - `architects`
   - `british mps`
   - `character actors`
   - `computer scientist`
   - `dead people rumoured to be living`
   - `deities`
   - `disappeared people`

- `fictional characters`
- `film actors`
- `living people`
- `musician stubs`
- `singer stubs`
- `star stubs`
- `united kingdom writer stubs`
- `united states singer stubs`
- `writer stubs`
- `year of birth missing`
- `year of death missing`

For an example of how this works see the appendix.
As it is obvious the list of categories is arbitrary and very far from complete. Multiple methods have been considered for fixing this. Some of them are:

- Supervised machine learning methods like SVM using other methods of determining person-ness to create training sets.
- Hand-pick common categories for person articles determined again with the other criteria

4. Resolvers

Resolvers are also singletons but their purpose is to find the value of the requested property. All resolvers descend from `BaseResolver` and should implement the following methods:

- `resolve(class, symbol, attribute)`: get the value of the `attribute` of symbol `symbol` as `class`
- `attributes(class, symbol)`: get a list of the attributes this resolver can resolve.

The implemented resolvers are the following:

Error the minimum priority resolver, it will always resolve to an error.

Infobox Resolve attributes found on infoboxes of a symbol.

Person resolve the following specific attributes of symbols referring to people:

- `birth-date`
- `death-date`
- `gender`

Sections resolve the content of sections in an article.

Term Can resolve a fixed set of ad-hoc attributes:

- `coordinates` *The coordinates of a geographical location*
- `image` *The image in the infobox*
- `number` *True if the symbol is plural (eg The Beatles)*
- `proper` *True if it refers to a unique entity.*
- `short-article` *A summary of the article. Typically the first paragraph*
- `url` *The article url*
- `word-cout` *The size of the article*

5. Lisp types

   Lisp type instances are wrappers for python objects or values that are presentable in s-expression form that START can understand. They are created either from the raw received query and unwrapped to be useful to the pipeline, or by the answer WikipediaBase comes up with and then encoded into a string sent over telnet to START.

4 Provider/Acquirer model

WikipediaBase attempts to be modular and extendible. To accomplish this it is often useful to multiplex multiple sources of the same type of data resource. This is particularly useful when accessing heuristic methods like classifier. To promote modularity and to avoid hard dependencies the provider/acquirer model was created:

A `Provider` is an object through which we can access resources that are stored in a key-value fashion. The `Provider` class offers facilities like decorators to make this provision easy. An `Acquirer` has transparent access to the resources of multiple `Providers` as if they were a single key value store. This pattern is most notably used for the `KnowledgeBase` to provide the `Frontend` with the way of accessing resources.

4.1 Example

We demostrate the pattern with an example: we will embed a small lisp dialect into python that we will call `p-lisp` (for `python lisp`, `provider-lisp` and `poor-lisp`)

```
from wikipediabase.provider import Provider, Acquirer
    , provide


class EvalContext(Acquirer):
    def __init__(self, closures):
        super(EvalContext, self).__init__(closures)
        self.closures = closures

    def __call__(self, _ctx, expr):
        if isinstance(expr, list):
            # Handle quotes
            if expr[0] is 'quote':
                return expr[1]

            # Call the lambda
            fn = self(_ctx, expr[0])
            return fn(self, *[self(_ctx, e) for e in
                expr[1:]])

        if isinstance(expr, basestring) and expr in
            self.resources():
            return self(_ctx, self.resources()[expr])

        return expr


class Lambda(Acquirer):
    def __init__(self, args, expr, env):
        # Get your symbols from all the available
            closures plus an
        # extra for local variables
        super(Lambda, self).__init__([env] + [Symbols
            ()])
        self.args = args
        self.expr = expr

    def __call__(self, _ctx, *args):
        # Add another closure to the list
        arg_provider = Provider();
        for s, v in zip(self.args, args):
            arg_provider.provide(s, v)

        # Build an eval context and run it
        ctx = EvalContext([arg_provider, Provider(
           self.resources())])
        return [ctx(ctx, e) for e in self.expr][-1]

class Symbols(Provider):
    @provide('setq')
    def setq(self, ctx, symbol, val):
        self.provide(symbol, val)

class Builtins(Provider):
    @provide('lambda')
    def _lambda(self, ctx, args, *body):
        return Lambda(args, list(body), Provider(ctx.
           resources()))

    @provide('if')
    def _if(self, ctx, proposition, then, _else):
        if ctx(ctx, proposition):
            return ctx(ctx, then)
        else:
            return ctx(ctx, _else)

GLOBAL_EVAL = EvalContext([Builtins(), Symbols()])
```

`p-lisp` supports:

- lambdas
- A global symbol table
- lexical scoping
- conditionals
- Quoted literals

It really is very far from being remotely close to a usable language but it can do some cute tricks:

We can evaluate python types:

```
>>> GLOBAL_EVAL({}, 1)
1
>>> GLOBAL_EVAL({}, True)
True
>>> GLOBAL_EVAL({}, "hello")
'hello'
>>> GLOBAL_EVAL({}, list)
<type 'list'>
```

We can define lambdas and call them. The following is equivalent to $(\lambda a.a)1$, which should evaluate to `1`:

```
>>> GLOBAL_EVAL({}, [["lambda", ['quote', ['a']], 'a'
    ], 1])
1
```

Our little lisp is not pure since we have a global symbol table. The best way to sequence expressions is to wrap them all up in a `lambda` and then evaluate that:

```
>>> GLOBAL_EVAL({}, [['lambda', ['quote', []], ['setq
    ', 'b', 2], 'b']])
2
```

The attentive reader may have noticed the quoted list for lambda arguments. The reason is that we do not want the list to be evaluated.

Back on our main subject, in `p-lisp` symbols get values from 3 different sources:

- The local closure
- The arguments of the lambda
- Builtin functions

All the above are abstracted using the provider-aquirer model. At each point a different `EvaluationContext` is responsible for evaluating and each `EvaluationContext` has access to its known symbols via an array of providers that are abstracted using the discussed model.

5 Testing

5.1 Unit testing

The good functioning of WikipediaBase is assured by a comprehensive test suite of unit tests, functional tests and regression tests.

1. Unit tests

   Unit tests test small blocks of functionality, that are composed to create the system at large. For unit testing we use python's default testing library. Each test is a class that implements the `TestCase` interface.

2. Functional and regression tests

   Functional tests are tests written before, during or shortly after the development of a system and they assert the correct overall functioning of the system. Regression tests are very akin to functional tests. They prove that a found bug was fixed and assert that it will not appear again later. Functional and regression tests currently reside in `tests/examples.py`

5.2 Examples

Virtually all tests begin with the following snippet:

```
from __future__ import unicode_literals

try:
    import unittest2 as unittest
except ImportError:
    import unittest

from wikipediabase import fetcher
```

The above is specific for the `fetcher` module. As is apparent we are using the `unittest` module from the standard python library. The test itself has the following format:

```
class TestFetcher(unittest.TestCase):

    def setUp(self):
        self.fetcher = fetcher.get_fetcher()
    def test_html(self):
        html = self.fetcher.html_source("Led␣Zeppelin
            ")
        self.assertIn("Jimmy␣Page", html)
```

The `setUp` method runs before each test of the `TestCase`. Tests of the testcase are represented by methods of the class whose name begins with `test_`. In this particular case we are getting the wikipedia page for Led Zeppelin and making sure the name of Jimmy Page is mentioned at least once. This is obviously not conclusive that fetcher did not for example bring up the page for *The Yardbirds*, Page's first band. For this reason we write a couple of these sort of tests. For the entire test see the Python test example in the appendix.

5.3 Running tests

We employ the `nosetests` tool to find and run our tests. To do so we add a test requirement in `setup.py` and assign `nose.collector` to manage our test suite:

```
from setuptools import setup

setup(
    tests_require=[
        'nose>=1.0',
        ...
    ],
    ...
    test_suite='nose.collector',
    ...
)
```

Then to run the tests

```
$ python setup.py test
```

Nose will find all files that are in `tests/` and have the prefix `test_`, for example `test_fetcher.py`. Inside those files nose looks into classes that subclass `TestCase` and whose name begins with `Test`, for example `TestFetcher`. It then runs all methods of the collected classes that have the `test_` prefix.

It is also possible to run specific tests.

```
$ python setup.py test --help
Common commands: (see '--help-commands' for more)

  setup.py build      will build the package
     underneath 'build/'
  setup.py install    will install the package

Global options:
  --verbose (-v)  run verbosely (default)
  --quiet (-q)    run quietly (turns verbosity off)
  --dry-run (-n)  don't actually do anything
  --help (-h)     show detailed help message
  --no-user-cfg   ignore pydistutils.cfg in your home
      directory

Options for 'test' command:
  --test-module (-m)  Run 'test_suite' in specified
     module
  --test-suite (-s)   Test suite to run (e.g. '
     some_module.test_suite')
  --test-runner (-r)  Test runner to use

usage: setup.py [global_opts] cmd1 [cmd1_opts] [cmd2
   [cmd2_opts] ...]
   or: setup.py --help [cmd1 cmd2 ...]
   or: setup.py --help-commands
   or: setup.py cmd --help
```

See the appendix for the full output of a successful test run.

6 Synonyms

Before we talk about synonyms it is important to concretely define symbols in the context of the omnibase universe:

> Symbols are identifiers of "objects" in a data source. (The term "symbol" is unfortunate, since it has so many meanings in computer science, but we're stuck with it for historical reasons.)

Since language tends to have multiple ways of referring to the same things, defining aliases for symbols is imperative.

> Synonyms are names which users can use to refer to symbols. (The term "synonym" is unfortunate, because this is really a one-way mapping -"gloss" would be a better term but we're stuck with "synonym" for historical reasons.)

The definition of synonyms is the job of the backend itself. Therefore it is the job of WikipediaBase to define the set of synonyms required.

6.1 Good/Bad synonyms

There are rules to what is considered a good and what a bad synonym. In short synonyms:

- Should not lead with articles ("the", "a", "an")
- Should not lead with "File:" or "TimedText:".
- Should not fragment anchors. Eg "Alexander$_{\text{Pushkin}}$#Legacy"
- Should not start with the following:
  - "List of "
  - "Lists of "
  - "Wikipedia: "
  - "Category: "
  - ":Category: "
  - "User: "
  - "Image: "
  - "Media: "
  - "Arbitration in location"
  - "Communications in location"
  - "Constitutional history of location"
  - "Economy of location"
  - "Demographics of location"
  - "Foreign relations of location"
  - "Geography of location"
  - "History of location"

  - "Military of location"
  - "Politics of location"
  - "Transport in location"
  - "Outline of topic"

- Should not match `\d\d\d\d in location` or `location in \d\d\d\d`
- Should not be names of disabiguation pages. To make this inclusive for all relevant pages, including typos, that means symbols that match `\([Dd]isambig[^)]*\)`
- Synonyms that both a) could be mistaken for ones that start with articles and b) might subsume something useful. That means that for example "A. House" (synonym of "Abraham House") is disqualified because it might mislead START in the case of questions like "How much does a house cost in the Silicon Valley?". On the other hand "a priori" can be kept because there are no sensible queries where "a" is an article before "priori".

6.2 Synonym generation

To accommodate these restrictions two approaches are employed: disqualification of synonym candidates and modification of synonym candidates. Modification is attempted first, and if that fails we disqualify. The rules for modification are as follows:

- Strip determiners (articles) that are at the beginning of a synonym (or would be at the beginning if not for punctuation):
  - "A "
  - "An "
  - "The "
  - '(The) '
  - The&nbsp;
  - etc.
- Generate both versions, with and without paren. Eg given symbol "Raven (journal)" generate both:
  - "Raven (journal)"

  - "Raven"

- Generate before and after slash, but not the original symbol, e.g.:
  - Given symbol "Russian language/Russian alphabet" generate both
    * "Russian language"
    * "Russian alphabet"

- Reverse inverted synonyms with commas. Eg given synonym "Congo, Democratic Republic Of The" invert it to get "Democratic Republic Of The Congo"

- As usual, get rid of leading articles if necessary. Eg given synonym "Golden ratio, the" replace it with "the Golden ratio", then strip articles to get: "Golden ratio" same goes for a, an, etc.

This way we generate an initial set of synonyms from the name of the object itself. Furthermore we can generate a set of synonyms from wikipedia redirects to the article. Wikipedia kindly provides an SQL dump for all redirects.

To load the table, in your database where you have loaded the wikipedia data, you should load the redirects table:

```
wget https://dumps.wikimedia.org/enwiki/latest/enwiki
    -latest-redirect.sql.gz \
   -O redirect.sql.gz && gzcat redirect.sql.gz | mysql
```

And then from the SQL db to find all (good and bad) synonyms to Bill Clinton you can run

```
select page_title, rd_title from redirect join page
    on rd_from = page_id and (rd_title = "Bill_Clinton
    " or page_title = "Bill_Clinton");
```

For the full output see the appendix.

7 Databases and data sources

Wikipediabase uses primarily a remote data store that implements the mediawiki HTTP interface and attempts to deal with the arising performance issues by aggressively caching pages to a backend key-value based database.

The interface with the database is abstracted by using a python-style dictionary interface, which is implemented in `persistentkv.py`. Implemented backends are presented below, but it is trivial to provide any backend one can come up with.

7.1 Data access

Data access is abstraced by the ad-hoc `Fetcher` interface. Currently the only useful fetcher implemented is the `CachingSiteFetcher` that supports retrieval of both mediawiki markup and rendered HTML for each page while caching all retrieved data.

1. HTML and MediaWiki API

   The initial approach to getting the data is to retrieve the normal HTML versions of wikipedia articles and using edit pages to retrieve the mediawiki markup. We invariably use the original wikipedia.org site for performance reasons (See wikipedia-mirror runtime performance section).

   Mediawiki provides a RESTful API for all the required functionality. The basic premise is that one can send requests with `POST` or `GET` methods and get a response formulated in XML or JSON. The preferred response type for WikipediaBase was sending `GET` HTTP requests to receive `JSON` data. `GET` was selected because it is explicitly suggested in the mediawiki API page because caching happens at the HTTP level.

   > Per the HTTP specification, POST requests cannot be cached. Therefore, whenever you're reading data from the web service API, you should use GET requests, not POST.
   >
   > Also note that a request cannot be served from cache unless the URL is exactly the same. If you make a request for api.php?....titles=Foo|Bar|Hello, and cache the result, then a request for api.php?....titles=Hello|Bar|Hello|Foo will not go through the cache even though MediaWiki returns the same data!

   `JSON` was selected simply because the python `json` package in the standard library is much easier to use than `lxml`, the library we use for XML/HTML parsing.

2. Dumps / Database

Direct interface with a local database, besides caching using mdb and/or sqlite was not implemented as part of the thesis. However shortly after caching and compile time data pools in redis and postrgres were implemented.

7.2 Caching

As mentioned WikipediaBase abstracts the caching mechanism functionally to a key-value storage object that behaves like a python dictionary plus an extra `sync` method for explicit flushing. However that is not all, another feature that the interface to the database should be able to handle is the encoding of the saved objects. Because virtually all of the stored data is text, the underlying database should be able to reliably retrieve exactly the text that was saved, taking into account the encoding. Because of DBM's limitation that keys of the DBM database should only be ASCII encoded, the base class for interfacing with the database, `EncodedDict`, implements the `_encode_key` and `_decode_key` methods (that default to identity functions) to provide an easy hook for implementations to deal with this possible issue.

1. DBM

   As mentioned before for caching several dbm implementations are provided by the python standard library. None of the implementations shipped with python are part of the python standard library itself however. Some of the DBM implementations that are available via the standard python library are:

   - AnyDBM
   - GNU DBM
   - Berkeley DBM

   It is worth noting that the performance and smooth functioning of these libraries is highly dependent on the underlying platform.

   As mentioned above, the interface classes to DBM transcode keys to ASCII. The precise way that is done is:

```
def _encode_key(self, key):
    if isinstance(key, unicode):
        return key.encode('unicode_escape')

    return str(key)
```

```
def _decode_key(self, key):
    return key.decode('unicode_escape')
```

2. SQLite

   SQLite was also considered as caching backend database. Unfortunately its performance for our particular purpose was disappointing.

   We used a very thin wrapper, `sqlitedict`, to get a key-value interface to SQLite – a relational database. The related WikipediaBase code is very short:

```
from sqlitedict import SqliteDict

  class SqlitePersistentDict(EncodedDict):
      def __init__(self, filename, configuration=
          configuration):
          if not filename.endswith('.sqlite'):
              filename += '.sqlite'

          db = SqliteDict(filename)
          super(SqlitePersistentDict, self).
              __init__(db)

      def sync(self):
self.db.close()
super(SqlitePersistentDict, self).sync()
```

   Below are two benchmark functions that will read/write 100000 times to a key-value database.

```
def benchmark_write(dic, times=100000):
      for i in xrange(times):
          dic['o' + str(i)] = str(i) * 1000

  def benchmark_read(dic, times=100000):
      for i in xrange(times):
dic['o' + str(i)]
```

   And here they are run over memory based `tmpfs` on debian.

```
>>> import timeit
>>> sqlkv = SqlitePersistentDict('/tmp/bench1.
   sqlite')
>>> timeit.timeit(lambda : benchmark_write(sqlkv)
    , number=100)
10.847157955169678
>>> timeit.timeit(lambda : benchmark_read(sqlkv),
     number=100)
18.88098978996277
>>> dbmkv = DbmPersistentDict('/tmp/bench.dbm')
>>> timeit.timeit(lambda : benchmark_write(dbmkv)
    , number=100)
0.18030309677124023
>>> timeit.timeit(lambda : benchmark_read(dbmkv),
     number=100)
0.14914202690124512
```

The DBM database is nearly 10 times faster than sqlite. The difference in performance is due to the different committing policies of the two. It might be possible to calibrate SQLite to be as fast as DBM but not in any trivial way.

3. Other backends

   Other backends were considered, most notably Redis which was actually implemented shortly after the project handoff by Alvaro Morales. The reason we did not initially use it was that it is modeled as a server-client which adds complexity to an aspect of the system that should be as simple as possible. Another reason for our initial skepticism towards third party – ie. not shipped with python – databases was to avoid extra dependencies, especially when they are the cool database du jour.

8 Date parser

Dateparser resides in a separate package called overlay-parse

8.1 Parsing with overlays

The concept of an overlay was inspired by emacs overlays. They are objects that specify the behavior of a subset of a text, by assigning properties to it, making for example text clickable or highlighted. An overlay over part of text t in our context is

- a tuple representing the range within that text
- a set of tags that define semantic sets that the said substring is a member of
- arbitrary information (of type A) that the underlying text describes.

More formally:

$$\begin{aligned} &o_i \in TextRanget \times Set(Tag) \times A \qquad numbers \\ &Text \rightarrow \{o_1, o_2, ..., o_n\} \end{aligned}$$

So for example out of the text

$$The\,weather\,today,\ \overbrace{Tuesday}^{o_1}\ \overbrace{21^{st}}^{o_2}\ of\ \overbrace{November}^{o_3}\ \overbrace{2016}^{o_4},\ was\,sunny.$$

We can extract overlays $\{o_1, ..., o_4\}$, so that

$$\begin{array}{llll} o_1 = (& r(\text{"}Tuesday\text{"}), & \{\text{DayOfWeek}, \text{FullName}\}, & 2) \\ o_2 = (& r(\text{"}21^{st}\text{"}), & \{\text{DayOfMonth}, \text{Numeric}\}, & 21) \\ o_3 = (& r(\text{"}November\text{"}), & \{\text{Month}, \text{FullName}\}, & 11) \\ o_4 = (& r(\text{"2016"}), & \{\text{Year}, \text{4digit}\}, & 2016) \end{array}$$

Notice how for all overlays of the example we have $A = \mathbb{N}$, as we encode day of the week, day of the month, month and year as natural numbers. We encode more precise type information (ie that a day is inherently different than a month) in the tag set.

Once we have a set of overlays we can define overlay sequences as overlays whose ranges are consecutive, that is their tag sets match particular patterns. For example we can search for sequences of overlays that match the pattern

$$p = \text{DayOfMonth}, \text{Separator}(/), (\text{Month} \wedge \text{Number}), \text{Separator}(/), \text{Year}$$

to match patterns like 22/07/1991, where *Separator*(/) matches only the character "/"

8.2 The dates example

The working example and motivation of the package is date parsing. The `dates` submodule exposes two main entry points:

- `just_dates` that looks for dates in a text.
- `just_ranges` that looks for data ranges in a corpus.

Below are presented some examples. Note that `0` means `unspecified`

```
>>> from overlay_parse.dates  import just_dates,
    just_ranges, just_props
>>> just_dates("Timestamp:␣22071991:␣She␣said␣she␣was
    ␣\
␣␣␣␣␣␣␣␣coming␣on␣april␣the␣18th,␣it's␣26␣apr␣2014␣
    and␣hope␣is␣leaving␣me.")
... [(22, 7, 1991), (18, 4, 0), (26, 4, 2014)]
>>> dates = just_dates("200␣AD␣300␣b.c.")
>>> just_dates("200␣AD␣300␣b.c.")
[(0, 0, 200), (0, 0, -300)]
>>> just_ranges(u"I␣will␣be␣there␣from␣2008␣to␣2009")
[((0, 0, 2008), (0, 0, 2009))]
>>> just_ranges("I␣will␣stay␣from␣July␣the␣20th␣until
    ␣today")
[((20, 7, 0), (29, 4, 2016))]
>>> just_dates('{{Birth␣date␣and␣age|1969|7|10|df=y}}
    ')
[(10, 7, 1969)]
>>> just_ranges(u'German:␣[\u02c8v\u0254lf\u0261a\
    u014b␣ama\u02c8de\u02d0\u028as␣\u02c8mo\u02d0tsa\
    u0281t],␣English␣see␣fn.;[1]␣27␣January␣1756\xa0\
    u2013␣5␣December␣1791')
[((27, 1, 1756), (5, 12, 1791))]
```

9 Appendix

9.1 Python unit test example

```
class TestFetcher(unittest.TestCase):
    def setUp(self):
        self.fetcher = fetcher.get_fetcher()
    def test_html(self):
        html = self.fetcher.html_source("Led␣Zeppelin
           ")
        self.assertIn("Jimmy␣Page", html)

    def test_markup_source(self):
        src = self.fetcher.markup_source("Led␣
           Zeppelin")
        self.assertIn("{{Infobox␣musical␣artist", src
           )


    def test_unicode_html(self):
        html = self.fetcher.html_source(u"Rhône")
        self.assertIn("France", html)

    def test_unicode_source(self):
        src = self.fetcher.markup_source("Rhône")
        self.assertIn("Geobox|River", src)

    def test_silent_redirect(self):
        # redirects are only supported when
           force_live is set to True
        src = self.fetcher.markup_source("Obama",
           force_live=True)
        self.assertFalse(re.match(fetcher.
           REDIRECT_REGEX, src))
```

9.2 Python test runs

```
$ python setup.py test -s tests.test_lispify
running test
running egg_info
writing requirements to wikipediabase.egg-info/
   requires.txt
writing wikipediabase.egg-info/PKG-INFO
writing top-level names to wikipediabase.egg-info/
   top_level.txt
writing dependency_links to wikipediabase.egg-info/
   dependency_links.txt
writing entry points to wikipediabase.egg-info/
   entry_points.txt
```

```
reading manifest file 'wikipediabase.egg-info/SOURCES
   .txt'
reading manifest template 'MANIFEST.in'
writing manifest file 'wikipediabase.egg-info/SOURCES
   .txt'
running build_ext
test_bool (tests.test_lispify.TestLispify) ... ok
test_bool_with_typecode (tests.test_lispify.
   TestLispify) ... ok
test_date_multiple_voting (tests.test_lispify.
   TestLispify) ... ok
test_date_simple (tests.test_lispify.TestLispify) ...
    ok
test_date_with_range (tests.test_lispify.TestLispify)
    ... ok
test_dict (tests.test_lispify.TestLispify) ... ok
test_dict_with_escaped_string (tests.test_lispify.
   TestLispify) ... ok
test_dict_with_list (tests.test_lispify.TestLispify)
   ... ok
test_double_nested_list (tests.test_lispify.
   TestLispify) ... ok
test_error (tests.test_lispify.TestLispify) ... ok
test_error_from_exception (tests.test_lispify.
   TestLispify) ... ok
test_keyword (tests.test_lispify.TestLispify) ... ok
test_keyword_with_typecode (tests.test_lispify.
   TestLispify) ... ok
test_list (tests.test_lispify.TestLispify) ... ok
test_list_of_dict (tests.test_lispify.TestLispify)
   ... ok
test_list_of_dict_with_typecode (tests.test_lispify.
   TestLispify) ... ok
test_list_with_typecode (tests.test_lispify.
   TestLispify) ... ok
test_nested_list (tests.test_lispify.TestLispify) ...
    ok
test_none (tests.test_lispify.TestLispify) ... ok
test_none_with_typecode (tests.test_lispify.
   TestLispify) ... ok
test_number (tests.test_lispify.TestLispify) ... ok
test_number_with_typecode (tests.test_lispify.
   TestLispify) ... ok
test_string (tests.test_lispify.TestLispify) ... ok
```

```
test_string_escaped (tests.test_lispify.TestLispify)
   ... ok
test_string_not_keyword (tests.test_lispify.
   TestLispify) ... ok
test_string_with_typecode (tests.test_lispify.
   TestLispify) ... ok
test_unicode_string (tests.test_lispify.TestLispify)
   ... ok

----------------------------------------------------------------------
Ran 27 tests in 0.047s

OK
```

9.3 Quickly finding synonyms with MySQL example

```
mysql> select page_title, rd_title from \
redirect join page on
rd_from = page_id and
(rd_title = "Bill_Clinton" or page_title = "
   Bill_Clinton");
+--
   -----------------------------------+--------------+

| page_title                          | rd_title
   |
+--
   -----------------------------------+--------------+

| BillClinton                         | Bill_Clinton
   |
| William_Jefferson_Clinton           | Bill_Clinton
   |
[.. see below for a formated verison of the data ...]
| William_Jefferson_Clinton           | Bill_Clinton
   |
+--
   -----------------------------------+--------------+

46 rows in set (11.77 sec)
```

page_title	rd_title
BillClinton	Bill_Clinton
William_Jefferson_Clinton	Bill_Clinton
President_Clinton	Bill_Clinton
William_Jefferson_Blythe_IV	Bill_Clinton
Bill_Blythe_IV	Bill_Clinton
Clinton_Gore_Administration	Bill_Clinton
Buddy_(Clinton's_dog)	Bill_Clinton
Bill_clinton	Bill_Clinton
William_Jefferson_Blythe_III	Bill_Clinton
President_Bill_Clinton	Bill_Clinton
Bull_Clinton	Bill_Clinton
Clinton,_Bill	Bill_Clinton
William_clinton	Bill_Clinton
42nd_President_of_the_United_States	Bill_Clinton
Bill_Jefferson_Clinton	Bill_Clinton
William_J._Clinton	Bill_Clinton
Billl_Clinton	Bill_Clinton
Bill_Clinton\	Bill_Clinton
Bill_Clinton's_Post_Presidency	Bill_Clinton
Bill_Clinton's_Post-Presidency	Bill_Clinton
Klin-ton	Bill_Clinton
Bill_J._Clinton	Bill_Clinton
William_Jefferson_"Bill"_Clinton	Bill_Clinton
William_Blythe_III	Bill_Clinton
William_J._Blythe	Bill_Clinton
William_J._Blythe_III	Bill_Clinton
Bil_Clinton	Bill_Clinton
WilliamJeffersonClinton	Bill_Clinton
William_J_Clinton	Bill_Clinton
Bill_Clinton's_sex_scandals	Bill_Clinton
Billy_Clinton	Bill_Clinton
Willam_Jefferson_Blythe_III	Bill_Clinton
William_"Bill"_Clinton	Bill_Clinton
Billll_Clinton	Bill_Clinton
Bill_Klinton	Bill_Clinton
William_Clinton	Bill_Clinton
Willy_Clinton	Bill_Clinton
William_Jefferson_(Bill)_Clinton	Bill_Clinton

Continued on next page

Continued from previous page

page_title	rd_title
Bubba_Clinton	Bill_Clinton
MTV_president	Bill_Clinton
MTV_President	Bill_Clinton
The_MTV_President	Bill_Clinton
Howard_G._Paster	Bill_Clinton
Clintonesque	Bill_Clinton
William_Clinton	Bill_Clinton
William_Jefferson_Clinton	Bill_Clinton

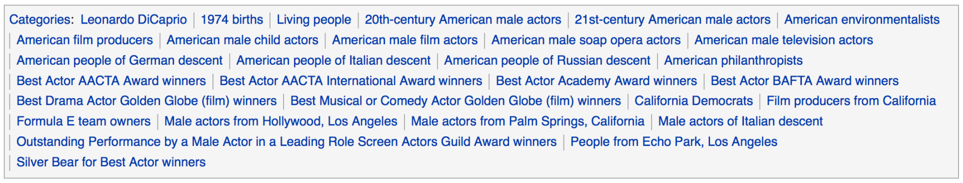
9.4 Article categories example

For example Leonardo DiCaprio's page has the following categories. Highlighted is the category that tells wikipediabase that Leonardo DiCaprio is a person:

- Leonardo DiCaprio
- 1974 births
- **Living people**
- 20th-century American male actors
- 21st-century American male actors
- American environmentalists
- American film producers
- American male child actors
- American male film actors
- American male soap opera actors
- American male television actors
- American people of German descent
- American people of Italian descent
- American people of Russian descent

- American philanthropists
- Best Actor AACTA Award winners
- Best Actor Academy Award winners
- Best Drama Actor Golden Globe (film) winners
- Best Musical or Comedy Actor Golden Globe (film) winners
- California Democrats
- Film producers from California
- Formula E team owners
- Male actors from Hollywood, California
- Male actors from Palm Springs, California
- Male actors of Italian descent
- People from Echo Park, Los Angeles
- Silver Bear for Best Actor winners

This looks like this on the wikipedia page.



Categories: Leonardo DiCaprio | 1974 births | Living people | 20th-century American male actors | 21st-century American male actors | American environmentalists | American film producers | American male child actors | American male film actors | American male soap opera actors | American male television actors | American people of German descent | American people of Italian descent | American people of Russian descent | American philanthropists | Best Actor AACTA Award winners | Best Actor AACTA International Award winners | Best Actor Academy Award winners | Best Actor BAFTA Award winners | Best Drama Actor Golden Globe (film) winners | Best Musical or Comedy Actor Golden Globe (film) winners | California Democrats | Film producers from California | Formula E team owners | Male actors from Hollywood, Los Angeles | Male actors from Palm Springs, California | Male actors of Italian descent | Outstanding Performance by a Male Actor in a Leading Role Screen Actors Guild Award winners | People from Echo Park, Los Angeles | Silver Bear for Best Actor winners

Figure 4: The rendered list of categores for Leonardo DiCaprio