IMDbPY Documentation

Release 6.6

Davide Alberani, H. Turgut Uyar

Aug 05, 2018

Contents:

IMDbPY is a Python package for retrieving and managing the data of the IMDb movie database about movies, people and companies.

Homepage https://imdbpy.sourceforge.io/

PyPI https://pypi.org/project/IMDbPY/

Repository https://github.com/alberanid/imdbpy

Documentation https://imdbpy.readthedocs.io/

Support https://imdbpy.sourceforge.io/support.html

Revamp notice

Starting on November 2017, many things were improved and simplified:

- moved the package to Python 3 (compatible with Python 2.7)
- removed dependencies: SQLObject, C compiler, BeautifulSoup
- removed the "mobile" and "httpThin" parsers
- introduced a test suite (please help with it!)

CHAPTER 1

Main features

- written in Python 3 (compatible with Python 2.7)
- platform-independent
- can retrieve data from both the IMDb's web server, or a local copy of the database
- simple and complete API
- released under the terms of the GPL 2 license

IMDbPY powers many other software and has been used in various research papers. Curious about that?

CHAPTER 2

Installation

Whenever possible, please use the latest version from the repository:

```
pip install git+https://github.com/alberanid/imdbpy
```

But if you want, you can also install the latest release from PyPI:

```
pip install imdbpy
```

CHAPTER 3

Example

Here's an example that demonstrates how to use IMDbPY:

```
from imdb import IMDb

# create an instance of the IMDb class
ia = IMDb()

# get a movie
movie = ia.get_movie('0133093')

# print the names of the directors of the movie
print('Directors:')
for director in movie['directors']:
    print(director['name'])

# print the genres of the movie
print('Genres:')
for genre in movie['genres']:
    print(genre)

# search for a person name
people = ia.search_person('Mel Gibson')
for person in people:
   print(person.personID, person['name'])
```

Disclaimer

IMDbPY and its authors are not affiliated with Internet Movie Database Inc.

IMDb is a trademark of Internet Movie Database Inc., and all content and data included on the IMDb's site is the property of IMDb or its content suppliers and protected by United States and international copyright laws.

Please read the IMDb's conditions of use on their website:

- https://www.imdb.com/conditions

- https://www.imdb.com/licensing
- any other notice on the https://www.imdb.com/ site

3.1 Usage

Here you can find information about how you can use IMDbPY in your own programs.

> **Warning:** This document is far from complete: the code is the final documentation! ;-)

3.1.1 Quick start

The first thing to do is to import `imdb` and call the `imdb.IMDb` function to get an access object through which IMDb data can be retrieved:

```
>>> import imdb
>>> ia = imdb.IMDb()
```

By default this will fetch the data from the IMDb web server but there are other options. See the *access systems* document for more information.

Searching

You can use the `search_movie` method of the access object to search for movies with a given (or similar) title. For example, to search for movies with titles like "matrix":

```
>>> movies = ia.search_movie('matrix')
>>> movies[0]
<Movie id:0133093[http] title:_The Matrix (1999)_>
```

Similarly, you can search for people and companies using the `search_person` and the `search_company` methods:

```
>>> people = ia.search_person('angelina')
>>> people[0]
<Person id:0001401[http] name:_Jolie, Angelina_>
>>> companies = ia.search_company('rko')
>>> companies[0]
<Company id:0226417[http] name:_RKO_>
```

As the examples indicate, the results are lists of `Movie`, `Person`, or `Company` objects. These behave like dictionaries, i.e. they can be queried by giving the key of the data you want to obtain:

```
>>> movies[0]['title']
'The Matrix'
>>> people[0]['name']
'Angelina Jolie'
>>> companies[0]['name']
'RKO'
```

Movie, person, and company objects have id attributes which -when fetched through the IMDb web server- store the IMDb id of the object:

```
>>> movies[0].movieID
'0133093'
>>> people[0].personID
'0001401'
>>> companies[0].companyID
'0226417'
```

Retrieving

If you know the IMDb id of a movie, you can use the `get_movie` method to retrieve its data. For example, the movie "The Untouchables" by Brian De Palma has the id "0094226":

```
>>> movie = ia.get_movie('0094226')
>>> movie
<Movie id:0094226[http] title:_The Untouchables (1987)_>
```

Similarly, the `get_person` and the `get_company` methods can be used for retrieving `Person` and `Company` data:

```
>>> person = ia.get_person('0000206')
>>> person['name']
'Keanu Reeves'
>>> person['birth date']
'1964-9-2'
>>> company = ia.get_company('0017902')
>>> company['name']
'Pixar Animation Studios'
```

Keywords

You can search for keywords similar to the one provided:

```
>>> keywords = ia.search_keyword('dystopia')
>>> keywords
['dystopia', 'dystopian-future', ..., 'dystopic-future']
```

And movies that match a given keyword:

```
>>> movies = ia.get_keyword('dystopia')
>>> len(movies)
50
>>> movies[0]
<Movie id:1677720[http] title:_Ready Player One (2018)_>
```

Top / bottom movies

It's possible to retrieve the list of top 250 and bottom 100 movies:[1]

[1] Beware that in an SQL-based access system, the bottom 100 list is limited to the first 10 results.

```
>>> top = ia.get_top250_movies()
>>> top[0]
<Movie id:0111161[http] title:_The Shawshank Redemption (1994)_>
>>> bottom = ia.get_bottom100_movies()
>>> bottom[0]
<Movie id:4458206[http] title:_Code Name: K.O.Z. (2015)_>
```

Exceptions

Any error related to IMDbPY can be caught by checking for the `imdb.IMDbError` exception:

```
from imdb import IMDb, IMDbError

try:
    ia = IMDb()
    people = ia.search_person('Mel Gibson')
except IMDbError as e:
    print(e)
```

3.1.2 Data interface

The IMDbPY objects that represent movies, people and companies provide a dictionary-like interface where the key identifies the information you want to get out of the object.

At this point, I have really bad news: what the keys are is a little unclear!

In general, the key is the label of the section as used by the IMDb web server to present the data. If the information is grouped into subsections, such as cast members, certifications, distributor companies, etc., the subsection label in the HTML page is used as the key.

The key is almost always lowercase; underscores and dashes are replaced with spaces. Some keys aren't taken from the HTML page, but are defined within the respective class.

Information sets

IMDbPY can retrieve almost every piece of information of a movie, person or company. This can be a problem, because (at least for the "http" data access system) it means that a lot of web pages must be fetched and parsed. This can be both time- and bandwidth-consuming, especially if you're interested in only a small part of the information.

The `get_movie`, `get_person` and `get_company` methods take an optional `info` parameter, which can be used to specify the kinds of data to fetch. Each group of data that gets fetched together is called an "information set".

Different types of objects have their own available information sets. For example, the movie objects have a set called "vote details" for the number of votes and their demographic breakdowns, whereas person objects have a set called "other works" for miscellaneous works of the person. Available information sets for each object type can be queried using the access object:

```
>>> from imdb import IMDb
>>> ia = IMDb()
>>> ia.get_movie_infoset()
['airing', 'akas', ..., 'video clips', 'vote details']
>>> ia.get_person_infoset()
['awards', 'biography', ..., 'other works', 'publicity']
```

(continues on next page)

(continued from previous page)

```
>>> ia.get_company_infoset()
['main']
```

For each object type, only the important information will be retrieved by default:

- for a movie: “main”, “plot”
- for a person: “main”, “filmography”, “biography”
- for a company: “main”

These defaults can be retrieved from the `default_info` attributes of the classes:

```
>>> from imdb.Person import Person
>>> Person.default_info
('main', 'filmography', 'biography')
```

Each instance also has a `current_info` attribute for tracking the information sets that have already been retrieved:

```
>>> movie = ia.get_movie('0133093')
>>> movie.current_info
['main', 'plot', 'synopsis']
```

The list of retrieved information sets and the keys they provide can be taken from the `infoset2keys` attribute:

```
>>> movie = ia.get_movie('0133093')
>>> movie.infoset2keys
{'main': ['cast', 'genres', ..., 'top 250 rank'], 'plot': ['plot', 'synopsis']}
>>> movie = ia.get_movie('0094226', info=['taglines', 'plot'])
>>> movie.infoset2keys
{'taglines': ['taglines'], 'plot': ['plot', 'synopsis']}
>>> movie.get('title')
>>> movie.get('taglines')[0]
'The Chicago Dream is that big'
```

Search operations retrieve a fixed set of data and don’t have the concept of information sets. Therefore objects listed in searches will have even less information than the defaults. For example, if you do a movie search operation, the movie objects in the result won’t have many of the keys that would be available on a movie get operation:

```
>>> movies = ia.search_movie('matrix')
>>> movie = movies[0]
>>> movie
<Movie id:0133093[http] title:_The Matrix (1999)_>
>>> movie.current_info
[]
>>> 'genres' in movie
False
```

Once an object is retrieved (through a get or a search), its data can be updated using the `update` method with the desired information sets. Continuing from the example above:

```
>>> 'median' in movie
False
>>> ia.update(movie, info=['taglines', 'vote details'])
>>> movie.current_info
['taglines', 'vote details']
>>> movie['median']
9
```

(continues on next page)

(continued from previous page)

```
>>> ia.update(movie, info=['plot'])
>>> movie.current_info
['taglines', 'vote details', 'plot', 'synopsis']
```

Beware that the information sets vary between access systems: locally not every piece of data is accessible, whereas -for example for SQL- accessing one set of data means automatically accessing a number of other information (without major performance drawbacks).

Composite data

In some data, the (not-so) universal `::` separator is used to delimit parts of the data inside a string, like the plot of a movie and its author:

```
>>> movie = ia.get_movie('0094226')
>>> plot = movie['plot'][0]
>>> plot
"1920's prohibition ... way to get him.::Jeremy Perkins <jwp@aber.ac.uk>"
```

As a rule, there's at most one such separator inside a string. Splitting the string will result in two logical pieces as in `TEXT::NOTE`. The `imdb.helpers.makeTextNotes()` function can be used to create a custom function to pretty-print this kind of information.

References

Sometimes the collected data contains strings with references to other movies or persons, e.g. in the plot of a movie or the biography of a person. These references are stored in the Movie, Person, and Character instances; in the strings you will find values like _A Movie (2003)_ (qv) or 'A Person' (qv) or '#A Character# (qv)'. When these strings are accessed (like movie['plot'] or person['biography']), they will be modified using a provided function, which must take the string and two dictionaries containing titles and names references as parameters.

By default the (qv) strings are converted in the "normal" format ("A Movie (2003)", "A Person" and "A Character").

You can find some examples of these functions in the imdb.utils module.

The function used to modify the strings can be set with the `defaultModFunct` parameter of the IMDb class or with the `modFunct` parameter of the `get_movie`, `get_person`, and `get_character` methods:

```
import imdb
i = imdb.IMDb(defaultModFunct=imdb.utils.modHtmlLinks)
```

or:

```
import imdb
i = imdb.IMDb()
i.get_person('0000154', modFunct=imdb.utils.modHtmlLinks)
```

3.1.3 Roles

When parsing data of a movie, you'll encounter references to the people who worked on it, like its cast, director and crew members.

For people in the cast (actors and actresses), the `currentRole` attribute is set to the name of the character they played:

```
>>> movie = ia.get_movie('0075860')
>>> movie
<Movie id:0075860[http] title:_Close Encounters of the Third Kind (1977)_>
>>> actor = movie['cast'][6]
>>> actor
<Person id:0447230[http] name:_Kemmerling, Warren J._>
>>> actor['name']
'Warren J. Kemmerling'
>>> actor.currentRole
'Wild Bill'
```

Miscellaneous data, such as an AKA name for the actor or an "uncredited" notice, is stored in the `notes` attribute:

```
>>> actor.notes
'(as Warren Kemmerling)'
```

For crew members other than the cast, the `notes` attribute contains the description of the person's job:

```
>>> crew_member = movie['art department'][0]
>>> crew_member
<Person id:0330589[http] name:_Gordon, Sam_>
>>> crew_member.notes
'property master'
```

The `in` operator can be used to check whether a person worked in a given movie or not:

```
>>> movie
<Movie id:0075860[http] title:_Close Encounters of the Third Kind (1977)_>
>>> actor
<Person id:0447230[http] name:_Kemmerling, Warren J._>
>>> actor in movie
True
>>> crew_member
<Person id:0330589[http] name:_Gordon, Sam_>
>>> crew_member in movie
True
>>> person
<Person id:0000210[http] name:_Roberts, Julia (I)_>
>>> person in movie
False
```

Obviously these Person objects contain only information directly available upon parsing the movie pages, e.g.: the name, an imdbID, the role. So if now you write:

```
print(writer['actor'])
```

to get a list of movies acted by Mel Gibson, you'll get a KeyError exception, because the Person object doesn't contain this kind of information.

The same is true when parsing person data: you'll find a list of movie the person worked on and, for every movie, the currentRole instance variable is set to a string describing the role of the considered person:

```
# Julia Roberts
julia = i.get_person('0000210')
# Output a list of movies she acted in and the played role
# separated by '::'
print([movie['title'] + '::' + movie.currentRole
       for movie in julia['actress']])
```

Here the various Movie objects only contain minimal information, like the title and the year; the latest movie with Julia Roberts:

```
last = julia['actress'][0]
# Retrieve full information
i.update(last)
# name of the first director
print(last['director'][0]['name'])
```

Note: Since the end of 2017, IMDb has removed the Character kind of information. This document is still valid, but only for the obsolete "sql" data access system.

Since version 3.3, IMDbPY supports the character pages of the IMDb database; this required some substantial changes to how actors' and acresses' roles were handled. Starting with release 3.4, "sql" data access system is supported, too - but it works a bit differently from "http". See "SQL" below.

The currentRole instance attribute can be found in every instance of Person, Movie and Character classes, even if actually the Character never uses it.

The currentRole of a Person object is set to a Character instance, inside a list of person who acted in a given movie. The currentRole of a Movie object is set to a Character instance, inside a list of movies played be given person. The currentRole of a Movie object is set to a Person instance, inside a list of movies in which a given character was portrayed.

Schema:

```
movie['cast'][0].currentRole -> a Character object.
               |
               +-> a Person object.

person['actor'][0].currentRole -> a Character object.
                |
                +-> a Movie object.

character['filmography'][0].currentRole -> a Person object.
                          |
                          +-> a Movie object.
```

The roleID attribute can be used to access/set the characterID or personID instance attribute of the current currentRole. When building Movie or Person objects, you can pass the currentRole parameter and the roleID parameter (to set the ID). The currentRole parameter can be an object (Character or Person), a string (in which case a Character or Person object is automatically instantiated) or a list of objects or strings (to handle multiple characters played by the same actor/actress in a movie, or character played by more then a single actor/actress in the same movie).

Anyway, currentRole objects (Character or Person instances) can be pretty-printed easily: calling unicode(CharacterOrPersonObject) will return a good-old-string.

SQL

Fetching data from the web, only characters with an active page on the web site will have their characterID; we don't have these information when accessing through "sql", so *every* character will have an associated characterID. This way, every character with the same name will share the same characterID, even if - in fact - they may not be portraying the same character.

Goodies

To help getting the required information from Movie, Person and Character objects, in the "helpers" module there's a new factory function, makeObject2Txt, which can be used to create your pretty-printing function. It takes some optional parameters: movieTxt, personTxt, characterTxt and companyTxt; in these strings %(value)s items are replaced with object['value'] or with obj.value (if the first is not present).

E.g.:

```
import imdb
myPrint = imdb.helpers.makeObject2Txt(personTxt=u'%(name)s ... %(currentRole)s')
i = imdb.IMDb()
m = i.get_movie('0057012')
ps = m['cast'][0]
print(myPrint(ps))
# The output will be something like:
# Peter Sellers ... Group Captain Lionel Mandrake / President Merkin Muffley / Dr.␣
↪Strangelove
```

Portions of the formatting string can be stripped conditionally: if the specified condition is false, they will be cancelled.

E.g.:

```
myPrint = imdb.helpers.makeObject2Txt(personTxt='<if personID><a href=/person/
↪%(personID)s></if personID>%(long imdb name)s<if personID></a></if personID><if␣
↪currentRole> ... %(currentRole)s<if notes> %(notes)s</if notes></if currentRole>'
```

Another useful argument is 'applyToValues': if set to a function, it will be applied to every value before the substitution; it can be useful to format strings for HTML output.

3.1.4 Series

As on the IMDb site, each TV series and also each of a TV series' episodes is treated as a regular title, just like a movie. The `kind` key can be used to distinguish series and episodes from movies:

```
>>> series = ia.get_movie('0389564')
>>> series
<Movie id:0389564[http] title:_"The 4400" (2004)_>
>>> series['kind']
'tv series'
>>> episode = ia.get_movie('0502803')
>>> episode
<Movie id:0502803[http] title:_"The 4400" Pilot (2004)_>
>>> episode['kind']
'episode'
```

The episodes of a series can be fetched using the "episodes" infoset. This infoset adds an `episodes` key which is a dictionary from season numbers to episodes. And each season is a dictionary from episode numbers within the season to the episodes. Note that the season and episode numbers don't start from 0; they are the numbers given by the IMDb:

```
>>> ia.update(series, 'episodes')
>>> sorted(series['episodes'].keys())
[1, 2, 3, 4]
>>> season4 = series['episodes'][4]
>>> len(season4)
13
```

(continues on next page)

(continued from previous page)

```
>>> episode = series['episodes'][4][2]
>>> episode
<Movie id:1038701[http] title:_"The 4400" Fear Itself (2007)_>
>>> episode['season']
4
>>> episode['episode']
2
```

The title of the episode doesn't contain the title of the series:

```
>>> episode['title']
'Fear Itself'
>>> episode['series title']
'The 4400'
```

The episode also contains a key that refers to the series, but beware that, to avoid circular references, it's not the same object as the series object we started with:

```
>>> episode['episode of']
<Movie id:0389564[http] title:_"The 4400" (None)_>
>>> series
<Movie id:0389564[http] title:_"The 4400" (2004)_>
```

Titles

The `analyze_title()` and `build_title()` functions now support TV episodes. You can pass a string to the `analyze_title` function in the format used by the web server (`"The Series" The Episode (2005)`) or in the format of the plain text data files (`"The Series" (2004) {The Episode (#ser.epi)}`).

For example, if you call the function:

```
analyze_title('"The Series" The Episode (2005)')
```

the result will be:

```
{
    'kind': 'episode',        # kind is set to 'episode'
    'year': '2005',           # release year of this episode
    'title': 'The Episode',   # episode title
    'episode of': {           # 'episode of' will contain
        'kind': 'tv series',  # information about the series
        'title': 'The Series'
    }
}
```

The `episode of` key can be a dictionary or a `Movie` instance with the same information.

The `build_title()` function takes an optional argument: `ptdf`, which when set to false (the default) returns the title of the episode in the format used by the IMDb's web server ("The Series" An Episode (2006)); otherwise, it uses the format used by the plain text data files (something like "The Series" (2004) {An Episode (#2.5)})

Full credits

When retrieving credits for a TV series or mini-series, you may notice that many long lists (like "cast" and "writers") are incomplete. You can fetch the complete list of cast and crew with the "full credits" data set:

```
>>> series = ia.get_movie('0285331')
>>> series
<Movie id:0285331[http] title:_"24" (2001)_>
>>> len(series['cast'])
50
>>> ia.update(series, 'full credits')
>>> len(series['cast'])
2514
```

If you prefer, you can retrieve the complete cast of every episode, keeping the lists separated for each episode. Instead of retrieving with:

```
ia.update(series, 'episodes')
```

use:

```
ia.update(series, 'episodes cast')
```

or the equivalent:

```
i.update(m, 'guests')
```

Now you end up having the same information as if you have updated the 'episodes' info set, but every Movie object inside the dictionary of dictionary has the complete cast, e.g.:

```
cast = m['episodes'][1][2]['cast']  # cast list for the second episode
                                    # of the first season.
```

Beware that both 'episodes cast' and 'guests' will update the keyword 'episodes' (and not 'episodes cast' or 'guests').

Ratings

You can retrieve rating information about every episode in a TV series or mini series using the 'episodes rating' data set.

People

You can retrieve information about single episodes acted/directed/... by a person.

```
from imdb import IMDb
i = IMDb()
p = i.get_person('0005041')  # Laura Innes
p['actress'][0]   # <Movie id:0568152[http] title:_"ER" (????)_>

# At this point you have an entry (in keys like 'actor', 'actress',
# 'director', ...) for every series the person starred/worked in, but
# you knows nothing about singles episodes.
i.update(p, 'episodes')  # updates information about single episodes.

p['episodes']    # a dictionary with the format:
                 #    {<TV Series Movie Object>: [
                 #                                 <Episode Movie Object>,
                 #                                 <Episode Movie Object>,
                 #                                 ...
                 #                                ],
```

(continues on next page)

(continued from previous page)

```
                    #     ...
                    #    }

er = p['actress'][0]  # ER tv series
p['episodes'][er]     # list of Movie objects; one for every ER episode
                      # she starred/worked in

p['episodes'][er][0]  # <Movie id:0568154[http] title:_"ER" Welcome Back Carter!␣
↪(1995)_>
p['episodes'][er]['kind']    # 'episode'
p['episodes'][er][0].currentRole   # 'Dr. Kerry Weaver'
```

Goodies

In the `imdb.helpers` module there are some functions useful to manage lists of episodes:

- `sortedSeasons(m)` returns a sorted list of seasons of the given series, e.g.:

```
>>> from imdb import IMDb
>>> i = IMDb()
>>> m = i.get_movie('0411008')
>>> i.update(m, 'episodes')
>>> sortedSeasons(m)
[1, 2]
```

- `sortedEpisodes(m, season=None)` returns a sorted list of episodes of the the given series for only the specified season(s) (if None, every season), e.g.:

```
>>> from imdb import IMDb
>>> i = IMDb()
>>> m = i.get_movie('0411008')
>>> i.update(m, 'episodes')
>>> sortedEpisodes(m, season=1)
[<Movie id:0636289[http] title:_"Lost" Pilot: Part 1 (2004)_>, <Movie␣
↪id:0636290[http] title:_"Lost" Pilot: Part 2 (2004)_>, ...]
```

3.1.5 Adult movies

IMDbPY for (too) sensitive people.

Since version 2.0 (shame on me! I've noticed this only after more than a year of development!!!) adult movies are included by default in search results.

If for some unintelligible reason you don't want classics like "Debbie Does Dallas" to show up in your searches, you can disable this feature by initializing the `IMDb` class with the `adultSearch` argument set to `False`:

```
>>> ia = IMDb(accessSystem='http', adultSearch=False)
```

This behavior can also be modified at runtime by calling the `do_adult_search` method:

```
>>> ia = IMDb(accessSystem='http')    # by default in horny mode
>>> movies = ia.search_movie('debby does dallas', results=5)
>>> movieIDs = [m.movieID for m in movies]
>>> '0077415' in movieIDs             # Debbie Does Dallas (1978)
```

(continues on next page)

(continued from previous page)

```
True
>>> ia.do_adult_search(False)        # switch to puritan behavior
>>> movies = ia.search_movie('debby does dallas', results=5)
>>> movieIDs = [m.movieID for m in movies]
>>> '0077415' in movieIDs            # Debbie Does Dallas (1978)
False
```

The `do_adult_search` method of the HTTP data access system can take two more arguments: `cookie_id` and `cookie_uu`, so that you can select *your own* IMDb account; if cookie id is set to None, no cookies are sent. These parameters can also be set in the `imdbpy.cfg` configuration file. To find the strings you need to use, see your "cookie" or "cookie.txt" files. Obviously, you'll need to activate the "adult movies" option for your account; see http://imdb.com/find/preferences?_adult=1

Since version 2.2 all data access systems (sql) support the same behavior (i.e.: you can set the `adultSearch` argument and use the `do_adult_search` method).

Note that for the sql data access system, only results from the `search_movie()` and `search_episode()` methods are filtered: there's no easy (and fast) way to tell whether an actor/actress is a porn star or not.

3.1.6 Information in XML format

Since version 4.0, IMDbPY can output information of Movie, Person, Character, and Company instances in XML format. It's possible to get a single information (a key) in XML format, using the `getAsXML(key)` method (it will return None if the key is not found). E.g.:

```
from imdb import IMDb
ia = IMDb('http')
movie = ia.get_movie(theMovieID)
print(movie.getAsXML('keywords'))
```

It's also possible to get a representation of a whole object, using the `asXML()` method:

```
print(movie.asXML())
```

The `_with_add_keys` argument of the `asXML()` method can be set to False (default: True) to exclude the dynamically generated keys (like 'smart canonical title' and so on).

XML format

Keywords are converted to tags, items in lists are enclosed in a 'item' tag, e.g.:

```
<keywords>
  <item>a keyword</item>
  <item>another keyword</item>
</keywords>
```

Except when keys are known to be not fixed (e.g.: a list of keywords), in which case this schema is used:

```
<item key="EscapedKeyword">
   ...
</item>
```

In general, the 'key' attribute is present whenever the used tag doesn't match the key name.

Movie, Person, Character and Company instances are converted as follows (portions in square brackets are optional):

```
<movie id="movieID" access-system="accessSystem">
  <title>A Long IMDb Movie Title (YEAR)</title>
  [<current-role>
     <person id="personID" access-system="accessSystem">
       <name>Name Surname</name>
       [<notes>A Note About The Person</notes>]
     </person>
   </current-role>]
   [<notes>A Note About The Movie</notes>]
</movie>
```

Every 'id' can be empty.

The returned XML string is mostly not pretty-printed.

References

Some text keys can contain references to other movies, persons and characters. The user can provide the `defaultModFunct` function (see the "MOVIE TITLES AND PERSON/CHARACTER NAMES REFERENCES" section of the README.package file), to replace these references with their own strings (e.g.: a link to a web page); it's up to the user, to be sure that the output of the defaultModFunct function is valid XML.

DTD

Since version 4.1 a DTD is available; it can be found in this directory or on the web, at: http://imdbpy.sf.net/dtd/imdbpy41.dtd

The version number changes with the IMDbPY version.

Localization

Since version 4.1 it's possible to translate the XML tags; see README.locale.

Deserializing

Since version 4.6, you can dump the generated XML in a string or in a file, using it -later- to rebuild the original object. In the `imdb.helpers` module there's the `parseXML()` function which takes a string as input and returns -if possible- an instance of the Movie, Person, Character or Company class.

3.1.7 Localization

Since version 4.1 the labels that describe the information are translatable.

Limitation

Internal messages or exceptions are not translatable, the internationalization is limited to the "tags" returned by the `getAsXML` and `asXML` methods of the Movie, Person, Character, or Company classes.

Beware that in many cases these "tags" are not the same as the "keys" used to access information in the same class. For example, you can translate the tag "long-imdb-name" -the tag returned by the call `person.getAsXML('long`

`imdb name')`, but not the key "long imdb name" itself. To translate keys, you can use the `helpers.translateKey` function.

If you want to add i18n to your IMDbPY-based application, all you need to do is to switch to the `imdbpy` text domain:

```
>>> import imdb.locale
>>> import gettext
>>> gettext.textdomain('imdbpy')
'imdbpy'
>>> from gettext import gettext as _
>>> _('art-department')
'Art department'
>>> import os
>>> os.environ['LANG'] = 'it_IT'
>>> _('art-department')
'Dipartimento artistico'
```

If you want to translate IMDbPY into another language, see the *How to translate* document for instructions.

Articles in titles

To convert a title to its canonical format as in "Title, The", IMDbPY makes some assumptions about what is an article and what isn't, and this can lead to some wrong canonical titles. For example, it can canonicalize the title "Die Hard" as "Hard, Die" because it guesses "Die" as an article (and it is, in Germany...).

To solve this problem, there are other keys: "smart canonical title", "smart long imdb canonical title", "smart canonical series title", "smart canonical episode title" which can be used to do a better job converting a title into its canonical format.

This works, but it needs to know about articles in various languages: if you want to help, see the `linguistics.LANG_ARTICLES` and `linguistics.LANG_COUNTRIES` dictionaries.

To guess the language of a movie title, call its 'guessLanguage' method (it will return None, if unable to guess). If you want to force a given language instead of the guessed one, you can call its 'smartCanonicalTitle' method, setting the 'lang' argument appropriately.

Alternative titles

Sometimes it's useful to manage a title's alternatives (AKAs) knowing their languages. In the 'helpers' module there are some (hopefully) useful functions:

- `akasLanguages(movie)` - Given a movie, return a list of tuples in (lang, AKA) format (lang can be None, if unable to detect).
- `sortAKAsBySimilarity(movie, title)` - Sort the AKAs on a movie considering how much they are similar to a given title (see the code for more options).
- `getAKAsInLanguage(movie, lang)` - Return a list of AKAs of the movie in the given language (see the code for more options).

3.1.8 Access systems

IMDbPY supports different ways of accessing the IMDb data:

- Fetching data directly from the web server.

- Getting the data from a SQL database that can be created from the downloadable data sets provided by the IMDb.

access system	aliases	data source
(default) 'http'	'https' 'web' 'html'	imdb.com web server
's3'	's3dataset'	downloadable dataset *after Dec 2017*
'sql'	'db' 'database'	downloadable dataset *until Dec 2017*

Note: Since release 3.4, the `imdbpy.cfg` configuration file is available, so that you can set a system-wide (or per-user) default. The file is commented with indication of the location where it can be put, and how to modify it.

If no `imdbpy.cfg` file is found (or is not readable or it can't be parsed), 'http' will be used the default.

See the *S3 datasets* and *Old data files* documents for more information about SQL based access systems.

3.1.9 S3 datasets

IMDb distributes some of its data as downloadable datasets. IMDbPY can import this data into a database and make it accessible through its API.[1]

For this, you will first need to install SQLAlchemy and the libraries that are needed for the database server you want to use. Check out the SQLAlchemy dialects documentation for more detail.

Then, follow these steps:

1. Download the files from the following address and put all of them in the same directory: https://datasets.imdbws.com/
2. Create a database. Use a collation like `utf8_unicode_ci`.
3. Import the data using the `s32imdbpy.py` script:

```
s32imdbpy.py /path/to/the/tsv.gz/files/ URI
```

URI is the identifier used to access the SQL database. For example:

```
s32imdbpy.py ~/Download/imdb-s3-dataset-2018-02-07/ \
    postgres://user:password@localhost/imdb
```

Once the import is finished -which should take about an hour or less on a modern system- you will have a SQL database with all the information and you can use the normal IMDbPY API:

```
from imdb import IMDb

ia = IMDb('s3', 'postgres://user:password@localhost/imdb')

results = ia.search_movie('the matrix')
for result in results:
```

(continues on next page)

[1] Until the end of 2017, IMDb used to distribute a more comprehensive subset of its data in a different format. IMDbPY can also import that data but note that the data is not being updated anymore. For more information, see *Old data files*.

(continued from previous page)

```
    print(result.movieID, result)

matrix = results[0]
ia.update(matrix)
print(matrix.keys())
```

Note: Running the script again will drop the current tables and import the data again.

3.1.10 Old data files

Warning: Since the end of 2017, IMDb is no longer updating the data files which are described in this document. For working with the updated -but less comprehensive- downloadable data, check the *S3 datasets* document.

Until the end of 2017, IMDb used to distribute some of its data as downloadable text files. IMDbPY can import this data into a database and make it accessible through its API.

For this, you will first need to install SQLAlchemy and the libraries that are needed for the database server you want to use. Check out the SQLAlchemy dialects documentation for more detail.

Then, follow these steps:

1. Download the files from the following address and put all of them in the same directory: ftp://ftp.funet.fi/pub/mirrors/ftp.imdb.com/pub/frozendata/

   You can just download the files you need instead of downloading all files. The files that are not downloaded will be skipped during import. This feature is still quite untested, so please report any bugs.

   Warning: Beware that the `diffs` subdirectory contains **a lot** of files you **don't** need, so don't start mirroring everything!

2. Create a database. Use a collation like `utf8_unicode_ci`.
3. Import the data using the `imdbpy2sql.py` script:

   ```
   imdbpy2sql.py -d /path/to/the/data_files_dir/ -u URI
   ```

   URI is the identifier used to access the SQL database. For example:

   ```
   imdbpy2sql.py -d ~/Download/imdb-frozendata/ \
       -u postgres://user:password@localhost/imdb
   ```

Once the import is finished, you will have a SQL database with all the information and you can use the normal IMDbPY API:

```
from imdb import IMDb

ia = IMDb('sql', uri='postgres://user:password@localhost/imdb')

results = ia.search_movie('the matrix')
for result in results:
```

(continues on next page)

(continued from previous page)

```
    print(result.movieID, result)

matrix = results[0]
ia.update(matrix)
print(matrix.keys())
```

Note: It should be noted that the `imdbpy2sql.py` script will not create any foreign keys, but only indexes. If you need foreign keys, try using the version in the "imdbpy-legacy" branch.

If you need instructions on how to manually build the foreign keys, see this comment by Andrew D Bate.

Performance

The import performance hugely depends on the underlying module used to access the database. The `imdbpy2sql.py` script has a number of command line arguments for choosing presets that can improve performance in specific database servers.

The fastest database appears to be MySQL, with about 200 minutes to complete on my test system (read below). A lot of memory (RAM or swap space) is required, in the range of at least 250/500 megabytes (plus more for the database server). In the end, the database requires between 2.5GB and 5GB of disk space.

As said, the performance varies greatly using one database server or another. MySQL, for instance, has an `executemany()` method of the cursor object that accepts multiple data insertion with a single SQL statement; other databases require a call to the `execute()` method for every single row of data, and they will be much slower -2 to 7 times slower than MySQL.

There are generic suggestions that can lead to better performance, such as turning off your filesystem journaling (so it can be a good idea to remount an ext3 filesystem as ext2 for example). Another option is using a ramdisk/tmpfs, if you have enough RAM. Obviously these have effect only at insert-time; during day-to-day use, you can turn journaling on again. You can also consider using CSV output as explained below, if your database server can import CSV files.

I've done some tests, using an AMD Athlon 1800+, 1GB of RAM, over a complete plain text data files set (as of 11 Apr 2008, with more than 1.200.000 titles and over 2.200.000 names):

database	time in minutes: total (insert data/create indexes)
MySQL 5.0 MyISAM	205 (160/45)
MySQL 5.0 InnoDB	_untested_, see NOTES below
PostgreSQL 8.1	560 (530/30)
SQLite 3.3	??? (150/???) -very slow building indexes Timed with the "–sqlite-transactions" command line option; otherwise it's _really_ slow: even 35 hours or more
SQLite 3.7	65/13 - with –sqlite-transactions and using an SSD disk
SQL Server	about 3 or 4 hours

If you have different experiences, please tell me!

As expected, the most important things that you can do to improve performance are:

1. Use an in-memory filesystem or an SSD disk.

2. Use the `-c /path/to/empty/dir` argument to use CSV files.
3. Follow the specific notes about your database server.

Notes

[save the output]

The imdbpy2sql.py will print a lot of debug information on standard output; you can save it in a file, appending (without quotes) "2>&1 | tee output.txt"

[Microsoft Windows paths]

It's much safer, in a Microsoft Windows environment, to use full paths for the values of the '-c' and '-d' arguments, complete with drive letter. The best thing is to use _UNIX_ path separator, and to add a leading separator, e.g.:

```
-d C:/path/to/imdb_files/ -c C:/path/to/csv_tmp_files/
```

[MySQL]

In general, if you get an annoyingly high number of "TOO MANY DATA ... SPLITTING" lines, consider increasing max_allowed_packet (in the configuration of your MySQL server) to at least 8M or 16M. Otherwise, inserting the data will be very slow, and some data may be lost.

[MySQL InnoDB and MyISAM]

InnoDB is abysmal slow for our purposes: my suggestion is to always use MyISAM tables and -if you really want to use InnoDB- convert the tables later. The imdbpy2sql.py script provides a simple way to manage these cases, see ADVANCED FEATURES below.

In my opinion, the cleaner thing to do is to set the server to use MyISAM tables or -if you can't modify the server- use the `--mysql-force-myisam` command line option of imdbpy2sql.py. Anyway, if you really need to use InnoDB, in the server-side settings I recommend to set innodb_file_per_table to "true".

Beware that the conversion will be extremely slow (some hours), but still faster than using InnoDB from the start. You can use the "–mysql-innodb" command line option to force the creation of a database with MyISAM tables, converted at the end into InnoDB.

[Microsoft SQL Server/SQLExpress]

If you get and error about how wrong and against nature the blasphemous act of inserting an identity key is, you can try to fix it with the new custom queries support; see ADVANCED FEATURES below.

As a shortcut, you can use the "–ms-sqlserver" command line option to set all the needed options.

[SQLite speed-up]

For some reason, SQLite is really slow, except when used with transactions; you can use the "–sqlite-transactions" command line option to obtain acceptable performance. The same command also turns off "PRAGMA synchronous".

SQLite seems to hugely benefit from the use of a non-journaling filesystem and/or of a ramdisk/tmpfs: see the generic suggestions discussed above in the Timing section.

[SQLite failure]

It seems that with older versions of the python-sqlite package, the first run may fail; if you get a DatabaseError exception saying "no such table", try running again the command with the same arguments. Double funny, huh? ;-)

[data truncated]

If you get an insane amount (hundreds or thousands, on various text columns) of warnings like these:

> imdbpy2sql.py:727: Warning: Data truncated for column 'person_role' at row 4979
> CURS.executemany(self.sqlString, self.converter(self.values()))

you probably have a problem with the configuration of your database. The error comes from strings that get cut at the first non-ASCII character (and so you're losing a lot of information).

To solves this problem, you must be sure that your database server is set up properly, with the use library/client configured to communicate with the server in a consistent way. For example, for MySQL you can set:

```
character-set-server   = utf8
default-collation      = utf8_unicode_ci
default-character-set  = utf8
```

or even:

```
character-set-server   = latin1
default-collation      = latin1_bin
default-character-set  = latin1
```

[adult titles]

Beware that, while running, the imdbpy2sql.py script will output a lot of strings containing both person names and movie titles. The script has absolutely no way of knowing that the processed title is an adult-only movie, so... if you leave it on and your little daughter runs to you screaming "daddy! daddy! what kind of animals does Rocco train in the documentary 'Rocco: Animal Trainer 17'???"... well, it's not my fault! ;-)

Advanced features

With the -e (or –execute) command line argument you can specify custom queries to be executed at certain times, with the syntax:

```
-e "TIME:[OPTIONAL_MODIFIER:]QUERY"
```

where TIME is one of: 'BEGIN', 'BEFORE_DROP', 'BEFORE_CREATE', 'AFTER_CREATE', 'BEFORE_MOVIES', 'BEFORE_CAST', 'BEFORE_RESTORE', 'BEFORE_INDEXES', 'END'.

The only available OPTIONAL_MODIFIER is 'FOR_EVERY_TABLE' and it means that the QUERY command will be executed for every table in the database (so it doesn't make much sense to use it with BEGIN, BEFORE_DROP or BEFORE_CREATE time...), replacing the "%(table)s" text in the QUERY with the appropriate table name.

Other available TIMEs are: 'BEFORE_MOVIES_TODB', 'AFTER_MOVIES_TODB', 'BEFORE_PERSONS_TODB', 'AFTER_PERSONS_TODB', 'BEFORE_CHARACTERS_TODB', 'AFTER_CHARACTERS_TODB', 'BEFORE_SQLDATA_TODB', 'AFTER_SQLDATA_TODB', 'BEFORE_AKAMOVIES_TODB' and 'AFTER_AKAMOVIES_TODB'; they take no modifiers. Special TIMEs 'BEFORE_EVERY_TODB' and 'AFTER_EVERY_TODB' apply to every BEFORE_* and AFTER_* TIME above mentioned.

These commands are executed before and after every _toDB() call in their respective objects (CACHE_MID, CACHE_PID and SQLData instances); the "%(table)s" text in the QUERY is replaced as above.

You can specify so many -e arguments as you need, even if they refer to the same TIME: they will be executed from the first to the last. Also, always remember to correctly escape queries: after all you're passing it on the command line!

E.g. (ok, quite a silly example...):

```
-e "AFTER_CREATE:SELECT * FROM title;"
```

The most useful case is when you want to convert the tables of a MySQL from MyISAM to InnoDB:

```
-e "END:FOR_EVERY_TABLE:ALTER TABLE %(table)s ENGINE=InnoDB;"
```

If your system uses InnoDB by default, you can trick it with:

```
-e "AFTER_CREATE:FOR_EVERY_TABLE:ALTER TABLE %(table)s ENGINE=MyISAM;" -e "END:FOR_
↪EVERY_TABLE:ALTER TABLE %(table)s ENGINE=InnoDB;"
```

You can use the "–mysql-innodb" command line option as a shortcut of the above command.

Cool, huh?

Another possible use is to fix a problem with Microsoft SQLServer/SQLExpress. To prevent errors setting IDENTITY fields, you can run something like this:

```
-e 'BEFORE_EVERY_TODB:SET IDENTITY_INSERT %(table)s ON' -e 'AFTER_EVERY_TODB:SET␣
↪IDENTITY_INSERT %(table)s OFF'
```

You can use the "–ms-sqlserver" command line option as a shortcut of the above command.

To use transactions to speed-up SQLite, try:

```
-e 'BEFORE_EVERY_TODB:BEGIN TRANSACTION;' -e 'AFTER_EVERY_TODB:COMMIT;'
```

Which is also the same thing the command line option "–sqlite-transactions" does.

CSV files

Note: Keep in mind that not all database servers support this.

Moreover, you can run into problems. For example, if you're using PostgreSQL, your server process will need read access to the directory where the CSV files are stored.

To create the database using a set of CSV files, run `imdbpy2sql.py` as follows:

```
imdbpy2sql.py -d /dir/with/plainTextDataFiles/ -u URI \
      -c /path/to/the/csv_files_dir/
```

The created CSV files will be imported near the end of processing. After the import is finished, you can safely remove these files.

Since version 4.5, it's possible to separate the two steps involved when using CSV files:

- With the `--csv-only-write` command line option, the old database will be truncated and the CSV files saved, along with imdbID information.
- With the `--csv-only-load` option, these saved files can be loaded into an existing database (this database MUST be the one left almost empty by the previous run).

Beware that right now the whole procedure is not very well tested. For both commands, you still have to specify the whole `-u URI -d /path/plainTextDataFiles/ -c /path/CSVfiles/` arguments.

3.2 Development

If you intend to do development on the IMDbPY package, it's recommended that you create a virtual environment for it. For example:

```
python -m venv ~/.virtualenvs/imdbpy
. ~/.virtualenvs/imdbpy/bin/activate
```

In the virtual environment, install IMDbPY in editable mode and include the extra packages. In the top level directory of the project (where the `setup.py` file resides), run:

```
pip install -e .[dev,doc,test]
```

I wanted to stay independent from the source of the data for a given movie/person/character/company, so the `imdb.IMDb()` function returns an instance of a class that provides specific methods to access a given data source (web server, SQL database, etc.).

Unfortunately this means that the `movieID` in the `Movie` class, the `personID` in the `Person` class, and the `characterID` in the `Character` class depend on the data access system being used. So, when a movie, person, or character object is instantiated, the `accessSystem` instance variable is set to a string used to identify the used data access system.

3.2.1 How to extend

To introduce a new data access system, you have to write a new package inside the "parser" package; this new package must provide a subclass of the imdb.IMDb class which must define at least the following methods:

`_search_movie(title)` To search for a given title; must return a list of (movieID, {movieData}) tuples.

`_search_episode(title)` To search for a given episode title; must return a list of (movieID, {movieData}) tuples.

`_search_person(name)` To search for a given name; must return a list of (movieID, {personData}) tuples.

`_search_character(name)` To search for a given character's name; must return a list of (characterID, {characterData}) tuples.

`_search_company(name)` To search for a given company's name; must return a list of (companyID, {companyData}) tuples.

`get_movie_*(movieID)` A set of methods, one for every set of information defined for a Movie object; should return a dictionary with the relative information.

This dictionary can contain some optional keys:

- 'data': must be a dictionary with the movie info
- 'titlesRefs': a dictionary of 'movie title': movieObj pairs
- 'namesRefs': a dictionary of 'person name': personObj pairs

`get_person_*(personID)` A set of methods, one for every set of information defined for a Person object; should return a dictionary with the relative information.

`get_character_*(characterID)` A set of methods, one for every set of information defined for a Character object; should return a dictionary with the relative information.

`get_company_*(companyID)` A set of methods, one for every set of information defined for a Company object; should return a dictionary with the relative information.

`_get_top_bottom_movies(kind)` Kind can be one of 'top' and 'bottom'; returns the related list of movies.

`_get_keyword(keyword)` Return a list of Movie objects with the given keyword.

`_search_keyword(key)` Return a list of keywords similar to the given key.

get_imdbMovieID(movieID) Convert the given movieID to a string representing the imdbID, as used by the IMDb web server (e.g.: '0094226' for Brian De Palma's "The Untouchables").

get_imdbPersonID(personID) Convert the given personID to a string representing the imdbID, as used by the IMDb web server (e.g.: '0000154' for "Mel Gibson").

get_imdbCharacterID(characterID) Convert the given characterID to a string representing the imdbID, as used by the IMDb web server (e.g.: '0000001' for "Jesse James").

get_imdbCompanyID(companyID) Convert the given companyID to a string representing the imdbID, as used by the IMDb web server (e.g.: '0071509' for "Columbia Pictures [us]").

_normalize_movieID(movieID) Convert the provided movieID in a format suitable for internal use (e.g.: convert a string to a long int).

NOTE: As a rule of thumb you *always* need to provide a way to convert a "string representation of the movieID" into the internally used format, and the internally used format should *always* be converted to a string, in a way or another. Rationale: A movieID can be passed from the command line, or from a web browser.

_normalize_personID(personID) idem

_normalize_characterID(characterID) idem

_normalize_companyID(companyID) idem

_get_real_movieID(movieID) Return the true movieID; useful to handle title aliases.

_get_real_personID(personID) idem

_get_real_characterID(characterID) idem

_get_real_companyID(companyID) idem

The class should raise the appropriate exceptions, when needed:

- `IMDbDataAccessError` must be raised when you cannot access the resource you need to retrieve movie info or you're unable to do a query (this is *not* the case when a query returns zero matches: in this situation an empty list must be returned).
- `IMDbParserError` should be raised when an error occurred parsing some data.

Now you've to modify the `imdb.IMDb` function so that, when the right data access system is selected with the "accessSystem" parameter, an instance of your newly created class is returned.

For example, if you want to call your new data access system "mysql" (meaning that the data are stored in a mysql database), you have to add to the imdb.IMDb function something like:

```
if accessSystem == 'mysql':
    from parser.mysql import IMDbMysqlAccessSystem
    return IMDbMysqlAccessSystem(*arguments, **keywords)
```

where "parser.mysql" is the package you've created to access the local installation, and "IMDbMysqlAccessSystem" is the subclass of imdb.IMDbBase.

Then it's possible to use the new data access system like:

```
from imdb import IMDb
i = IMDb(accessSystem='mysql')
results = i.search_movie('the matrix')
print(results)
```

Note: This is a somewhat misleading example: we already have a data access system for SQL database (it's called 'sql' and it supports MySQL, amongst others). Maybe I'll find a better example...

A specific data access system implementation can define its own methods. As an example, the IMDbHTTPAccessSystem that is in the parser.http package defines the method `set_proxy()` to manage the use a web proxy; you can use it this way:

```
from imdb import IMDb
i = IMDb(accessSystem='http') # the 'accessSystem' argument is not
                              # really needed, since "http" is the default.
i.set_proxy('http://localhost:8080/')
```

A list of special methods provided by the imdb.IMDbBase subclass, along with their description, is always available calling the `get_special_methods()` of the IMDb class:

```
i = IMDb(accessSystem='http')
print(i.get_special_methods())
```

will print a dictionary with the format:

```
{'method_name': 'method_description', ...}
```

3.2.2 How to test

IMDbPY has a test suite based on pytest. The simplest way to run the tests is to run the following command in the top level directory of the project:

```
pytest
```

You can execute a specific test module:

```
pytest tests/test_http_movie_combined.py
```

Or execute test functions that match a given keyword:

```
pytest -k cover
```

make

A `Makefile` is provided for easier invocation of jobs. The following targets are defined (among others, run "make" to see the full list):

test Run tests quickly with the default Python.

lint Check style with flake8.

docs Generate Sphinx HTML documentation, including API docs.

coverage Check code coverage quickly with the default Python.

clean Clean everything.

tox

Multiple test environments can be tested using tox:

```
tox
```

This will test all the environments listed in the `tox.ini` file. If you want to run all tests for a specific environment, for example python 3.4, supply it as an argument to tox:

```
tox -e py34
```

You can supply commands that will be executed in the given environment. For example, to run the test function that have the string "cover" in them using pypy3, execute:

```
tox -e pypy3 -- pytest -k cover
```

Or to get a Python prompt under Python 3.5 (with IMDbPY and all dependencies already installed), execute:

```
tox -e py35 -- python
```

S3 dataset

The tests will use the HTTP access system by default. If you would also like to test the database generated from the S3 dataset, define the `IMDBPY_S3_URI` environment variable:

```
IMDBPY_S3_URI='postgres://imdb@localhost/imdb' pytest
```

This will run the tests for both HTTP and S3 access systems.

3.2.3 How to translate

Note: You can (but you don't have to) use Transifex to manage/coordinate your translations: http://www.transifex.net/projects/p/imdbpy/

The `imdb.locale` package contains some scripts that are useful for building your own internationalization files:

- The `generatepot.py` script should be used only when the DTD is changed; it's used to create the `imdbpy.pot` file (the one that gets shipped is always up-to-date).
- You can copy the `imdbpy.pot` file as your language's `.po` file (for example `imdbpy-fr.po` for French) and modify it according to your language.
- Then you have to run the `rebuildmo.py` script (which is automatically executed at install time) to create the `.mo` files.

If you need to upgrade an existing translation, after changes to the `.pot` file (usually because the DTD was changed), you can use the `msgmerge` utility which is part of the GNU gettext suite:

```
msgmerge -N imdbpy-fr.po imdbpy.pot > new-imdbpy-fr.po
```

If you create a new translation or update an existing one, you can send it to the <imdbpy-devel@lists.sourceforge.net> mailing list, for inclusion in upcoming releases.

3.2.4 How to make a release

During development

setup.cfg

The `egg_info` section must include the lines below:

```
[egg_info]
tag_build = dev
tag_date = true
```

setup.py

The `version` variable must be set to the **next** version.

imdb/__init__.py

When a major fix or feature is committed, the `VERSION` and `__version__` variables must be updated to something in the form *{next.version}devISO8601DATE* (not mandatory, but...)

docs/Changelog.rst

When a major fix or feature is committed, the changelog must be updated.

When a new release is planned

setup.cfg

In the `egg_info` section, the lines mentioned above must be commented out.

setup.py

Not touched.

imdb/__init__.py

The *devISO8601DATE* part must be removed from the version variables.

docs/Changelog.rst

The date of the release has to be added.

How to release

- Commit the above changes.
- Add an annotated tag like *major.minor*; e.g.: `git tag -a 6.3` (the commit message is not important).
- `python3 setup.py sdist`
- `python3 setup.py bdist_wheel`
- `git push`
- `git push --tags`
- Don't forget to push both sources and tags to both the GitHub and Bitbucket repositories (they are kept in sync).
- Upload to pypi: `twine upload dist/IMDbPY-*` (you probably need a recent version of twine and the appropriate ~/.pypi file)
- The new tar.gz must also be uploaded to https://sourceforge.net/projects/imdbpy/ (along with a new "news").

communication

- access the web site with: *sftp ${your-sourceforge-username}@frs.sourceforge.net* and move to the *imdbpy_web/htdocs/*

- download *index.html* and add an *article* section, removing the one or more of the old ones
- upload the page
- add a news on https://sourceforge.net/p/imdbpy/news/new
- send an email to imdbpy-devel@lists.sourceforge.net and imdbpy-help@lists.sourceforge.net

After the release

setup.cfg

Uncomment the two lines again.

setup.py

Bump the `version` variable.

imdb/__init__.py

Bump the `VERSION` and `__version__` variables.

docs/Changelog.rst

Add a new section for the next release, on top.

After that, you can commit the above changes with a message like "version bump"

3.3 FAQs

Q Is IMDbPY compatible with Python 3?

A Yes. Actually, the versions after 6.0 are compatible only with Python 3. If you need an older, unmaintained, version for Python, see the imdbpy-legacy branch in the repository.

Q Why is the movieID (and other IDs) used in the old "sql" database not the same as the ID used on the IMDb.com site?

A First, a bit of nomenclature: "movieID" is the term we use for a unique identifier used by IMDbPY to manage a single movie (similar terms for other kinds of data such as "personID" for persons). An "imdbID" is the term we use for a unique identifier that the IMDb.com site uses for the same kind of data (e.g.: the 7-digit number in tt0094226, as seen in the URL for "The Untouchables").

When using IMDbPY to access the web ("http" data access system), movieIDs and imdbIDs are the same thing -beware that in this case a movieID is a string, with the leading zeroes.

Unfortunately, when populating a SQL database with data from the plain text data files, we don't have access to imdbIDs -since they are not distributed at all- and so we have to generate them ourselves (they are the "id" columns in tables like "title" or "name"). This means that these values are valid only for your current database: if you update it with a newer set of plain text data files, these IDs will surely change (and, by the way, they are integers). It's also obvious, now, that you can't exchange IDs between the "http" and the "sql" data access systems, and similarly you can't use imdbIDs with your local database or vice-versa.

Q When using a SQL database, what's the "imdb_id" (or something like that) column in tables like "title", "name" and so on?

A It's internally used by IMDbPY to remember the imdbID of a movie (the one used by the web site), once it has been encountered. This way, if IMDbPY is asked again about the imdbID of a movie (or person, or . . .), it won't have to contact the web site again.

When accessing the database, you'll use the numeric value of the "id" column, e.g. "movieID". Note that, to update the SQL database, you have to access it using a user who has write permission.

As a bonus, when possible, the values of the imdbIDs are saved between updates of the SQL database (using the imdbpy2sql.py script). Beware that it's tricky and not always possible, but the script does its best to succeed.

Q But what if I really need the imdbIDs, to use in my database?

A No, you don't. Search for a title, get its information. Be happy!

Q I have a great idea: Write a script to fetch all the imdbIDs from the web site! Can't you do it?

A Yeah, I can. But I won't. :-)

It would be quite easy to map every title on the web to its imdbID, but there are still lots of problems. First of all, every user will end up doing it for their own copy of the plain text data files (and this will make the imdbpy2sql.py script painfully slow and prone to all sort of problems). Moreover, the imdbIDs are unique and never reused, true, but movie titles _do_ change: to fix typos, to override working titles, to cope with a new movie with the same title release in the same year, not to mention cancelled or postponed movies.

Other than that, we'd have to do the same for persons, characters, and companies. Believe me: it doesn't make sense. Work on your local database using your movieIDs (or even better: don't mind about movieIDs and think in terms of searches and Movie instances!) and retrieve the imdbID only in the rare circumstances when you really need them (see the next FAQ). Repeat after me: I DON'T NEED ALL THE imdbIDs. :-)

Q When using a SQL database, how can I convert a movieID (whose value is valid only locally) to an imdbID (the ID used by the imdb.com site)?

A Various functions can be used to convert a movieID (or personID or other IDs) to the imdbID used by the web site. Example:

```
from imdb import IMDb
ia = IMDb('sql', uri=URI_TO_YOUR_SQL_DATABASE)
movie = ia.search_movie('The Untouchables')[0] # a Movie instance.
print('The movieID for The Untouchables:', movie.movieID)
print('The imdbID used by the site:', ia.get_imdbMovieID(movie.movieID))
print('Same ID, smarter function:', ia.get_imdbID(movie))
```

It goes without saying that `get_imdbMovieID` method has some sibling methods: `get_imdbPersonID`, `get_imdbCompanyID` and `get_imdbCharacterID`. Also notice that the `get_imdbID` method is smarter, and takes any kind of instance (the other functions need a movieID, personID, ...)

Another method that will try to retrieve the imdbID is `get_imdbURL`, which works like `get_imdbID` but returns a URL.

In case of problems, these methods will return None.

Q I have a movie title (in the format used by the plain text data files) or other kind of data (like a person/character/company name) and I want to get its imdbID. How can I do it?

A The safest thing is probably to do a normal search on IMDb (using the "http" data access system of IMDbPY) and see if the first item is the correct one. You can also try the "title2imdbID" method (and similar) of the IMDb instance (no matter if you're using "http" or "sql"), but expect some failures -in which case it will return None.

Q I have an URL (of a movie, person or something else), how can I get a Movie/Person/... instance?

A Import the `imdb.helpers` module and use the `get_byURL` function.

Q I'm writing an interface based on IMDbPY and I have problems handling encoding, chars conversions, replacements of references and so on.

A See the many functions in the imdb.helpers module.

3.4 Contributors

3.4.1 Authors

People who contributed a substantial amount of work and share the copyright over some portions of the code:

Davide Alberani <da –> erlug.linux.it>

Main author and project leader.

H. Turgut Uyar <uyar –> tekir.org>

The whole "http" data access system (using a DOM and XPath-based approach) is based on his work. The imdbpykit interface was mostly written by him and he holds the copyright over the whole code (with some portions shared with others). He provided the tox testsuite.

Giuseppe "Cowo" Corbelli <cowo –> lugbs.linux.it>

Provided a lot of code and hints to integrate IMDbPY with SQLObject, working on the imdbpy2sql.py script and the dbschema.py module.

Beside Turgut, Giuseppe and me, the following people are listed as developers for the IMDbPY project on sourceforge and may share copyright on some (minor) portions of the code:

Alberto Malagoli

Developed the new web site, and detains the copyright of it, and provided helper functions and other code.

Martin Kirst <martin.kirst –> s1998.tu-chemnitz.de>

Has done an important refactoring of the imdbpyweb program and shares with me the copyright on the whole program.

Jesper Nøhr <jesper –> noehr.org>

Provided extensive testing and some patches for the "http" data access system.

Joachim Selke <j.selke –> tu-bs.de>

Many tests on IBM DB2 and work on the CSV support.

Timo Schulz <gnuknight –> users.sourceforge.net>

Did a lot of work "sql", DB2 and CSV support and extensive analysis aimed at diff files support.

Roy Stead <roystead247 –> gmail.com>

Provided the download_applydiffs.py script.

3.4.2 Translators

Additional translations were provided by:

- strel (Spanish)
- Stéphane Aulery (French)
- RainDropR (Arabic)
- Atanas Kovachki (Bulgarian)

- lukophron (French)
- Raphael (German)

3.4.3 Credits

First of all, I want to thank all the package maintainers, and especially Ana Guerrero. Another big thanks to the developers who used IMDbPY for their projects and research; they can be found here: https://imdbpy.sourceforge.io/ecosystem.html

Other very special thanks go to some people who followed the development of IMDbPY very closely, providing hints and insights: Ori Cohen, James Rubino, Tero Saarni, and Jesper Noer (for a lot of help, and also for the wonderful https://bitbucket.org/); and let's not forget all the translators on https://www.transifex.com/davide_alberani/imdbpy/.

Below is a list of people who contributed with bug reports, small patches, and hints (kept in reverse order since IMDbPY 4.5):

- Ali Momen Sani for a report about mini biography
- David Runge for package tests and hints
- antonioforte for a bug report about XML output
- Jakub Synowiec for multiple bug reports and patches
- Piotr Staszewski for multiple bug reports and patches
- tiagoaquinofl and rednibia for extensive debugging on SSL certificate issues
- Tim King for a report about birth and death dates.
- Lars Gustäbel for a report about series seasons.
- Filip Bačić for a report about full-size headshot
- Matthew Clapp for a report about pip installation
- Jannik S for a report on tech parser
- Brad Pirtle, Adrien C. and Markus-at-GitHub for improvements to full-size covers
- Tim Belcher for a report about forgotten debug code.
- Paul Jensen for many bug reports about tv series.
- Andrew D Bate for documentation on how to reintroduce foreign keys.
- yiqingzhu007 for a bug report about synopsis.
- David Runge for managing the Arch Linux package.
- enriqueav for fixes after the IMDb redesign.
- Piotr Staszewski for a fix for external sites parser.
- Mike Christopher for the user reviews parser.
- apelord for a parser for full credits.
- Mike Christopher for a patch for synopsis parser.
- Damon Brodie for a bug report about technical parser.
- Sam Petulla for a bug report about searching for keywords.
- zoomorph for an improvement for parsing your own votes.
- Fabrice Laporte for a bug report on setup.py.

- Wael Sinno for a patch for parsing movie-links.
- Tool Man, for a fix on sound track parsing.
- Rafael Lopez for a series of fixes on top/bottom lists.
- Derek Duoba for a bug report about XML output.
- Cody Hatfield for a parser for the Persons's resume page.
- Mystfit for a fix handling company names.
- Troy Deck for a path for MySQL.
- miles82 for a patch on metascore parsing.
- Albert Claret for the parser of the critic reviews page.
- Shobhit Singhal for fixes in parsing biographies and plots.
- Dan Poirier for documentation improvements.
- Frank Braam for a fix for MSSQL.
- Darshana Umakanth for a bug report the search functions.
- Osman Boyaci for a bug report on movie quotes.
- Mikko Matilainen for a patch on encodings.
- Roy Stead for the download_applydiffs.py script.
- Matt Keenan for a report about i18n in search results.
- belgabor for a patch in the goofs parser.
- Ian Havelock for a bug report on charset identification.
- Mikael Puhakka for a bug report about foreign language results in a search.
- Wu Mao for a bug report on the GAE environment.
- legrostdg for a bug report on the new search pages.
- Haukur Páll Hallvarðsson for a patch on query parameters.
- Arthur de Peretti-Schlomoff for a list of French articles and fixes to Spanish articles.
- John Lambert, Rick Summerhill and Maciej for reports and fixes for the search query.
- Kaspars "Darklow" Sprogis for an impressive amount of tests and reports about bugs parsing the plain text data files and many new ideas.
- Damien Stewart for many bug reports about the Windows environment.
- Vincenzo Ampolo for a bug report about the new imdbIDs save/restore queries.
- Tomáš Hnyk for the idea of an option to reraise caught exceptions.
- Emmanuel Tabard for ideas, code and testing on restoring imdbIDs.
- Fabian Roth for a bug report about the new style of episodes list.
- 25. Josuin for a bug report on missing info in crazy credits file.
- Arfrever Frehtes Taifersar Arahesis for a patch for locales.
- Gustaf Nilsson for bug reports about BeautifulSoup.
- Jernej Kos for patches to handle "in production" information and birth/death years.

- Saravanan Thirumuruganathan for a bug report about genres in mobile.
- Paul Koan, for a bug report about DVD pages and movie references.
- Greg Walters for a report about a bug with queries with too many results.
- Olav Kolbu for tests and report about how the IMDb.com servers reply to queries made with and without cookies.
- Jef "ofthelit", for a patch for the reduce.sh script bug reports for Windows.
- Reiner Herrmann for benchmarks using SSD hard drives.
- Thomas Stewart for some tests and reports about a bug with charset in the plain text data files.
- Ju-Hee Bae for an important series of bug reports about the problems derived by the last IMDb's redesign.
- Luis Liras and Petite Abeille for a report and a bugfix about imdbpy2sql.py used with SQLite and SQLObject.
- Kevin S. Anthony for a bug report about episodes list.
- Bhupinder Singh for a bug report about exception handling in Python 2.4.
- Ronald Hatcher for a bug report on the GAE environment.
- Ramusus for a lot of precious bug reports.
- Laurent Vergne for a hint about InnoDB, MyISAM and foreign keys.
- Israel Fruch for patches to support the new set of parsers.
- Inf3cted MonkeY, for a bug report about 'vote details'.
- Alexmipego, for suggesting to add a md5sum to titles and names.
- belgabortm for a bug report about movies with multiple 'countries'.
- David Kaufman for an idea to make the 'update' method more robust.
- Dustin Wyatt for a bug with SQLite of Python 2.6.
- Julian Scheid for bug reports about garbage in the ptdf.
- Adeodato Simó for a bug report about the new imdb.com layout.
- Josh Harding for a bug report about the new imdb.com layout.
- Xavier Naidoo for a bug report about top250 and BeautifulSoup.
- Basil Shubin for hints about a new helper function.
- Mark Jeffery, for some help debugging a lxml bug.
- Hieu Nguyen for a bug report about fetching real imdbIDs.
- Rdian06 for a patch for movies without plot authors.
- Tero Saarni, for the series 60 GUI and a lot of testing and debugging.
- Ana Guerrero, for maintaining the official debian package.
- H. Turgut Uyar for a number of bug reports and a lot of work on the test-suite.
- Ori Cohen for some code and various hints.
- Jesper Nøhr for a lot of testing, especially on 'sql'.
- James Rubino for many bug reports.
- Cesare Lasorella for a bug report about newer versions of SQLObject.
- Andre LeBlanc for a bug report about airing date of tv series episodes.

- aow for a note about some misleading descriptions.
- Sébastien Ragons for tests and reports.
- Sridhar Ratnakumar for info about PKG-INF.
- neonrush for a bug parsing Malcolm McDowell filmography!
- Alen Ribic for some bug reports and hints.
- Joachim Selke for some bug reports with SQLAlchemy and DB2 and a lot of testing and debugging of the ibm_db driver (plus a lot of hints about how to improve the imdbpy2sql.py script).
- Karl Newman for bug reports about the installer of version 4.5.
- Saruke Kun and Treas0n for bug reports about 'Forbidden' errors from the imdb.com server.
- Chris Thompson for some bug reports about summary() methods.
- Mike Castle for performace tests with SQLite and numerous hints.
- Indy (indyx) for a bug about series cast parsing using BeautifulSoup.
- Yoav Aviram for a bug report about tv mini-series.
- Arjan Gijsberts for a bug report and patch for a problem with movies listed in the Bottom 100.
- Helio MC Pereira for a bug report about unicode.
- Michael Charclo for some bug reports performing 'http' queries.
- Amit Belani for bug reports about plot outline and other changes.
- Matt Warnock for some tests with MySQL.
- Mark Armendariz for a bug report about too long field in MySQL db and some tests/analyses.
- Alexy Khrabrov, for a report about a subtle bug in imdbpy2sql.py.
- Clark Bassett for bug reports and fixes about the imdbpy2sql.py script and the cutils.c C module.
- mumas for reporting a bug in summary methods.
- Ken R. Garland for a bug report about 'cover url' and a lot of other hints.
- Steven Ovits for hints and tests with Microsoft SQL Server, SQLExpress and preliminary work on supporting diff files.
- Fredrik Arnell for tests and bug reports about the imdbpy2sql.py script.
- Arnab for a bug report in the imdbpy2sql.py script.
- Elefterios Stamatogiannakis for the hint about transactions and SQLite, to obtain an impressive improvement in performances.
- Jon Sabo for a bug report about unicode and the imdbpy2sql.py script and some feedback.
- Andrew Pendleton for a report about a very hideous bug in the imdbpy2sql.py (garbage in the plain text data files + programming errors + utf8 strings + postgres).
- Ataru Moroboshi ;-) for a bug report about role/duty and notes.
- Ivan Kedrin for a bug report about the analyze_title function.
- Hadley Rich for reporting bugs and providing patches for troubles parsing tv series' episodes and searching for tv series' titles.
- Jamie R. Rytlewski for a suggestion about saving imbIDs in 'sql'.
- Vincent Crevot, for a bug report about unicode support.

- Jay Klein for a bug report and testing to fix a nasty bug in the imdbpy2sql.py script (splitting too large data sets).
- Ivan Garcia for an important bug report about the use of IMDbPY within wxPython programs.
- Kessia Pinheiro for a bug report about tv series list of episodes.
- Michael G. Noll for a bug report and a patch to fix a bug retrieving 'plot keywords'.
- Alain Michel, for a bug report about search_*.py and get_*.py scripts.
- Martin Arpon and Andreas Schoenle for bug reports (and patches) about "runtime", "aka titles" and "production notes" information not being parsed.
- none none (dclist at gmail.com) for a useful hint and code to retrieve a movie/person object, given an URL.
- Sebastian Pölsterl, for a bug report about the cover url for tv (mini) series, and another one about search_* methods.
- Martin Kirst for many hints and the work on the imdbpyweb program.
- Julian Mayer, for a bug report and a patch about non-ascii chars.
- Wim Schut and "eccentric", for bug reports and a patches about movies' cover url.
- Alfio Ferrara, for a bug report about the get_first_movie.py script.
- Magnus Lie Hetland for an hint about the searches in sql package.
- Thomas Jadjewski for a bug report about the imdbpy2sql.py script.
- Trevor MacPhail, for a bug report about search_* methods and the ParserBase.parse method.
- Guillaume Wisniewski, for a bug report.
- Kent Johnson, for a bug report.
- Andras Bali, for the hint about the "plot outline" information.
- Nick S. Novikov, who provided the Windows installer until I've managed to set up a Windows development environment.
- Simone Bacciglieri, who downloaded the plain text data files for me.
- Carmine Noviello, for some design hints.
- "Basilius" for a bug report.
- Davide for a bug report.

3.4.4 Donations

We'd like to thank the following people for their donations:

- Paulina Wadecka
- Oleg Peil
- Diego Sarmentero
- Fabian Winter
- Lacroix Scott

3.5 Changelog

- What's new in release 6.6 "Stranger Things" (05 Aug 2018)

  [general]

  - #154: exclude docs and etc directories from packaging
  - introduce 'https' as an alias for 'http'
  - #151: the 'in' operator also considers key names
  - #172: fix for ASCII keys in XML output
  - #174: improve XML output
  - #179: introduce Travis CI at https://travis-ci.org/alberanid/imdbpy

  [http]

  - #149: store person birth and death dates in ISO8601 format
  - #166: fix birth and death dates without itemprop attributes
  - #160: fix series seasons list
  - #155 and #165: ignore certificate to prevent validation errors
  - #156: fix tech parser
  - #157: full-size headshot for persons
  - #161: fix string/unicode conversion in Python 2.7
  - #173: raw akas and raw release dates fields
  - #178: fix mini biography parser

  [s3]

  - #158: fetch and search AKAs
  - update the goodies/download-from-s3 script to use the datasets.imdbws.com site
- What's new in release 6.5 "Poultrygeist: Night of the Chicken Dead" (15 Apr 2018)

  [general]

  - converted the documentation to Sphinx rst format

  [http]

  - fix title parser for in-production movies
  - parsers are based on piculet
  - improve collection of full-size cover images
- What's new in release 6.4 "Electric Dreams" (14 Mar 2018)

  [http]

  - remove obsolete parsers
  - remove Character objects
  - fix for search parsers

- What's new in release 6.3 "Altered Carbon" (27 Feb 2018)

  [general]

  - documentation updates
  - introduced the 'imdbpy' CLI
  - s3 accessSystem to access the new dataset from IMDb

  [http]

  - fixes for IMDb site redesign
  - Person parser fixes
  - users review parser
  - improve external sites parser
  - switch from akas.imdb.com domain to www.imdb.com
  - fix for synopsis
  - fix for tv series episodes

  [s3]

  - ability to import and access all the information
- What's new in release 6.2 "Justice League" (19 Nov 2017)

  [general]

  - introduce check for Python version
  - SQLAlchemy can be disabled using –without-sqlalchemy
  - fix #88: configuration file parser
  - update documentation

  [http]

  - fixed ratings parser
  - moved cookies from json to Python source
- What's new in release 6.1

  - skipped version 6.1 due to a wrong release on pypi
- What's new in release 6.0 "Life is Strange" (12 Nov 2017)

  [general]

  - now IMDbPY is a Python 3 package
  - simplified the code base: #61
  - remove dependencies: SQLObject, BeautifulSoup, C compiler
  - introduced a tox testsuite
  - fix various parsers
- What's new in release 5.1 "Westworld" (13 Nov 2016)

  [general]

  - fix for company names containing square brackets.

  - fix XML output when imdb long name is missing.
  - fixes #33: unable to use –without-sql

  [http]

  - fix birth/death dates parsing.
  - fix top/bottom lists.
  - Persons's resume page parser (courtesy of codynhat)
  - fixes #29: split color info
  - parser for "my rating" (you have to use your own cookies)

  [sql]

  - sound track list correctly identified.
  - fixes #50: process splitted data in order
  - fixes #53: parser for movie-links

- What's new in release 5.0 "House of Cards" (02 May 2014)

  [general]

  - Spanish, French, Arabic, Bulgarian and German translations.
  - Introduced the list of French articles.
  - fix for GAE.
  - download_applydiffs.py script.
  - fixed wrong handling of encoding in episode titles
  - renamed README.utf8 to README.unicode

  [http]

  - fixed searches (again).
  - search results are always in English.
  - updated the cookies.
  - support for obtaining metacritic score and URL.
  - fixed goofs parser.
  - fixed url for top250.
  - fixes for biography page.
  - fix for quotes.
  - better charset identification.
  - category and spoiler status for goofs.
  - changed query separators from ; to &.
  - fix for episodes of unknown seasons.
  - new cookie.

  [mobile]

  - fixed searches.

[sql]

- fix for MSSQL

- What's new in release 4.9 "Iron Sky" (15 Jun 2012)

  [general]

  - urls used to access the IMDb site can be configured.
  - helpers function to handle movie AKAs in various languages (code by Alberto Malagoli).
  - renamed the 'articles' module into 'linguistics'.
  - introduced the 'reraiseExceptions' option, to re-raise evey caught exception.

  [http]

  - fix for changed search parameters.
  - introduced a 'timeout' parameter for connections to the web server.
  - fix for business information.
  - parser for the new style of episodes list.
  - unicode searches handled as iso8859-1.
  - fix for garbage in AKA titles.

  [sql]

  - vastly improved the store/restore of imdbIDs; now it should be faster and more accurate.
  - now the 'name' table contains a 'gender' field that can be 'm', 'f' or NULL.
  - fix for nicknames.
  - fix for missing titles in the crazy credits file.
  - handled exceptions creating indexes, foreign keys and executing custom queries.
  - fixed creation on index for keywords.
  - excluded {{SUSPENDED}} titles.

- What's new in release 4.8.2 "The Big Bang Theory" (02 Nov 2011)

  [general]

  - fixed install path of locales.

  [http]

  - removed debug code.

- What's new in release 4.8 "Super" (01 Nov 2011)

  [general]

  - fix for a problem managing exceptions with Python 2.4.
  - converted old-style exceptions to instances.
  - enanchements for the reduce.sh script.
  - added notes about problems connecting to IMDb's web servers.
  - improvements in the parsers of movie titles.
  - improvements in the parser of person names.

[http]

- potential fix for GAE environment.
- handled the new style of "in production" information.
- fix for 'episodes' list.
- fix for 'episodes rating'.
- fix for queries that returned too many results.
- fix for wrong/missing references.
- removed no more available information set "amazon reviews" and "dvd".
- fix for cast of tv series.
- fix for title of tv series.
- now the beautiful parses work again.

[httpThin]

- removed "httpThin", falling back to "http".

[mobile]

- fix for missing headshots.
- fix for rating and number of votes.
- fix for missing genres.
- many other fixes to keep up-to-date with the IMDb site.

[sql]

- fix for a nasty bug parsing notes about character names.
- fixes for SQLite with SQLOjbect.

- What's new in release 4.7 "Saw VI" (23 Jan 2011)

  [http]

  - first fixes for the new set of parsers.
  - first changes to support the new set of web pages.
  - fix for lists of uncategorized episodes.
  - fix for movies with multiple countries.
  - fix for the currentRole property.
  - more robust handling for vote details.

  [mobile]

  - first fixes for the new set of parsers.

  [sql]

  - the tables containing titles and names (and akas) now include a 'md5sum' column calculated on the "long imdb canonical title/name".

- What's new in release 4.6 "The Road" (19 Jun 2010)

  [general]

  - introduced the 'full-size cover url' and 'full-size headshot' keys for Movie, Person and Character instances.
  - moved the development to a Mercurial repository.
  - introduced the parseXML function in the imdb.helpers module.
  - now the asXML method can exclude dynamically generated keys.
  - rationalized the use of the 'logging' and 'warnings' modules.
  - the 'update' method no longer raises an exception, if asked for an unknown info set.

  [http/mobile]

  - removed new garbage from the imdb pages.
  - support new style of akas.
  - fix for the "trivia" page.
  - fixes for searches with too many results.

  [sql]

  - fixes for garbage in the plain text data files.
  - support for SQLite shipped with Python 2.6.

- What's new in release 4.5.1 "Dollhouse" (01 Mar 2010)

  [general]

  - reintroduced the ez_setup.py file.
  - fixes for AKAs on 'release dates'.
  - added the dtd.

- What's new in release 4.5 "Invictus" (28 Feb 2010)

  [general]

  - moved to setuptools 0.6c11.
  - trying to make the SVN release versions work fine.
  - http/mobile should work in GAE (Google App Engine).
  - added some goodies scripts, useful for programmers (see the docs/goodies directory).

  [http/mobile]

  - removed urllib-based User-Agent header.
  - fixes for some minor changes to IMDb's html.
  - fixes for garbage in movie quotes.
  - improvements in the handling of AKAs.

  [mobile]

  - fixes for AKAs in search results.

  [sql]

  - fixes for bugs restoring imdbIDs.
  - first steps to split CSV creation/insertion.

- What's new in release 4.4 "Gandhi" (06 Jan 2010)

  [general]

  - introduced a logging facility; see README.logging.
  - the 'http' and 'mobile' should be a lot more robust.

  [http]

  - fixes for the n-th set of changes to IMDb's HTML.
  - improvements to perfect-match searches.
  - slightly simplified the parsers for search results.

  [mobile]

  - fixes for the n-th set of changes to IMDb's HTML.
  - slightly simplified the parsers for search results.

  [sql]

  - movies' keywords are now correctly imported, using CSV files.
  - minor fixes to handle crap in the plain text data files.
  - removed an outdate parameter passed to SQLObject.
  - made imdbpy2sql.py more robust in some corner-cases.
  - fixes for the Windows environment.
- What's new in release 4.3 "Public Enemies" (18 Nov 2009)

  [general]

  - the installer now takes care of .mo files.
  - introduced, in the helpers module, the functions keyToXML and translateKey, useful to translate dictionary keys.
  - support for smart guessing of the language of a movie title.
  - updated the DTD.

  [http]

  - fixed a lot of bugs introduced by the new IMDb.com design.
  - nicer handling of HTTP 404 response code.
  - fixed parsers for top250 and bottom100 lists.
  - fixed a bug parsing AKAs.
  - fixed misc bugs.

  [mobile]

  - removed duplicates in list of genres.

  [sql]

  - fixed a bug in the imdbpy2sql.py script using CSV files; the 'movie_info_idx' and 'movie_keyword' were left empty/with wrong data.
- What's new in release 4.2 "Battlestar Galactica" (31 Aug 2009)

  [general]

  - the 'local' data access system is gone. See README.local.
  - the imdb.parser.common package was removed, and its code integrated in imdb.parser.sql and in the imdbpy2sql.py script.
  - fixes for the installer.
  - the helpers module contains the fullSizeCoverURL function, to convert a Movie, Person or Character instance (or a URL in a string) in an URL to the full-size version of its cover/headshot. Courtesy of Basil Shubin.
  - used a newer version of msgfmt.py, to work around a hideous bug generating locales.
  - minor updates to locales.
  - updated the DTD to version 4.2.

  [http]

  - removed garbage at the end of quotes.
  - fixed problems parsing company names and notes.
  - keys in character's quotes dictionary are now Movie instances.
  - fixed a bug converting entities char references (affected BeautifulSoup).
  - fixed a long-standing bug handling & with BeautifulSoup.
  - top250 is now correctly parsed by BeautifulSoup.

  [sql]

  - fixed DB2 call for loading blobs/cblobs.
  - information from obsolete files are now used if and only if they refer to still existing titles.
  - the –fix-old-style-titles argument is now obsolete.

- What's new in release 4.1 "State Of Play" (02 May 2009)

  [general]

  - DTD definition.
  - support for locale.
  - support for the new style for movie titles ("The Title" and no more "Title, The" is internally used).
  - minor fix to XML code to work with the test-suite.

  [http]

  - char references in the &#xHEXCODE; format are handled.
  - fixed a bug with movies containing '....' in titles. And I'm talking about Malcolm McDowell's filmography!
  - 'airing' contains object (so the accessSystem variable is set).
  - 'tv schedule' ('airing') pages of episodes can be parsed.
  - 'tv schedule' is now a valid alias for 'airing'.
  - minor fixes for empty/wrong strings.

  [sql]

  - in the database, soundex values for titles are always calculated after the article is stripped (if any).

  - imdbpy2sql.py has the –fix-old-style-titles option, to handle files in the old format.
  - fixed a bug saving imdbIDs.

  [local]

  - the 'local' data access system should be considered obsolete, and will probably be removed in the next release.

- What's new in release 4.0 "Watchmen" (12 Mar 2009)

  [general]

  - the installer is now based on setuptools.
  - new functions get_keyword and search_keyword to handle movie's keywords (example scripts included).
  - Movie/Person/... keys (and whole instances) can be converted to XML.
  - two new functions, get_top250_movies and get_bottom100_movies, to retrieve lists of best/worst movies (example scripts included).
  - searching for movies and persons - if present - the 'akas' keyword is filled, in the results.
  - 'quotes' for movies is now always a list of lists.
  - the old set of parsers (based on sgmllib.SGMLParser) are gone.
  - fixed limitations handling multiple roles (with notes).
  - fixed a bug converting somethingIDs to real imdbIDs.
  - fixed some summary methods.
  - updates to the documentation.

  [http]

  - adapted BeautifulSoup to lxml (internally, the lxml API is used).
  - currentRole is no longer populated, for non-cast entries (everything ends up into .notes).
  - fixed a bug search for too common terms.
  - fixed a bug identifying 'kind', searching for titles.
  - fixed a bug parsing airing dates.
  - fixed a bug searching for company names (when there's a direct hit).
  - fixed a bug handling multiple characters.
  - fixed a bug parsing episode ratings.
  - nicer keys for technical details.
  - removed the 'agent' page.

  [sql]

  - searching for a movie, the original titles are returned, instead of AKAs.
  - support for Foreign Keys.
  - minor changes to the db's design.
  - fixed a bug populating tables with SQLAlchemy.
  - imdbpy2sql.py shows user time and system time, along with wall time.

  [local]

  - searching for a movie, the original titles are returned, instead of AKAs.

- What's new in release 3.9 "The Strangers" (06 Jan 2009)

  [general]

  - introduced the search_episode method, to search for episodes' titles.
  - movie['year'] is now an integer, and no more a string.
  - fixed a bug parsing company names.
  - introduced the helpers.makeTextNotes function, useful to pretty-print strings in the 'TEXT::NOTE' format.

  [http]

  - fixed a bug regarding movies listed in the Bottom 100.
  - fixed bugs about tv mini-series.
  - fixed a bug about 'series cast' using BeautifulSoup.

  [sql]

  - fixes for DB2 (with SQLAlchemy).
  - improved support for movies' aka titles (for series).
  - made imdbpy2sql.py more robust, catching exceptions even when huge amounts of data are skipped due to errors.
  - introduced CSV support in the imdbpy2sql.py script.

- What's new in release 3.8 "Quattro Carogne a Malopasso" (03 Nov 2008)

  [http]

  - fixed search system for direct hits.
  - fixed IDs so that they always are str and not unicode.
  - fixed a bug about plot without authors.
  - for pages about a single episode of a series, "Series Crew" are now separated items.
  - introduced the preprocess_dom method of the DOMParserBase class.
  - handling rowspan for DOMHTMLAwardsParser is no more a special case.
  - first changes to remove old parsers.

  [sql]

  - introduced support for SQLAlchemy.

  [mobile]

  - fixed multiple 'nick names'.
  - added 'aspect ratio'.
  - fixed a "direct hit" bug searching for people.

  [global]

  - fixed search_* example scripts.
  - updated the documentation.

- What's new in release 3.7 "Burn After Reading" (22 Sep 2008)

  [http]

  - introduced a new set of parsers, active by default, based on DOM/XPath.
  - old parsers fixed; 'news', 'genres', 'keywords', 'ratings', 'votes', 'tech', 'taglines' and 'episodes'.

  [sql]

  - the pure python soundex function now behaves correctly.

  [general]

  - minor updates to the documentation, with an introduction to the new set of parsers and notes for packagers.

- What's new in release 3.6 "RahXephon" (08 Jun 2008)

  [general]

  - support for company objects for every data access systems.
  - introduced example scripts for companies.
  - updated the documentation.

  [http and mobile]

  - changes to support the new HTML for "plot outline" and some lists of values (languages, genres, ...)
  - introduced the set_cookies method to set cookies for IMDb's account and the del_cookies method to remove the use of cookies; in the imdbpy.cfg configuration file, options "cookie_id" and "cookie_uu" can be set to the appropriate values; if "cookie_id" is None, no cookies are sent.
  - fixed parser for 'news' pages.
  - fixed minor bug fetching movie/person/character references.

  [http]

  - fixed a search problem, while not using the IMDbPYweb's account.
  - fixed bugs searching for characters.

  [mobile]

  - fixed minor bugs parsing search results.

  [sql]

  - fixed a bug handling movieIDs, when there are some inconsistencies in the plain text data files.

  [local]

  - access to 'mpaa' and 'miscellaneous companies' information.

- What's new in release 3.5 "Blade Runner" (19 Apr 2008)

  [general]

  - first changes to work on Symbian mobile phones.
  - now there is an imdb.available_access_systems() function, that can be used to get a list of available data access systems.
  - it's possible to pass 'results' as a parameter of the imdb.IMDb function; it sets the number of results to return for queries.
  - fixed summary() method in Movie and Person, to correctly handle unicode chars.

  - the helpers.makeObject2Txt function now supports recursion over dictionaries.
  - cutils.c MXLINELEN increased from 512 to 1024; some critical strcpy replaced with strncpy.
  - fixed configuration parser to be compatible with Python 2.2.
  - updated list of articles and some stats in the comments.
  - documentation updated.

  [sql]

  - fixed minor bugs in imdbpy2sql.py.
  - restores imdbIDs for characters.
  - now CharactersCache honors custom queries.
  - the imdbpy2sql.py's –mysql-force-myisam command line option can be used to force usage of MyISAM tables on InnoDB databases.
  - added some warnings to the imdbpy2sql.py script.

  [local]

  - fixed a bug in the fall-back function used to scan movie titles, when the cutils module is not available.
  - mini biographies are cut up to 2**16-1 chars, to prevent troubles with some MySQL servers.
  - fixed bug in characters4local.py, dealing with some garbage in the files.

- What's new in release 3.4 "Flatliners" (16 Dec 2007)

  [general]

  - *** NOTE FOR PACKAGERS *** in the docs directory there is the "imdbpy.cfg" configuration file, which should be installed in /etc or equivalent directory; the setup.py script *doesn't* manage its installation.
  - introduced a global configuration file to set IMDbPY's parameters.
  - supported characters using "sql" and "local" data access systems.
  - fixed a bug retrieving characterID from a character's name.

  [http]

  - fixed a bug in "release dates" parser.
  - fixed bugs in "episodes" parser.
  - fixed bugs reading "series years".
  - stricter definition for ParserBase._re_imdbIDmatch regular expression.

  [mobile]

  - fixed bugs reading "series years".
  - fixed bugs reading characters' filmography.

  [sql]

  - support for characters.

  [local]

  - support for characters.
  - introduced the characters4local.py script.

- What's new in release 3.3 "Heroes" (18 Nov 2007)

  [general]

  - first support for character pages; only for "http" and "mobile", so far.
  - support for multiple characters.
  - introduced an helper function to pretty-print objects.
  - added README.currentRole.
  - fixed minor bug in the __hash__ method of the _Container class.
  - fixed changes to some key names for movies.
  - introduced the search_character.py, get_character.py and get_first_character.py example scripts.

  [http]

  - full support for character pages.
  - fixed a bug retrieving some 'cover url'.
  - fixed a bug with multi-paragraphs biographies.
  - parsers are now instanced on demand.
  - accessSystem and modFunct are correctly set for every Movie, Person and Character object instanced.

  [mobile]

  - full support for character pages.

  [sql]

  - extended functionality of the custom queries support for the imdbpy2sql.py script to circumvent a problem with MS SQLServer.
  - introducted the "–mysql-innodb" and "–ms-sqlserver" shortcuts for the imdbpy2sql.py script.
  - introduced the "–sqlite-transactions" shortcut to activate transaction using SQLite which, otherwise, would have horrible performances.
  - fixed a minor bug with top/bottom ratings, in the imdbpy2sql.py script.

  [local]

  - filtered out some crap in the "quotes" plain text data files, which also affected sql, importing the data.

- What's new in release 3.2 "Videodrome" (25 Sep 2007)

  [global]

  - now there's an unique place where "akas.imdb.com" is set, in the main module.
  - introduced __version__ and VERSION in the main module.
  - minor improvements to the documentation.

  [http]

  - updated the main movie parser to retrieve the recently modified cast section.
  - updated the crazy credits parser.
  - fixed a bug retrieving 'cover url'.

  [mobile]

  - fixed a bug parsing people's filmography when only one duty was listed.

  - updated to retrieve series' creator.

  [sql]

  - added the ability to perform custom SQL queries at the command line of the imdbpy2sql.py script.
  - minor fixes for the imdbpy2sql.py script.

- What's new in release 3.1 "The Snake King" (18 Jul 2007)

  [global]

  - the IMDbPYweb account now returns a single item, when a search returns only one "good enough" match (this is the IMDb's default).
  - updated the documentation.
  - updated list of contributors and developers.

  [http]

  - supported the new result page for searches.
  - supported the 'synopsis' page.
  - supported the 'parents guide' page.
  - fixed a bug retrieving notes about a movie's connections.
  - fixed a bug for python2.2 (s60 mobile phones).
  - fixed a bug with 'Production Notes/Status'.
  - fixed a bug parsing role/duty and notes (also for httpThin).
  - fixed a bug retrieving user ratings.
  - fixed a bug (un)setting the proxy.
  - fixed 2 bugs in movie/person news.
  - fixed a bug in movie faqs.
  - fixed a bug in movie taglines.
  - fixed a bug in movie quotes.
  - fixed a bug in movie title, in "full cast and crew" page.
  - fixed 2 bugs in persons' other works.

  [sql]

  - hypothetical fix for a unicode problem in the imdbpy2sql.py script.
  - now the 'imdbID' fields in the Title and Name tables are restored, updating from an older version.
  - fixed a nasty bug handling utf-8 strings in the imdbpy2sql.py script.

  [mobile]

  - supported the new result page for searches.
  - fixed a bug for python2.2 (s60 mobile phones).
  - fixed a bug searching for persons with single match and no messages in the board.
  - fixed a bug parsing role/duty and notes.

- What's new in release 3.0 "Spider-Man 3" (03 May 2007)

  [global]

  - IMDbPY now works with the new IMDb's site design; a new account is used to access data; this affect a lot of code, especially in the 'http', 'httpThin' and 'mobile' data access systems.
  - every returned string should now be unicode; dictionary keywords are _not_ guaranteed to be unicode (but they are always 7bit strings).
  - fixed a bug in the __contains__ method of the Movie class.
  - fix in the analyze_title() function to handle malformed episode numbers.

  [http]

  - introduced the _in_content instance variable for objects instances of ParserBase, True when inside the tag. Opening and closing this pair of tags two methods, named _begin_content() and _end_content() are called with no parameters (by default, they do nothing).
  - in the utils module there's the build_person function, useful to create a Person instance from the tipical formats found in the IMDb's web site.
  - an analogue build_movie function can be used to instance Movie objects.
  - inverted the getRefs default - now if not otherwise set, it's False.
  - added a parser for the "merchandising" ("for sale") page for persons.
  - the 'rating' parser now collects also 'rating' and 'votes' data.
  - the HTMLMovieParser class (for movies) was rewritten from zero.
  - the HTMLMaindetailsParser class (for persons) was rewritten from zero.
  - unified the "episode list" and "episodes cast" parsers.
  - fixed a bug parsing locations, which resulted in missing information.
  - locations_parser splitted from "tech" parser.
  - "connections" parser now handles the recently introduced notes.

  [http parser conversion]

  - these parsers worked out-of-the-box; airing, eprating, alternateversions, dvd, goofs, keywords, movie_awards, movie_faqs, person_awards, rec, releasedates, search_movie, search_person, soundclips, soundtrack, trivia, videoclips.
  - these parsers were fixed; amazonrev, connections, episodes, crazycredits, externalrev, misclinks, newsgrouprev, news, officialsites, otherworks, photosites, plot, quotes, ratings, sales, taglines, tech, business, literature, publicity, trivia, videoclips, maindetails, movie.

  [mobile]

  - fixed to work with the new design.
  - a lot of code is now shared amongst 'http' and 'mobile'.

  [sql]

  - fixes for other bugs related to unicode support.
  - minor changes to slightly improve performances.

- What's new in release 2.9 "Rodan! The Flying Monster" (21 Feb 2007)

  [global]

  - on 19 February IMDb has redesigned its site; this is the last IMDbPY's release to parse the "old layout" pages; from now on, the development will be geared to support the new web pages. See the README.redesign file for more information.
  - minor clean-ups and functions added to the helpers module.

  [http]

  - fixed some unicode-related problems searching for movie titles and person names; also changed the queries used to search titles/names.
  - fixed a bug parsing episodes for tv series.
  - fixed a bug retrieving movieID for tv series, searching for titles.

  [mobile]

  - fixed a problem searching exact matches (movie titles only).
  - fixed a bug with cast entries, after minor changes to the IMDb's web site HTML.

  [local and sql]

  - fixed a bug parsing birth/death dates and notes.

  [sql]

  - (maybe) fixed another unicode-related bug fetching data from a MySQL database. Maybe. Maybe. Maybe.

- What's new in release 2.8 "Apollo 13" (14 Dec 2006)

  [general]

  - fix for environments where sys.stdin was overridden by a custom object.

  [http data access system]

  - added support for the movies' "FAQ" page.
  - now the "full credits" (aka "full cast and crew") page can be parsed; it's mostly useful for tv series, because this page is complete while "combined details" contains only partial data. E.g.

    ia.update(tvSeries, 'full credits')

  - added support for the movies' "on television" (ia.update(movie, "airing"))
  - fixed a bug with 'miscellaneous companies'.
  - fixed a bug retrieving the list of episodes for tv series.
  - fixed a bug with tv series episodes' cast.
  - generic fix for XML single tags (unvalid HTML tags) like
  - fixed a minor bug with 'original air date'.

  [sql data access system]

  - fix for a unicode bug with recent versions of SQLObject and MySQL.
  - fix for a nasty bug in imdbpy2sql.py that will show up splitting a data set too large to be sent in a single shot to the database.

  [mobile data access system]

  - fixed a bug searching titles and names, where XML char references were not converted.

- What's new in release 2.7 "Pitch Black" (26 Sep 2006)

  [general]

  - fixed search_movie.py and search_person.py scripts; now they return both the movieID/personID and the imdbID.
  - the IMDbPY account was configured to hide the mini-headshots.
  - http and mobile data access systems now try to handle queries with too many results.

  [http data access system]

  - fixed a minor bug retrieving information about persons, with movies in production.
  - fixed support for cast list of tv series.
  - fixed a bug retrieving 'plot keywords'.
  - some left out company credits are now properly handled.

  [mobile data access system]

  - fixed a major bug with the cast list, after the changes to the IMDb web site.
  - fixed support for cast list of tv series.
  - fixed a minor bug retrieving information about persons, with movies in production.
  - now every AKA title is correctly parsed.

  [sql data access system]

  - fixed a(nother) bug updating imdbID for movies and persons.
  - fixed a bug retrieving personID, while handling names references.

  [local data access system]

  - "where now" information now correctly handles multiple lines (also affecting the imdbpy2sql.py script).

- What's new in release 2.6 "They Live" (04 Jul 2006)

  [general]

  - renamed sortMovies to cmpMovies and sortPeople to cmpPeople; these function are now used to compare Movie/Person objects. The cmpMovies also handles tv series episodes.

  [http data access system]

  - now information about "episodes rating" are retrieved.
  - fixed a bug retrieving runtimes and akas information.
  - fixed an obscure bug trying an Exact Primary Title/Name search when the provided title was wrong/incomplete.
  - support for the new format of the "DVD details" page.

  [sql data access system]

  - now at insert-time the tables doesn't have indexes, which are added later, resulting in a huge improvement of the performances of the imdbpy2sql.py script.
  - searching for tv series episodes now works.
  - fixed a bug inserting information about top250 and bottom10 films rank.
  - fixed a bug sorting movies in people's filmography.
  - fixed a bug filtering out adult-only movies.
  - removed unused ForeignKeys in the dbschema module.

  - fixed a bug inserting data in databases that require a commit() call, after a call to executemany().
  - fixed a bug inserting aka titles in database that checks for foreign keys consistency.
  - fixed an obscure bug splitting too huge data sets.
  - MoviesCache and PersonsCache are now flushed few times.
  - fixed a bug handling excessive recursion.
  - improved the exceptions handling.

- What's new in release 2.5 "Ninja Thunderbolt" (15 May 2006)

  [general]

  - support for tv series episodes; see the README.series file.
  - modified the DISCLAIMER.txt file to be compliant to the debian guidelines.
  - fixed a bug in the get_first_movie.py script.
  - Movie and Person instances are now hashable, so that they can be used as dictionary keys.
  - modified functions analyze_title and build_title to support tv episodes.
  - use isinstance for type checking.
  - minor updates to the documentation.
  - the imdbID for Movie and Person instances is now searched if either one of movieID/personID and title/name is provided.
  - introduced the isSame() method for both Movie and Person classes, useful to compare object by movieID/personID and accessSystem.
  - __contains__() methods are now recursive.
  - two new functions in the IMDbBase class, title2imdbID() and name2imdbID() are used to get the imdbID, given a movie title or person name.
  - two new functions in the helpers module, sortedSeasons() and sortedEpisodes(), useful to manage lists/dictionaries of tv series episodes.
  - in the helpers module, the get_byURL() function can be used to retrieve a Movie or Person object for the given URL.
  - renamed the "ratober" C module to "cutils".
  - added CONTRIBUTORS.txt file.

  [http data access system]

  - fixed a bug regarding currentRole for tv series.
  - fixed a bug about the "merchandising links" page.

  [http and mobile data access systems]

  - fixed a bug retrieving cover url for tv (mini) series.

  [mobile data access system]

  - fixed a bug with tv series titles.
  - retrieves the number of episodes for tv series.

  [local data access system]

  - new get_episodes function in the cutils/ratober C module.

  - search functions (both C and pure python) are now a lot faster.
  - updated the documentation with work-arounds to make the mkdb program works with a recent set of plain text data files.

  [sql data access system]

  - uses the SQLObject ORM to support a wide range of database engines.
  - added in the cutils C module the soundex() function, and a fall back Python only version in the parser.sql package.

- What's new in release 2.4 "Munich" (09 Feb 2006)

  [general]

  - strings are now unicode/utf8.
  - unified Movie and Person classes.
  - the strings used to store every kind of information about movies and person now are modified (substituting titles and names references) only when it's really needed.
  - speed improvements in functions modifyStrings, sortMovies, canonicalName, analyze_name, analyze_title.
  - performance improvements in every data access system.
  - removed the deepcopy of the data, updating Movie and Person information.
  - moved the "ratober" C module in the imdb.parser.common package, being used by both ""http" and "sql" data access systems.
  - C functions in the "ratober" module are always case insensitive.
  - the setup.py script contains a work-around to make installation go on even if the "ratober" C module can't be compiled (displaying a warning), since it's now optional.
  - minor updates to documentation, to keep it in sync with changes in the code.
  - the new helpers.py module contains functions useful to write IMDbPY-based programs.
  - new doc file README.utf8, about unicode support.

  [http data access system]

  - the ParserBase class now inherits from sgmllib.SGMLParser, instead of htmllib.HTMLParser, resulting in a little improvement in parsing speed.
  - fixed a bug in the parser for the "news" page for movies and persons.
  - removed special handlers for entity and chardefs in the HTMLMovieParser class.
  - fixed bugs related to non-ascii chars.
  - fixed a bug retrieving the URL of the cover.
  - fixed a nasty bug retrieving the title field.
  - retrieve the 'merchandising links' page.
  - support for the new "episodes cast" page for tv series.
  - fixed a horrible bug retrieving guests information for tv series.

  [sql data access system]

  - fixed the imdbpy2sql.py script, to handle files with spurious lines.

  - searches for names and titles are now much faster, if the imdb.parser.common.ratober C module is compiled and installed.
  - imdbpy2sql.py now works also on partial data (i.e. if you've not downloaded every single plain text file).
  - imdbpy2sql.py considers also a couple of files in the contrib directory.
  - searching names and titles, only the first 5 chars returned from the SOUNDEX() SQL function are compared.
  - should works if the database is set to unicode/utf-8.

  [mobile data access system]

  - fixed bugs related to non-ascii chars.
  - fixed a bug retrieving the URL of the cover.
  - retrieve currentRole/notes also for tv guest appearances.

  [local data access system]

  - it can work even if the "ratober" C module is not compiled; obviously the pure python substitute is painfully slow (a warning is issued).

- What's new in release 2.3 "Big Fish" (03 Dec 2005)

  [general]

  - uniformed numerous keys for Movie and Person objects.
  - 'birth name' is now always in canonical form, and 'nick names' are always normalized; these changes also affect the sql data access system.

  [http data access system]

  - removed the 'imdb mini-biography by' key; the name of the author is now prepended to the 'mini biography' key.
  - fixed an obscure bug using more than one access system (http in conjunction with mobile or httpThin).
  - fixed a bug in amazon reviews.

  [mobile data access system]

  - corrected some bugs retrieving filmography and cast list.

  [sql data access system]

  - remove 'birth name' and 'nick names' from the list of 'akas'.
  - in the SQL database, 'crewmembers' is now 'miscellaneous crew'.
  - fixed a bug retrieving "guests" for TV Series.

- What's new in release 2.2 "The Thing" (17 Oct 2005)

  [general]

  - now the Person class has a 'billingPos' instance variable used to keep record of the position of the person in the list of credits (as an example, "Laurence Fishburne" is billed in 2nd position in the cast list for the "Matrix, The (1999)" movie.
  - added two functions to the utils module, to sort respectively movies (by year/title/imdbIndex) and persons (by billingPos/name/imdbIndex).
  - every data access system support the 'adultSearch' argument and the do_adult_search() method to exclude the adult movies from your searches. By default, adult movies are always listed.

- renamed the scripts, appending the ".py" extension.
- added an "IMDbPY Powered" logo and a bitmap used by the Windows installer.
- now Person and Movie objects always convert name/title to the canonical format (Title, The).
- minor changes to the functions used to convert to "canonical format" names and titles; they should be faster and with better matches.
- 'title' is the first argument, instancing a Movie object (instead of 'movieID').
- 'name' is the first argument, instancing a Movie object (instead of 'personID').

[http data access system]

- retrieves the 'guest appearances' page for TV series.
- fixed a bug retrieving newsgroup reviews urls.
- fixed a bug managing non-breaking spaces (they're truly a damnation!)
- fixed a bug with mini TV Series in people's biographies.
- now keywords are in format 'bullet-time' and no more 'Bullet Time'.

[mobile data access system]

- fixed a bug with direct hits, searching for a person's name.
- fixed a bug with languages and countries.

[local data access system]

- now cast entries are correctly sorted.
- new search system; it should return better matches in less time (searching people's name is still somewhat slow); it's also possibile to search for "long imdb canonical title/name".
- fixed a bug retrieving information about a movie with the same person listed more than one time in a given role/duty (e.g., the same director for different episodes of a TV series). Now it works fine and it should also be a bit faster.
- 'notable tv guest appearences' in biography is now a list of Movie objects.
- writers are sorted in the right order.

[sql data access system]

- search results are now sorted in correct order; difflib is used to calculate strings similarity.
- new search SQL query and comparison algorithm; it should return much better matches.
- searches for only a surname now returns much better results.
- fixed a bug in the imdbpy2sql.py script; now movie quotes are correctly managed.
- added another role, 'guests', for notable tv guest appearences.
- writers are sorted in the right order.
- put also the 'birth name' and the 'nick names' in the akanames table.

- What's new in release 2.1 "Madagascar" (30 Aug 2005)

  [general]

  - introduced the "sql data access system"; now you can transfer the whole content of the plain text data files (distributed by IMDb) into a SQL database (MySQL, so far).
  - written a tool to insert the plain text data files in a SQL database.

  - fixed a bug in items() and values() methods of Movie and Person classes.
  - unified portions of code shared between “local” and “sql”.

  [http data access system]

  - fixed a bug in the search_movie() and search_person() methods.
  - parse the “external reviews”, “newsgroup reviews”, “newsgroup reviews”, “misc links”, “sound clips”, “video clips”, “amazon reviews”, “news” and “photo sites” pages for movies.
  - parse the “news” page for persons.
  - fixed a bug retrieving personID and movieID within namesRefs and titlesRefs.

  [local data access system]

  - fixed a bug; ‘producer’ data where scanned two times.
  - some tags were missing for the laserdisc entries.

  [mobile data access system]

  - fixed a bug retrieving cast information (sometimes introduced with “Cast overview” and sometimes with “Credited cast”).
  - fixed a bug in the search_movie() and search_person() methods.

- What’s new in release 2.0 “Land Of The Dead” (16 Jul 2005)

  [general]

  - WARNING! Now, using http and mobile access methods, movie/person searches will include by default adult movie titles/pornstar names. You can still deactivate this feature by setting the adultSearch argument to false, or calling the do_adult_search() method with a false value.
  - fixed a bug using the ‘all’ keyword of the ‘update’ method.

  [http data access system]

  - added the “recommendations” page.
  - the ‘notes’ instance variable is now correctly used to store miscellaneous information about people in non-cast roles, replacing the ‘currentRole’ variable.
  - the adultSearch initialization argument is by default true.
  - you can supply the proxy to use with the ‘proxy’ initialization argument.
  - retrieve the “plot outline” information.
  - fixed a bug in the BasicMovieParser class, due to changes in the IMDb’s html.
  - the “rating details” parse information about the total number of voters, arithmetic mean, median and so on. The values are stored as integers and floats, and no more as strings.
  - dictionary keys in soundtrack are lowercase.
  - fixed a bug with empty ‘location’ information.

  [mobile data access system]

  - number of votes, rating and top 250 rank are now integers/floats.
  - retrieve the “plot outline” information.

  [local data access system]

  - number of votes, rating and top 250 rank are now integers/floats.

- What's new in release 1.9 "Ed Wood" (02 May 2005)

  [general]

  - introduced the new "mobile" data access system, useful for small systems. It should be from 2 to 20 times faster than "http" or "httpThin".
  - the "http", "httpThin" and "mobile" data access system can now search for adult movies. See the README.adult file.
  - now it should works again with python 2.0 and 2.1.
  - fixed a bug affecting performances/download time.
  - unified some keywords amongst differents data access systems.

  [http data access system]

  - fixed some bugs; now it retrieves names akas correctly.

- What's new in release 1.8 "Paths Of Glory" (24 Mar 2005)

  [general]

  - introduced a new data access system "httpThin", useful for systems with limited bandwidth and CPU power, like PDA, hand-held devices and mobile phones.
  - the setup.py script can be configured to not compile/install the local access system and the example scripts (useful for hand-held devices); introduced setup.cfg and MANIFEST.in files.
  - updated the list of articles used to manage movie titles.
  - removed the all_info tuples from Movie and Person classes, since the list of available info sets depends on the access system. I've added two methods to the IMDbBase class, get_movie_infoset() and get_person_infoset().
  - removed the IMDbNotAvailable exception.
  - unified some code in methods get_movie(), get_person() and update() in IMDbBase class.
  - minor updates to the documentation; added a 46x46 PNG icon.
  - documentation for small/mobile systems.

  [Movie class]

  - renamed the m['notes'] item of Movie objects to m['episodes'].

  [Person class]

  - the p.__contains__(m) method can be used to check if the p Person has worked in the m Movie.

  [local data access system]

  - gather information about "laserdisc", "literature" and "business".
  - fixed a bug in ratober.c; now the search_name() function handles search strings already in the "Surname, Name" format.
  - two new methods, get_lastMovieID() and get_lastPersonID().

  [http data access system]

  - limit the number of results for the query; this will save a lot of bandwidth.
  - fixed a bug retrieving the number of episodes of tv series.
  - now it retrieves movies information about "technical specifications", "business data", "literature", "soundtrack", "dvd" and "locations".

  - retrieves people information about “publicity” and “agent”.

- What’s new in release 1.7 “Saw” (04 Feb 2005)

  [general]

  - Person class has two new keys; ‘canonical name’ and ‘long imdb canonical name’, like “Gibson, Mel” and “Gibson, Mel (I)”.
  - now titles and names are always internally stored in the canonical format.
  - search_movie() and search_person() methods return the “read” movieID or personID (handling aliases).
  - Movie and Person objects have a ‘notes’ instance attribute, used to specify comments about the role of a person in a movie. The Movie class can also contain a [‘notes’] item, used to store information about the runtime; e.g. (26 episodes).
  - fixed minor bugs in the IMDbBase, Person and Movie classes.
  - some performance improvements.

  [http data access system]

  - fixed bugs retrieving the currentRole.
  - try to handle unicode chars; return unicode strings when required.
  - now the searches return also “popular titles” and “popular names” from the new IMDb’s search system.

  [local data access system]

  - information about movie connections are retrieved.
  - support for multiple biographies.
  - now it works with Python 2.2 or previous versions.
  - fixed a minor glitch in the initialization of the ratober C module.
  - fixed a pair buffer overflows.
  - fixed some (very rare) infinite loops bugs.
  - it raises IMDbDataAccessError for (most of) I/O errors.

  [Movie class] - fixed a bug getting the “long imdb canonical title”.

- What’s new in release 1.6 “Ninja Commandments” (04 Jan 2005)

  [general]

  - now inside Movie and Person object, the text strings (biography, movie plot, etc.) contain titles and names references, like “_Movie, The (1999)_ (qv)” or “‘A Person’ (qv)”; these reference are transformed at access time with a user defined function.
  - introduced _get_real_movieID and _get_real_personID methods in the IMDbBase class, to handle title/name aliases for the local access system.
  - split the _normalize_id method in _normalize_movieID and _normalize_personID.
  - fixed some bugs.

  [Movie class]

  - now you can access the ‘canonical title’ and ‘long imdb canonical title’ attributes, to get the movie title in the format “Movie Title, The”.

  [local data access system]

  - title and name aliases now work correctly.
  - now get_imdbMovieID and get_imdbPersonID methods should work in almost every case.
  - people's akas are handled.

  [http data access system]

  - now the BasicMovieParser class can correctly gather the imdbID.

- What's new in release 1.5 "The Incredibles" (23 Dec 2004)

  [local database]

  - support a local installation of the IMDb database! WOW! Now you can download the plain text data files from http://imdb.com/interfaces.html and access those information through IMDbPY!

  [general]

  - movie titles and person names are "fully normalized"; Not "Matrix, The (1999)", but "The Matrix (1999)"; Not "Cruise, Tom" but "Tom Cruise".
  - get_mop_infoSet() methods can now return a tuple with the dictionary data and a list of information sets they provided.

  [http data access system]

  - support for the new search system (yes, another one. . .)
  - a lot of small fixes to stay up-to-date with the html of the IMDb web server.
  - modified the personParser module so that it will no more download both "filmoyear" and "maindetails" pages; now only the latter is parsed.
  - movie search now correctly reports the movie year and index.
  - gather "locations" information about a movie.
  - modified the HTMLAwardsParser class so that it doesn't list empty entries.

- What's new in release 1.4 "The Village" (10 Nov 2004)

  [http data access system]

  - modified the personParser.HTMLMaindetailsParser class, because IMDb has changed the img tag for the headshot.
  - now 'archive footage' is handled correctly.

  [IMDb class]

  - fixed minor glitches (missing "self" parameter in a couple of methods).

  [misc]

  - now distutils installs also the example scripts in ./bin/*

- What's new in release 1.3 "House of 1000 Corpses" (6 Jul 2004)

  [http data access system]

  - modified the BasicMovieParser and BasicPersonParser classes, because IMDb has removed the "page-flicker" from the html pages.

  [general]

  - the test suite was moved outside the tgz package.

- What's new in release 1.2 "Kill Bill" (2 May 2004)

  [general]

  - now it retrieves almost every available information about movie and people!
  - introduced the concept of "data set", to retrieve different sets of information about a movie/person (so that it's possibile to fetch only the needed information).
  - introduced a test suite, using the PyUnit (unittest) module.
  - fixed a nasty typo; the analyze_title and build_title functions now use the strings 'tv mini series' and 'tv series' for the 'kind' key (previously the 'serie' word ws used).
  - new design; removed the mix-in class and used a factory pattern; imdb.IMDb is now a function, which returns an instance of a class, subclass of imdb.IMDbBase.
  - introduced the build_name(name_dict) function in the utils module, which takes a dictionary and build a long imdb name.
  - fixed bugs in the analyze_name function; now it correctly raise an IMDbParserError exception for empty/all spaces strings.
  - now the analyze_title function sets only the meaningful information (i.e.: no 'kind' or 'year' key, if they're not set)

  [http data access system]

  - removed all non-greedy regular expressions.
  - removed all regular expressions in the movieParser module; now self.rawdata is no more used to search "strange" matches.
  - introduced a ParserBase class, used as base class for the parsers.
  - retrieve information about the production status (pre-production, announced, in production, etc.)
  - mpaa is now a string.
  - now when an IMDbDataAccessError is raised it shows also the used proxy.
  - minor changes to improve performances in the handle_data method of the HTMLMovieParser class.
  - minor changes to achieve a major performances improvement in the BasicPersonParser class in the search-PersonParse module.

  [Movie class]

  - fixed a bug in isSameTitle method, now the accessSystem is correctly checked.
  - fixed some typos.

  [Person class]

  - minor changes to the isSamePerson method (now it uses the build_name function).

- What's new in release 1.1 "Gigli" (17 Apr 2004)

  [general]

  - added support for persons (search & retrieve information about people).
  - removed the dataSets module.
  - removed the MovieTitle and the SearchMovieResults classes; now information about the title is stored directly in the Movie object and the search methods return simple lists (of Movie or Person objects).
  - removed the IMDbTitleError exception.

  - added the analyze_name() function in the imdb.utils module, which returns a dictionary with the 'name' and 'imdbIndex' keys from the given long imdb name string.

  [http data access system]

  - http search uses the new search system.
  - moved the plotParser module content inside the movieParser module.
  - fixed a minor bug handling AKAs for movie titles.

  [IMDb class]

  - introduced the update(obj) method of the IMDb class, to update the information of the given object (a Movie or Person instance).
  - added the get_imdbURL(obj) method if the IMDb class, which returns the URL of the main IMDb page for the given object (a Movie or Person).
  - renamed the 'kind' parameter of the IMDb class to 'accessSystem'.

  [Movie class]

  - now __str__() returns only the short name; the summary() method returns a pretty-printed string for the Movie object.
  - persons are no more simple strings, but Person objects (the role/duty is stored in the currentRole variable of the object).
  - isSameTitle(obj) method to compare two Movie objects even when not all information are gathered.
  - new __contains__() method, to check is a given person was in a movie.

  [misc]

  - updated the documentation.
  - corrected some syntax/grammar errors.

- What's new in release 1.0 "Equilibrium" (01 Apr 2004)

  [general]

  - first public release.
  - retrieve data only from the web server.
  - search only for movie titles.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search