okcupyd

Search docs

Getting Started
API Objects
Core Objects
Project Modules

Read the Docs v: stable

okcupyd

Getting Started

Installation/Setup

pip/PyPI

okcupyd is available for install from PyPI. If you have pip you can simply run:

```
pip install okcupyd
```

to make okcupyd available for import in python.

From Source

You can install from source by running the setup.py script included as part of this repository as follows:

```
python setup.py install
```

This can be useful if you want to install a version that has not yet been released on PyPI.

From Docker

okcupyd is available on docker (see https://registry.hub.docker.com/u/imalison/okcupyd/)

If you have docker installed on your machine, you can run

```
docker run -t -i imalison/okcupyd okcupyd
```

to get an interactive okcupyd shell.

Use

Interactive

Installing the okcupyd package should add an executable script to a directory in your $PATH that will allow you to type *okcupyd* into your shell of choice to enter an interactive ipython shell that has been prepared for use with okcupyd. Before the shell starts, you will be prompted for your username and password. This executable script accepts the flags *–enable-logger* which enables a logger of the given name, and *–credentials* whose action is described below.

It is highly recommended that you use the *–enable-logger=requests* and *–enable-logger=okcupyd* flags if you encounter any problems.

Credentials

If you wish to avoid entering your password each time you start a new session you can do one of the following things:

1. Create a python module (.py file) with your username and password set to the variables USERNAME and PASSWORD respectively. You can start an interactive session with the USERNAME and PASSWORD stored in *my_credentials.py* by running

```
PYTHONPATH=. okcupyd --credentials my_credentials
```

from the directory that *my_credentials.py* is stored in

The *PYTHONPATH=.* at the front of this command is necessary to ensure that the current directory is searched for modules.

If you wish to use a version of this library that you have cloned but not installed, you can use the tox environment *venv* to do the same thing with such a version of the code:

```
PYTHONPATH=. tox -e venv -- okcupyd --credentials my_credentials
```

2. Set the shell environment variables *OKC_USERNAME* and *OKC_PASSWORD* to your username and password respectively. Make sure to export the variables so they are visible in

processes started from the shell. You can make a credentials.sh file to do this using the following template:

```
export OKC_USERNAME='your_username'
export OKC_PASSWORD='your_password'
```

Simply run source credentials.sh to set the environment variables and your shell should be properly configured. Note that this approach requires that the relevant environment variables be set before okcupyd.settings is imported.

3. Manually override the values in okcupyd/settings.py. This method is not recommended because it requires you to find the installation location of the package. Also, If you are working with a source controlled version, you could accidentally commit your credentials.

Using `--credentials` in a custom script

The ~okcupyd.util.misc.add_command_line_options and ~okcupyd.util.misc.handle_command_line_options can be used to make a custom script support the `--credentials` and `--enable-loggers` command line flags. The interface to these functions is admittedly a little bit strange. Refer to the example below for details concerning how to use them:

```
import argparse
parser = argparse.ArgumentParser()
util.add_command_line_options(parser.add_argument)
args = parser.parse_args()
util.handle_command_line_options(args)
```

Basic Examples

All examples in this section assume that the variable u has been initialized as follows:

```
import okcupyd
user = okcupyd.User()
```

Searching profiles

To search through the user:

```
profiles = user.search(age_min=26, age_max=32)
for profile in profiles[:10]:
```

```
    profile.message("Pumpkins are just okay.")
```

To search for users that have answered a particular question in a way that is consistent with the user's preferences for that question:

```
user_question = user.questions.very_important[0]
profiles = user.search(question=user_question)
for profile in profiles[:10]:
    their_question = profile.find_question(user_question.id)
    profile.message("I'm really glad that you answered {0} to {1}".format(
        their_question.their_answer, their_question.question.text
    ))
```

The search functionality can be accessed without a `User` instance:

```
from okcupyd.search import SearchFetchable

for profile in SearchFetchable(attractiveness_min=8000)[:5]:
    profile.message("hawt...")
```

This is particularly useful if you want to explicitly provide the session that should be used to search:

```
from okcupyd.session import Session
from okcupyd.search import SearchFetchable

session = Session.login('username', 'password')
for profile in SearchFetchable(session=session, attractiveness_min=8000)[:5]:
    profile.message("hawt...")
```

For more details about what filter arguments can be used with these search functions, see the doucmentation for `SearchFetchable()`

Messaging another user

```
user.message('foxylady899', 'Do you have a map?')
# This has slightly different semantics; it will not look through the user's
# inbox for an existing thread.
user.get_profile('foxylady889').message('Do you have a map?')
```

Rating a profile

```
user.get_profile('foxylady899').rate(5)
```

Mailbox

```
first_thread = user.inbox[0]
print(first_thread.messages)
```

Quickmatch, Essays, Looking For, Details

You can access the essays, looking for attributes and detail attributes of a profile very easily

```
profile = user.quickmatch()
print(profile.essays.self_summary)
print(profile.looking_for.ages)
print(profile.details.orientation)
```

The data for these attributes is loaded from the profile page, but it should be noted that this page is only loaded on demand, so the first of these attribute access calls will make an http request.

A logged in user can update their own details using these objects:

```
user.profile.essays.self_summary = "I'm pretty boring."
user.profile.looking_for.ages = 18, 19
user.profile.details.ethnicities = ['asian', 'black', 'hispanic']
```

These assignments will result in updates to the okcupid website. When these updates happen, subsequent access to any profile attribute will result in a new http request to reload the profile page.

Fetchable

Most of the collection objects that are returned from function invocations in the okcupyd library are instances of `Fetchable`. In most cases, it is fine to treat these objects as though they are lists because they can be iterated over, sliced and accessed by index, just like lists:

```
for question in user.profile.questions:
    print(question.answer.text)
```

```
a_random_question = user.profile.questions[2]
for question in questions[2:4]:
    print(question.answer_options[0])
```

However, in some cases, it is important to be aware of the subtle differences between `Fetchable` objects and python lists. `Fetchable` construct the elements that they "contain" lazily. In most of its uses in the okcupyd library, this means that http requests can be made to populate `Fetchable` instances as its elments are requested.

The `questions` `Fetchable` that is used in the example above fetches the pages that are used to construct its contents in batches of 10 questions. This means that the actual call to retrieve data is made when iteration starts. If you enable the request logger when you run this code snippet, you get output that illustrates this fact:

```
2014-10-29 04:25:04 Livien-MacbookAir requests.packages.urllib3.connectionpool[82461] DEBUG
"GET /profile/ShrewdDrew/questions?leanmode=1&low=11 HTTP/1.1" 200 None
 Yes
 Yes
 Kiss someone.
 Yes.
 Yes
 Sex.
 Both equally
 No, I wouldn't give it as a gift.
 Maybe, I want to know all the important stuff.
 Once or twice a week
 2014-10-29 04:25:04 Livien-MacbookAir requests.packages.urllib3.connectionpool[82461] DEBUG
"GET /profile/ShrewdDrew/questions?leanmode=1&low=21 HTTP/1.1" 200 None
 No.
 No
 No
 Yes
 Rarely / never
 Always.
 Discovering your shared interests
 The sun
 Acceptable.
 No.
```

Some fetchables will continue fetching content for quite a long time. The search fetchable, for example, will fetch content until okcupid runs out of search results. As such, things like:

```
for profile in user.search():
    profile.message("hey!")
```

should be avoided, as they are likely to generate a massive number of requests to okcupid.com.

Another subtlety of the `Fetchable` class is that its instances cache its contained results. This means that the second iteration over `okcupyd.profile.Profile.questions` in the example below does not result in any http requests:

```
for question in user.profile.questions:
    print(question.text)

for question in user.profile.questions:
    print(question.answer)
```

It is important to understand that this means that the contents of a `Fetchable` are not guarenteed to be in sync with okcupid.com the second time they are requested. Calling `refresh()` will cause the `Fetchable` to request new data from okcupid.com when its contents are requested. The code snippet that follows prints out all the questions that the logged in user has answered roughly once per hour, including ones that are answered while the program is running.

```
import time

while True:
    for question in user.profile.questions:
        print(question.text)
    user.profile.questions.refresh()
    time.sleep(3600)
```

Without the call to *user.profile.questions.refresh()*, this program would never update the user.profile.questions instance, and thus what would be printed to the screen with each iteration of the for loop.

Development

tox

If you wish to contribute to this project, it is recommended that you use tox to run tests and enter the interactive environment. You can get tox by running

```
pip install tox
```

if you do not already have it.

Once you have cloned the project and installed tox, run:

```
tox -e py27
```

This will create a virtualenv that has all dependencies as well as the useful ipython and ipdb libraries installed, and run all okcupyds test suite.

If you want to run a command with access to a virtualenv that was created by tox you can run

```
tox -e venv -- your_command
```

To use the development version of the interactive shell (and avoid any conflicts with versions installed in site-packages) you would run the following command:

```
tox -e venv -- okcupyd
```

git hooks

If you plan on editing this file (getting_started.rst) you must install the provided git hooks that are included in this repository by running:

```
bin/create-githook-symlinks.sh
```

from the root directory of the repository.

API Objects

The classes and functions documented on this page constitute the public API to the okcupyd library.

`User`

`User` serves as the primary entry point to the okcupyd library. Most of the objects mentioned on this page are accessible in some way or another from instances of `User`.

class **okcupyd.user.User**(*session=None*) [source]

Encapsulate a logged in okcupid user.

classmethod **from_credentials**(*username*, *password*) [source]

Parameters:
- **username** (*str*) – The username to log in with.
- **password** (*str*) – The password to log in with.

__init__(*session=None*) [source]

Parameters: **session** (`Session`) – The session which will be used for interacting with okcupid.com If none is provided, one will be instantiated automatically with the credentials in `settings`

profile = *None*

A `Profile` object belonging to the logged in user.

inbox = *None*

A `Fetchable` of `MessageThread` objects corresponding to messages that are currently in the user's inbox.

outbox = *None*

A `Fetchable` of `MessageThread` objects corresponding to messages that are currently in the user's outbox.

drafts = *None*

A `Fetchable` of `MessageThread` objects corresponding to messages that are currently in the user's drafts folder.

visitors = *None*

A `Fetchable` of `Profile` objects of okcupid.com users that have visited the user's profile.

questions = *None*

A `Questions` object that is instantiated with the owning `User` instance's session.

attractiveness_finder = *None*

An `_AttractivenessFinder` object that is instantiated with the owning `User` instance's session.

photo= *None*

A `PhotoUploader` that is instantiated with the owning `User` instance's session.

get_profile(*username*) [source]

Get the `Profile` associated with the supplied username.

Parameters: **username** – The username of the profile to retrieve.

username [source]

Return the username associated with the `User`.

message(*username*, *message_text*) [source]

Message an okcupid user. If an existing conversation between the logged in user and the target user can be found, reply to that thread instead of starting a new one.

Parameters:
- **username** (*str*) – The username of the user to which the message should be sent.
- **message_text** (*str*) – The body of the message.

search(***kwargs*) [source]

Call `SearchFetchable()` to get a `Fetchable` object that will lazily perform okcupid searches to provide `Profile` objects matching the search criteria.

Defaults for *gender*, *gentation*, *location* and *radius* will be provided if none are given.

Parameters: **kwargs** – See the `SearchFetchable()` docstring for details about what parameters are available.

delete_threads(*thread_ids_or_threads*) [source]

Call `delete_threads()`.

Parameters: **thread_ids_or_threads** – A list whose members are either `MessageThread` instances or okc_ids of message threads.

get_user_question(*question*, *fast=False*, *bust_questions_cache=False*) [source]

Get a `UserQuestion` corresponding to the given `Question`.

HUGE CAVEATS: If the logged in user has not answered the relevant question, it will

automatically be answered with whatever the first answer to the question is.

For the sake of reducing the number of requests made when this function is called repeatedly this function does not bust the cache of this `User`'s `okcupyd.profile.Profile.questions` attribute. That means that a question that HAS been answered could still be answered by this function if this `User`'s `questions` was populated previously (This population happens automatically – See `Fetchable` for details about when and how this happens).

Parameters:
- **question** (`BaseQuestion`) – The question for which a `UserQuestion` should be retrieved.
- **fast** (*bool*) – Don't try to look through the users existing questions to see if arbitrarily answering the question can be avoided.
- **bust_questions_cache** (*bool*) – clear the `questions` attribute of this users `Profile` before looking for an existing answer. Be aware that even this does not eliminate all race conditions.

get_question_answer_id(*question*, *fast=False*, *bust_questions_cache=False*) [source]

Get the index of the answer that was given to *question*

See the documentation for `get_user_question()` for important caveats about the use of this function.

Parameters:
- **question** (`BaseQuestion`) – The question whose *answer_id* should be retrieved.
- **fast** (*bool*) – Don't try to look through the users existing questions to see if arbitrarily answering the question can be avoided.
- **bust_questions_cache** (*bool*) –

  param bust_questions_cache:

  clear the

  `questions` attribute of this users `Profile` before looking for an existing answer. Be aware that even this does not eliminate all race conditions.

quickmatch() [source]

Return a `Profile` obtained by visiting the quickmatch page.

copy(*profile_or_user*) [source]

Create a `Copy` instance with the provided object as the source and this `User` as the destination.

Parameters: **profile_or_user** – A `User` or `Profile` object.

`Profile`

class **okcupyd.profile.Profile**(*session*, *username*, ***kwargs*) [source]

Represent the profile of an okcupid user.

Many of the attributes on this object are `cached_property` instances which lazily load their values, and cache them once they have been accessed. This makes it so that this object avoids making unnecessary HTTP requests to retrieve the same piece of information twice.

Because of this caching behavior, care must be taken to invalidate cached attributes on the object if an up to date view of the profile is needed. It is recommended that you call `refresh()` to accomplish this, but it is also possible to use `bust_self()` to bust individual properties if necessary.

username = *None*

The username of the user to whom this profile belongs.

questions = *None*

A `Fetchable` of `Question` instances, each corresponding to a question that has been answered by the user to whom this profile belongs. The fetchable consists of `UserQuestion` instead when the profile belongs to the logged in user.

details = *None*

A `Details` instance belonging to the same user that this profile belongs to.

refresh(*reload=False*) [source]

Parameters: **reload** – Make the request to return a new profile tree. This will result in the caching of the profile_tree attribute. The new profile_tree will be returned.

is_logged_in_user [source]

Returns: *True* if this profile and the session it was created with belong to the same user and False otherwise.

profile_tree [source]

Returns: a `lxml.etree` created from the html of the profile page of the account associated with the username that this profile was insantiated with.

photo_infos [source]

Returns: list of `Info` instances for each photo displayed on okcupid.

looking_for [source]

Returns: A `LookingFor` instance associated with this profile.

rating [source]

Deprecated. Use `liked()` instead.

Returns: the rating that the logged in user has given this user or 0 if no rating has been given.

liked [source]

Returns: Whether or not the logged in user liked this profile

contacted [source]

Retuns: A boolean indicating whether the logged in user has contacted the owner of this profile.

responds [source]

Returns: The frequency with which the user associated with this profile responds to messages.

id [source]

Returns: The id that okcupid.com associates with this profile.

essays [source]

Returns: A `Essays` instance that is associated with this profile.

age [source]

Returns: The age of the user associated with this profile.

match_percentage [source]

Returns: The match percentage of the logged in user and the user associated with this object.

enemy_percentage [source]

Returns: The enemy percentage of the logged in user and the user associated with this object.

location [source]

Returns: The location of the user associated with this profile.

gender [source]

The gender of the user associated with this profile.

orientation [source]

The sexual orientation of the user associated with this profile.

message [source]

Message the user associated with this profile.

Parameters:
- **message** – The message to send to this user.
- **thread_id** – The id of the thread to respond to, if any.

attractiveness [source]

Returns: The average attractiveness rating given to this profile by the okcupid.com community.

toggle_like() [source]

Toggle whether or not the logged in user likes this profile.

like() [source]

Like this profile.

unlike() [source]

Unlike this profile.

rate(*rating*) [source]

Rate this profile as the user that was logged in with the session that this object was

instantiated with.

Parameters: **rating** – The rating to give this user.

`find_question`(*question_id*, *question_fetchable=None*) [source]

Parameters:
- **question_id** – The id of the question to search for
- **question_fetchable** – The question fetchable to iterate through if none is provided *self.questions* will be used.

`question_fetchable`(***kwargs*) [source]

Returns: A `Fetchable` instance that contains objects representing the answers that the user associated with this profile has given to okcupid.com match questions.

`authcode_get`(*path*, ***kwargs*) [source]

Perform an HTTP GET to okcupid.com using this profiles session where the authcode is automatically added as a query parameter.

`authcode_post`(*path*, ***kwargs*) [source]

Perform an HTTP POST to okcupid.com using this profiles session where the authcode is automatically added as a form item.

`LookingFor`

class `okcupyd.looking_for.LookingFor`(*profile*) [source]

Represent the looking for attributes belonging to an okcupid.com profile.

`gentation` [source]

The sex/orientation that the user is looking for.

`ages` [source]

The age range that the user is interested in.

`single` [source]

Whether or not the user is only interested in people that are single.

`near_me` [source]

Whether the user is only interested in people that are close to them.

kinds [source]

The kinds of relationship tha the user is looking for.

update(*ages=None*, *single=None*, *near_me=None*, *kinds=None*, *gentation=None*) [source]

Update the looking for attributes of the logged in user.

Parameters:
- **ages** (*tuple*) – The ages that the logged in user is interested in.
- **single** (*bool*) – Whether or not the user is only interested in people that are single.
- **near_me** (*bool*) – Whether or not the user is only interested in people that are near them.
- **kinds** (*list*) – What kinds of relationships the user should be updated to be interested in.
- **gentation** (*str*) – The sex/orientation of people the user is interested in.

class **Ages**

Ages(min, max)

max

Alias for field number 1

min

Alias for field number 0

Details

class **okcupyd.details.Details**(*profile*) [source]

Represent the details belonging to an okcupid.com profile.

classmethod **name_detail_pairs**() [source]

refresh() [source]

id_to_display_name_value [source]

as_dict [source]

convert_and_update(*data*) [source]

update(*data*) [source]

bodytype

The bodytype detail of an okcupid.com user's profile.

orientation

The orientation detail of an okcupid.com user's profile.

smokes

The smoking detail of an okcupid.com user's profile.

drugs

The drugs detail of an okcupid.com user's profile.

drinks

The drinking detail of an okcupid.com user's profile.

job

The job detail of an okcupid.com user's profile.

status

The status detail of an okcupid.com user's profile.

monogamy

The monogamous detail of an okcupid.com user's profile.

children

The children detail of an okcupid.com user's profile.

education

The education detail of an okcupid.com user's profile.

pets

The pets detail of an okcupid.com user's profile.

diet

The diet detail of an okcupid.com user's profile.

religion

The religion detail of an okcupid.com user's profile.

sign

The sign detail of an okcupid.com user's profile.

height

The height detail of an okcupid.com user's profile.

ethnicities [source]

The ethnicities detail of an okcupid.com user's profile.

income [source]

The income detail of an okcupid.com user's profile.

languages [source]

The languages detail of an okcupid.com user's profile.

Essays

class **okcupyd.essay.Essays**(*profile*) [source]

Interface to reading and writing essays.

self_summary

The contents of the essay labeled 'Self Summary'. Write to this attribute to change its value for the logged in user.

my_life

The contents of the essay labeled 'What I'm doing with my life'. Write to this attribute to

change its value for the logged in user.

good_at

The contents of the essay labeled ‘I’m really good at’. Write to this attribute to change its value for the logged in user.

people_first_notice

The contents of the essay labeled ‘The first thing people notice about me’. Write to this attribute to change its value for the logged in user.

favorites

The contents of the essay labeled ‘Favorite books, movies, shows, music, and food’. Write to this attribute to change its value for the logged in user.

six_things

The contents of the essay labeled ‘Six things I could never live without’. Write to this attribute to change its value for the logged in user.

think_about

The contents of the essay labeled ‘I spend a lot of time thinking about’. Write to this attribute to change its value for the logged in user.

friday_night

The contents of the essay labeled ‘On a typical friday night I am’. Write to this attribute to change its value for the logged in user.

private_admission

The contents of the essay labeled ‘The most private thing I’m willing to admit’. Write to this attribute to change its value for the logged in user.

message_me_if

The contents of the essay labeled ‘You should message me if’. Write to this attribute to change its value for the logged in user.

essay_names
= ['self_summary', 'my_life', 'good_at', 'people_first_notice', 'favorites', 'six_things', 'think_about', 'friday_night', 'private_admission', 'message_me_if']

A list of the attribute names that are used to store the text of of essays on instances of this class.

`PhotoUploader`

class **okcupyd.photo.PhotoUploader**(*session=None*, *user_id=None*, *authcode=None*) [source]

Upload photos to okcupid.com.

upload_and_confirm(*incoming*, ***kwargs*) [source]

Upload the file to okcupid and confirm, among other things, its thumbnail position.

Parameters:
- **incoming** – A filepath string, `Info` object or a file like object to upload to okcupid.com. If an info object is provided, its thumbnail positioning will be used by default.
- **caption** – The caption to add to the photo.
- **thumb_nail_left** – For thumb nail positioning.
- **thumb_nail_top** – For thumb nail positioning.
- **thumb_nail_right** – For thumb nail positioning.
- **thumb_nail_bottom** – For thumb nail positioning.

delete(*photo_id*, *album_id=0*) [source]

Delete a photo from the logged in users account.

Parameters:
- **photo_id** – The okcupid id of the photo to delete.
- **album_id** – The album from which to delete the photo.

`Info`

class **okcupyd.photo.Info**(*photo_id*, *tnl*, *tnt*, *tnr*, *tnb*) [source]

Represent a photo that appears on a okcupid.com user's profile.

thumb_nail_left= *None*

The horizontal position of the left side of this photo's thumbnail.

thumb_nail_top= *None*

The vertical position of the top side of this photo's thumbnail.

thumb_nail_right= *None*

The horizontal position of the right side of this photo's thumbnail.

thumb_nail_bottom= *None*

The vertical position of the bottom side of this photo's thumbnail.

jpg_uri [source]

Returns: A uri from which this photo can be downloaded in jpg format.

MessageThread

class **okcupyd.messaging.MessageThread**(*session*, *thread_element*) [source]

Represent a message thread between two users.

classmethod **delete_threads**(*session*, *thread_ids_or_threads*, *authcode=None*) [source]

Parameters:
- **session** – A logged in `Session`.
- **thread_ids_or_threads** – A list whose members are either `MessageThread` instances or okc_ids of message threads.
- **authcode** – Authcode to use for this request. If none is provided A request to the logged in user's messages page will be made to retrieve one.

messages= *None*

A `Fetchable` of `Message` objects.

id [source]

Returns: The id assigned to this message by okcupid.com.

correspondent_id [source]

Returns: The id assigned to the correspondent of this message.

correspondent [source]

Returns: The username of the user with whom the logged in user is conversing in this `MessageThread`.

read [source]

Returns: Whether or not the user has read all the messages in this `MessageThread`.

`initiator` [source]

Returns: A `Profile` instance belonging to the initiator of this `MessageThread`.

`respondent` [source]

Returns: A `Profile` instance belonging to the respondent of this `MessageThread`.

`correspondent_profile` [source]

Returns: The `Profile` of the user with whom the logged in user is conversing in this `MessageThread`.

`user_profile` [source]

Returns: A `Profile` belonging to the logged in user.

`got_response` [source]

Returns: Whether or not the `MessageThread`. has received a response.

`delete`() [source]

Delete this thread for the logged in user.

`Message`

class **`okcupyd.messaging.Message`**(*message_element*, *message_thread*) [source]

Represent a message sent on okcupid.com

`id` [source]

Returns: The id assigned to this message by okcupid.com.

`sender` [source]

Returns: A `Profile` instance belonging to the sender of this message.

`recipient` [source]

Returns: A `Profile` instance belonging to the recipient of this message.

`content` [source]

Returns: The text body of the message.

Questions

class `okcupyd.question.Questions`(*session*, *importances=('not_important', 'little_important', 'somewhat_important', 'very_important', 'mandatory')*, *user_id=None*) [source]

Interface to accessing and answering questions belonging to the logged in user.

`mandatory`

A `Fetchable` of `UserQuestion` instances that correspond to questions that have been answered by the logged in user and assigned the 'mandatory' importance.

`very_important`

A `Fetchable` of `UserQuestion` instances that correspond to questions that have been answered by the logged in user and assigned the 'very_important' importance.

`somewhat_important`

A `Fetchable` of `UserQuestion` instances that correspond to questions that have been answered by the logged in user and assigned the 'somewhat_important' importance.

`little_important`

A `Fetchable` of `UserQuestion` instances that correspond to questions that have been answered by the logged in user and assigned the 'little_important' importance.

`not_important`

A `Fetchable` of `UserQuestion` instances that correspond to questions that have been answered by the logged in user and assigned the 'not_important' importance.

`importance_name_to_number`
= {'not_important': 5, 'little_important': 4, 'mandatory': 0, 'very_important': 1, 'somewhat_important': 3}

Human readable importance name to integer used to represent them on okcupid.com

`respond_from_user_question`(*user_question*, *importance*) [source]

Respond to a question in exactly the way that is described by the given user_question.

Parameters:
- **user_question** (`UserQuestion`) – The user question to respond with.
- **importance** (int see `importance_name_to_number`) – The importance that should be used in responding to the question.

respond_from_question(*question*, *user_question*, *importance*) [source]

Copy the answer given in *question* to the logged in user's profile.

Parameters:
- **question** – A `Question` instance to copy.
- **user_question** – An instance of `UserQuestion` that corresponds to the same question as *question*. This is needed to retrieve the answer id from the question text answer on question.
- **importance** – The importance to assign to the response to the answered question.

respond(*question_id*, *user_response_ids*, *match_response_ids*, *importance*, *note=''*, *is_public=1*, *is_new=1*) [source]

Respond to an okcupid.com question.

Parameters:
- **question_id** – The okcupid id used to identify this question.
- **user_response_ids** – The answer id(s) to provide to this question.
- **match_response_ids** – The answer id(s) that the user considers acceptable.
- **importance** – The importance to attribute to this question. See `importance_name_to_number` for details.
- **note** – The explanation note to add to this question.
- **is_public** – Whether or not the question answer should be made public.

clear() [source]

USE WITH CAUTION. Delete the answer to every question that the logged in user has responded to.

`Question`

class **okcupyd.question.Question**(*question_element*) [source]

Represent a question answered by a user other than the logged in user.

Note: Because of the way that okcupid presents question data it is actually not very easy to get the index of the answer to a question that belongs to a user other than the logged in user. It is

possible to retrieve this value (see `okcupyd.user.User.get_question_answer_id()` and `get_answer_id_for_question()`), but it can take quite a few requests to do so. For this reason, the answer_id is NOT included as an attribute on this object, despite its inclusion in `UserQuestion`.

their_answer [source]

The answer that the user whose `Profile` this question was retrieved from provided to this `Question`.

my_answer [source]

The answer that the user whose `Session` was used to create this `Question` provided.

their_answer_matches [source]

Returns: whether or not the answer provided by the user answering the question is acceptable to the logged in user.

Return type: rbool

my_answer_matches [source]

Returns: bool indicating whether or not the answer provided by the logged in user is acceptable to the user answering the question.

their_note [source]

Returns: The note the answering user provided as explanation for their answer to this question.

my_note [source]

Returns: The note the logged in user provided as an explanation for their answer to this question.

`UserQuestion`

class **okcupyd.question.UserQuestion**(*question_element*) [source]

Represent a question answered by the logged in user.

get_answer_id_for_question(*question*) [source]

Get the answer_id corresponding to the answer given for question by looking at this `UserQuestion`'s answer_options. The given `Question` instance must have the same id as this `UserQuestion`.

That this method exists is admittedly somewhat weird. Unfortunately, it seems to be the only way to retrieve this information.

answer_options [source]

Returns: A list of `AnswerOption` instances representing the available answers to this question.

explanation [source]

Returns: The explanation written by the logged in user for this question (if any).

answer [source]

Returns: A `AnswerOption` instance corresponding to the answer the user gave to this question.

AnswerOption

class **okcupyd.question.AnswerOption**(*option_element*) [source]

is_users [source]

Returns: Whether or not this was the answer selected by the logged in user.

is_match [source]

Returns: Whether or not this was the answer is acceptable to the logged in user.

text [source]

Returns: The text of this answer.

id [source]

Returns: The integer index associated with this answer.

SearchFetchable()

okcupyd.search.SearchFetchable(*session=None*, ***kwargs*) [source]

Search okcupid.com with the given parameters. Parameters are registered to this function through `register_filter_builder()` of `search_filters`.

Returns: A `Fetchable` of `Profile` instances.

Parameters:

- **session** (`Session`) – A logged in session.
- **location** – A location string which will be used to filter results.
- **gender** – The gender of the user performing the search.
- **keywords** – A list or space delimeted string of words to search for.
- **order_by** – The criteria to use for ordering results. expected_values: ‘match’, ‘online’, ‘special_blend’
- **age_max** (*int*) – The maximum age of returned search results.
- **age_min** (*int*) – The minimum age of returned search results.
- **attractiveness_max** (*int*) – The maximum attractiveness of returned search results.
- **attractiveness_min** (*int*) – The minimum attractiveness of returned search results.
- **bodytype** (*str*) – expected values: ‘jacked’, ‘rather not say’, ‘fit’, ‘athletic’, ‘used up’, ‘average’, ‘full figured’, ‘overweight’, ‘curvy’, ‘thin’, ‘a little extra’, ‘skinny’
- **cats** (*str*) – expected values: ‘likes cats’, ‘dislikes cats’, ‘has cats’
- **diet** (*str*) – expected values: ‘anything’, ‘vegetarian’, ‘vegan’, ‘kosher’, ‘other’, ‘halal’
- **dogs** (*str*) – expected values: ‘dislikes dogs’, ‘has dogs’, ‘likes dogs’
- **drinks** (*str*) – expected values: ‘desperately’, ‘often’, ‘socially’, ‘very often’, ‘not at all’, ‘rarely’
- **drugs** (*str*) – expected values: ‘never’, ‘often’, ‘sometimes’
- **education_level** (*str*) – expected values: ‘law school’, ‘two[-]year college’, ‘university’, ‘space camp’, ‘high ?school’, ‘college’, ‘ph.d program’, ‘med school’, ‘masters program’
- **ethnicities** (*str*) – expected values: ‘latin’, ‘pacific islander’, ‘middle eastern’, ‘hispanic’, ‘indian’, ‘black’, ‘asian’, ‘white’, ‘other’, ‘native american’
- **gentation** (*str*) – The gentation of returned search results. expected values: ‘men and women who like bi women’, ‘gay girls only’, ‘bi guys only’, ‘gay guys only’, ‘’, ‘straight girls only’, ‘women who like men’, ‘bi men and women’, ‘guys who like girls’, ‘straight guys only’, ‘bi girls only’, ‘men and women who like bi men’, ‘guys and girls who like bi guys’, ‘both who like bi women’, ‘gay women only’, ‘gay men only’, ‘women’, ‘everybody’, ‘straight men only’, ‘girls who like girls’, ‘bi men only’, ‘both who like bi men’, ‘bi women only’, ‘guys who like guys’, ‘bi guys and girls’, ‘guys and girls who like bi girls’, ‘both who like bi guys’, ‘men who like women’, ‘girls who like guys’, ‘women who like women’, ‘both who like bi girls’, ‘straight women only’, ‘men who like men’
- **has_kids** – expected values: ‘has a kid’, “doesn’t have kids”, ‘has kids’
- **height_max** – The maximum height of returned search results. expected values: ‘A height int in inches’, ‘An imperial height string e.g. 5‘4”’, ‘A metric height string e.g. 1.54m’
- **height_min** – The minimum height of returned search results. expected values: ‘A height int in inches’, ‘An imperial height string e.g. 5‘4”’, ‘A metric height string e.g. 1.54m’

- **income** (*str*) – expected values: '$30,000-$40,000', '$20,000-$30,000', '$80,000-$100,000', '$100,000-$150,000', '$250,000-$500,000', 'less than $20,000', 'More than $1,000,000', '$500,000-$1,000,000', '$60,000-$70,000', '$70,000-$80,000', '$40,000-$50,000', '$50,000-$60,000', '$150,000-$250,000'
- **job** (*str*) – expected values: 'art', 'sales', 'engineering', 'politics', 'education', 'technology', 'management', 'entertainment', 'media', 'administration', 'writing', 'other', 'music', 'medicine', 'transportation', 'finance', 'retired', 'government', 'marketing', 'unemployed', 'construction', 'student', 'hospitality', 'law', 'rather not say', 'science', 'banking', 'military'
- **join_date** (*int*) – expected values: 'week', 'year', 'day', 'hour', 'month'
- **language** – expected values: 'portuguese', 'irish', 'chinese', 'czech', 'slovenian', 'sign language', 'hebrew', 'indonesian', 'rotuman', 'spanish', 'maori', 'slovak', 'mongolian', 'basque', 'urdu', 'polish', 'arabic', 'hungarian', 'esperanto', 'breton', 'italian', 'belarusan', 'icelandic', 'estonian', 'gujarati', 'occitan', 'serbian', 'sardinian', 'ancient greek', 'german', 'other', 'chechen', 'dutch', 'sanskrit', 'korean', 'farsi', 'hindi', 'danish', 'bulgarian', 'latin', 'khmer', 'latvian', 'hawaiian', 'ukrainian', 'welsh', 'georgian', 'lithuanian', 'malay', 'french', 'japanese', 'catalan', 'armenian', 'yiddish', 'swedish', 'russian', 'vietnamese', 'thai', 'afrikaans', 'tamil', 'cebuano', 'tagalog', 'finnish', 'norwegian', 'lisp', 'albanian', 'turkish', 'ilongo', 'romanian', 'c++', 'greek', 'persian', 'tibetan', 'frisian', 'english', 'croatian', 'swahili', 'bengali'
- **last_online** (*str*) – expected values: 'day', 'today', 'week', 'month', 'year', 'decade'
- **monogamy** (*str*) – expected values: '(:?[^\-]monogamous)|(:?^monogamous)', 'non-monogamous'
- **question** (`UserQuestion`) – A question whose answer should be used to match search results, or a question id. If a question id, *question_answers* must be supplied.
- **question_answers** (*list*) – A list of acceptable question answer indices.
- **question_count_min** (*int*) – The minimum number of questions answered by returned search results.
- **radius** (*int*) – The maximum distance from the specified location of returned search results.
- **religion** (*str*) – expected values: 'judaism', 'catholicism', 'buddhism', 'christianity', 'atheism', 'agnosticism', 'other', 'hinduism', 'islam'
- **sign** (*str*) – expected values: 'libra', 'sagittarius', 'cancer', 'scorpio', 'aquarius', 'taurus', 'leo', 'virgo', 'capricorn', 'gemini', 'aries', 'pisces'
- **smokes** (*str*) – expected values: 'yes', 'sometimes', 'trying to quit', 'no', 'when drinking'
- **status** (*str*) – The relationship status of returned search results. expected values: 'not single', 'married', 'single', 'any'
- **wants_kids** – expected values: 'wants', "doesn't want", 'might want'

`_AttractivenessFinder`

class **okcupyd.attractiveness_finder._AttractivenessFinder**(*session=None*) [source]

Find the attractiveness of okcupid.com users.

This class is typically wrapped in several different attractiveness finder decorators that allow for cacheing of results and rounding.

find_attractiveness(*username*, *accuracy=1000*, *_lower=0*, *_higher=10000*) [source]

Parameters:
- **username** – The username to lookup attractiveness for.
- **accuracy** – The accuracy required to return a result.
- **_lower** – The lower bound of the search.
- **_higher** – The upper bound of the search.

`Copy`

class **okcupyd.profile_copy.Copy**(*source_profile_or_user*, *dest_user*) [source]

Copy photos, essays and other attributes from one profile to another.

__init__(*source_profile_or_user*, *dest_user*) [source]

Parameters:
- **source_profile_or_user** – A `User` or `Profile` object from which to copy attributes. `questions()` will not will not preserve the importance of copied questions if a `Profile` instance is provided.
- **dest_user** – A `User` to which data will be copied

questions() [source]

Copy questions to the destination user. When this class was initialized with a `Profile`, this will delete any existing questions answers on the destination account.

photos() [source]

Copy photos to the destination user.

essays() [source]

Copy essays from the source profile to the destination profile.

looking_for() [source]

Copy looking for attributes from the source profile to the destination profile.

`details()` [source]

Copy details from the source profile to the destination profile.

`all()` [source]

Invoke all of `questions()`, `details()`, `essays()`, `photos()`, `looking_for()`

`Statistics`

class `okcupyd.statistics.Statistics`(*user*, *message_threads=None*, *filters=()*, *attractiveness_finder=None*) [source]

`add_command_line_options()`

`okcupyd.util.misc.add_command_line_options`(*add_argument*, *use_short_options=True*) [source]

Parameters:
- **add_argument** – The add_argument method of an ArgParser.
- **use_short_options** – Whether or not to add short options.

`handle_command_line_options()`

`okcupyd.util.misc.handle_command_line_options`(*args*) [source]

Parameters: **args** – The args returned from an ArgParser

Core Objects

`util`

class `okcupyd.util.cached_property`(*func*) [source]

Descriptor that caches the result of the first call to resolve its contents.

`bust_self`(*obj*) [source]

Remove the value that is being stored on *obj* for this `cached_property` object.

Parameters: **obj** – The instance on which to bust the cache.

classmethod **bust_caches**(*obj*, *excludes=()*) [source]

Bust the cache for all `cached_property` objects on *obj*

Parameters: **obj** – The instance on which to bust the caches.

class **okcupyd.util.REMap**(*re_value_pairs=()*, *default=<object object at 0x7ff8d7089c40>*) [source]

A mapping object that matches regular expressions to values.

classmethod **from_string_pairs**(*string_value_pairs*, ***kwargs*) [source]

Build an `REMap` from str, value pairs by applying *re.compile* to each string and calling the __init__ of `REMap`

okcupyd.util.IndexedREMap(**re_strings*, ***kwargs*) [source]

Build a `REMap` from the provided regular expression string. Each string will be associated with the index corresponding to its position in the argument list.

Parameters:
- **re_strings** – The re_strings that will serve as keys in the map.
- **default** – The value to return if none of the regular expressions match
- **offset** – The offset at which to start indexing for regular expressions defaults to 1.

`curry`

class **okcupyd.util.currying.curry** [source]

Curry a function or method.

Applying `curry` to a function creates a callable with the same functionality that can be invoked with an incomplete argument list to create a partial application of the original function.

```
@curry
def greater_than(x, y):
   return x > y

>>> less_than_40 = greater_than(40)
>>> less_than_40(39)
True
>>> less_than_40(50)
False
```

`curry` allows functions to be partially invoked an arbitary number of times:

```
@curry
def add_5_things(a, b, c, d, e):
    return a + b + c + d + e

# All of the following invocations of add_5_things
>>> add_5_things(1)(1)(1)(1)(1)
5

one_left = add_5_things(1, 1)(3)(4) # A one place function that will
# add 1 + 1 + 3 + 4 = 9 to whatever is provided as its argument.

>>>> one_left(5)
14
>>> one_left(6)
15
```

A particular compelling use case for `curry` is the creation of decorators that take optional arguments:

```
@curry
def add_n(function, n=1):
    def wrapped(*args, **kwargs):
        return function(*args, **kwargs) + n
    return wrapped

@add_n(n=12)
def multiply_plus_twelve(x, y):
    return x * y

@add_n
def multiply_plus_one(x, y):
    return x * y

>>> multiply_plus_one(1, 1)
2
>>> multiply_plus_twelve(1, 1)
13
```

Notice that we were able to apply *add_n* regardless of whether or not an optional argument had been supplied earlier.

The version of curry that is available for import has been curried itself. That is, its constructor can be invoked partially:

```
@curry(evaluation_checker=lambda *args, **kwargs: len(args) > 2)
def args_taking_function(*args):
    return reduce(lambda x, y: x*y, args)

>>> args_taking_function(1, 2)
2
```

```
>>> args_taking_function(2)(3)
6
>>> args_taking_function(2, 2, 2, 2)
16
```

function

alias of `curry`

fetchable

Most of the collection objects that are returned from function invocations in the okcupyd library are instances of `Fetchable`. In most cases, it is fine to treat these objects as though they are lists because they can be iterated over, sliced and accessed by index, just like lists:

```
for question in user.profile.questions:
    print(question.answer.text)

a_random_question = user.profile.questions[2]
for question in questions[2:4]:
    print(question.answer_options[0])
```

However, in some cases, it is important to be aware of the subtle differences between `Fetchable` objects and python lists. `Fetchable` construct the elements that they "contain" lazily. In most of its uses in the okcupyd library, this means that http requests can be made to populate `Fetchable` instances as its elments are requested.

The `questions` `Fetchable` that is used in the example above fetches the pages that are used to construct its contents in batches of 10 questions. This means that the actual call to retrieve data is made when iteration starts. If you enable the request logger when you run this code snippet, you get output that illustrates this fact:

```
2014-10-29 04:25:04 Livien-MacbookAir requests.packages.urllib3.connectionpool[82461] DEBUG
"GET /profile/ShrewdDrew/questions?leanmode=1&low=11 HTTP/1.1" 200 None
 Yes
 Yes
 Kiss someone.
 Yes.
 Yes
 Sex.
 Both equally
 No, I wouldn't give it as a gift.
 Maybe, I want to know all the important stuff.
 Once or twice a week
```

```
 2014-10-29 04:25:04 Livien-MacbookAir requests.packages.urllib3.connectionpool[82461] DEBUG
"GET /profile/ShrewdDrew/questions?leanmode=1&low=21 HTTP/1.1" 200 None
 No.
 No
 No
 Yes
 Rarely / never
 Always.
 Discovering your shared interests
 The sun
 Acceptable.
 No.
```

Some fetchables will continue fetching content for quite a long time. The search fetchable, for example, will fetch content until okcupid runs out of search results. As such, things like:

```
for profile in user.search():
    profile.message("hey!")
```

should be avoided, as they are likely to generate a massive number of requests to okcupid.com.

Another subtlety of the `Fetchable` class is that its instances cache its contained results. This means that the second iteration over `okcupyd.profile.Profile.questions` in the example below does not result in any http requests:

```
for question in user.profile.questions:
    print(question.text)

for question in user.profile.questions:
    print(question.answer)
```

It is important to understand that this means that the contents of a `Fetchable` are not guarenteed to be in sync with okcupid.com the second time they are requested. Calling `refresh()` will cause the `Fetchable` to request new data from okcupid.com when its contents are requested. The code snippet that follows prints out all the questions that the logged in user has answered roughly once per hour, including ones that are answered while the program is running.

```
import time

while True:
    for question in user.profile.questions:
        print(question.text)
    user.profile.questions.refresh()
```

```
    time.sleep(3600)
```

Without the call to *user.profile.questions.refresh()*, this program would never update the user.profile.questions instance, and thus what would be printed to the screen with each iteration of the for loop.

class **okcupyd.util.fetchable.Fetchable**(*fetcher*, ***kwargs*) [source]

List-like container object that lazily loads its contained items.

__init__(*fetcher*, ***kwargs*) [source]

Parameters:
- **fetcher** – An object with a *fetch* generator method that retrieves items for the fetchable.
- **nice_repr** – Append the repr of a list containing the items that have been fetched to this point by the fetcher. Defaults to True
- **kwargs** – Arguments that should be passed to the fetcher when it's fetch method is called. These are stored on the fetchable so they can be passed to the fetcher whenever `refresh()` is called.

refresh(*nice_repr=True*, ***kwargs*) [source]

Parameters:
- **nice_repr** (*bool*) – Append the repr of a list containing the items that have been fetched to this point by the fetcher.
- **kwargs** – kwargs that should be passed to the fetcher when its fetch method is called. These are merged with the values provided to the constructor, with the ones provided here taking precedence if there is a conflict.

class **okcupyd.util.fetchable.SimpleProcessor**(*session*, *object_factory*, *element_xpath*) [source]

Applies object_factory to each element found with element_xpath

Accepts session merely to be consistent with the FetchMarshall interface.

`Session`

class **okcupyd.session.Session**(*requests_session*) [source]

A *requests.Session* with convenience methods for interacting with okcupid.com

classmethod **login**(*username=None*, *password=None*, *requests_session=None*) [source]

Get a session that has authenticated with okcupid.com. If no username and password is supplied, the ones stored in `okcupyd.settings` will be used.

Parameters:
- **username** (*str*) – The username to log in with.
- **password** (*str*) – The password to log in with.

`get_profile(username)` [source]

Get the profile associated with the supplied username :param username: The username of the profile to retrieve.

`get_current_user_profile()` [source]

Get the *okcupyd.profile.Profile* associated with the supplied username.

Parameters: **username** – The username of the profile to retrieve.

`Filters`

class `okcupyd.filter.Filters` [source]

Registrar for functions that construct filters for submission in requests to okcupid.com

`build_documentation_lines()` [source]

Build a parameter documentation string that can appended to the docstring of a function that uses this `Filters` instance to build filters.

`register_filter_builder` [source]

Register a filter function with this `Filters` instance. This function is curried with `curry` – that is, it can be invoked partially before it is fully evaluated. This allows us to pass kwargs to this function when it is used as a decorator:

```
@register_filter_builder(keys=('real_name',),
                         decider=Filters.any_decider)
def my_filter_function(argument):
    return '4,{0}'.format(argument)
```

Parameters:
- **function** – The filter function to register.
- **keys** – Keys that should be used as the argument names for *function*, if none are provided, the filter functions argument names will be used instead.
- **decider** – a function of signature *(function, incoming_keys, accepted_keys)* that

returns True if the filter function should be called and False otherwise. Defaults to `all_not_none_decider()`

- **acceptable_values** – A list of acceptable values for the parameter of the filter function (or a list of lists if the filter function takes multiple parameters)
- **types** – The type of the parameter accepted by the incoming filter function (or a list of types if the function takes multiple parameters)
- **descriptions** – A description for the incoming filter function's argument (or a list of descriptions if the filter function takes multiple arguments)

Project Modules

okcupyd Package

`okcupyd` Package

okcupyd.__init__.interactive() [source]

class **okcupyd.__init__.User**(*session=None*)

Bases: `object`

Encapsulate a logged in okcupid user.

copy(*profile_or_user*)

Create a `Copy` instance with the provided object as the source and this `User` as the destination.

Parameters: **profile_or_user** – A `User` or `Profile` object.

delete_threads(*thread_ids_or_threads*)

Call `delete_threads()`.

Parameters: **thread_ids_or_threads** – A list whose members are either `MessageThread` instances or okc_ids of message threads.

classmethod **from_credentials**(*username*, *password*)

Parameters:
- **username** (*str*) – The username to log in with.
- **password** (*str*) – The password to log in with.

get_profile(*username*)

Get the `Profile` associated with the supplied username.

Parameters: **username** – The username of the profile to retrieve.

get_question_answer_id(*question*, *fast=False*, *bust_questions_cache=False*)

Get the index of the answer that was given to *question*

See the documentation for `get_user_question()` for important caveats about the use of this function.

Parameters:

- **question** (`BaseQuestion`) – The question whose *answer_id* should be retrieved.
- **fast** (*bool*) – Don't try to look through the users existing questions to see if arbitrarily answering the question can be avoided.
- **bust_questions_cache** (*bool*) –

  param bust_questions_cache:

  clear the

  `questions` attribute of this users `Profile` before looking for an existing answer. Be aware that even this does not eliminate all race conditions.

get_user_question(*question*, *fast=False*, *bust_questions_cache=False*)

Get a `UserQuestion` corresponding to the given `Question`.

HUGE CAVEATS: If the logged in user has not answered the relevant question, it will automatically be answered with whatever the first answer to the question is.

For the sake of reducing the number of requests made when this function is called repeatedly this function does not bust the cache of this `User` 's `okcupyd.profile.Profile.questions` attribute. That means that a question that HAS been answered could still be answered by this function if this `User` 's **`questions`** was populated previously (This population happens automatically – See `Fetchable` for details about when and how this happens).

Parameters:

- **question** (`BaseQuestion`) – The question for which a `UserQuestion` should be retrieved.
- **fast** (*bool*) – Don't try to look through the users existing questions to see if arbitrarily answering the question can be avoided.
- **bust_questions_cache** (*bool*) – clear the `questions` attribute of this users

`Profile` before looking for an existing answer. Be aware that even this does not eliminate all race conditions.

message(*username*, *message_text*)

Message an okcupid user. If an existing conversation between the logged in user and the target user can be found, reply to that thread instead of starting a new one.

Parameters:
- **username** (*str*) – The username of the user to which the message should be sent.
- **message_text** (*str*) – The body of the message.

quickmatch()

Return a `Profile` obtained by visiting the quickmatch page.

search(***kwargs*)

Call `SearchFetchable()` to get a `Fetchable` object that will lazily perform okcupid searches to provide `Profile` objects matching the search criteria.

Defaults for *gender*, *gentation*, *location* and *radius* will be provided if none are given.

Parameters: **kwargs** – See the `SearchFetchable()` docstring for details about what parameters are available.

username

Return the username associated with the `User`.

okcupyd.__init__.AttractivenessFinder(**args*, ***kwargs*)

class **okcupyd.__init__.Statistics**(*user*, *message_threads=None*, *filters=()*, *attractiveness_finder=None*)

Bases: `object`

attractiveness_filter(*attractiveness_finder=None*, *min_attractiveness=0*, *max_attractiveness=10000*)

average_attractiveness

`average_conversation_length`

`average_first_message_length`

`count`

`has_attractiveness`

`has_messages`

`has_response`

`initiated`

`no_responses`

`portion_initiated`

`portion_received`

`received`

`response_rate`

`threads`

`time_filter`(*min_date=None*, *max_date=None*)

`with_filters`(**filters*, ***kwargs*)

`okcupyd.__init__.save_file`(*filename*, *data*)

class `okcupyd.__init__.PhotoUploader`(*session=None*, *user_id=None*, *authcode=None*)

Bases: `object`

Upload photos to okcupid.com.

confirm(*pic_id*, ***kwargs*)

delete(*photo_id*, *album_id=0*)

Delete a photo from the logged in users account.

Parameters:
- **photo_id** – The okcupid id of the photo to delete.
- **album_id** – The album from which to delete the photo.

upload(*incoming*)

upload_and_confirm(*incoming*, ***kwargs*)

Upload the file to okcupid and confirm, among other things, its thumbnail position.

Parameters:
- **incoming** – A filepath string, `Info` object or a file like object to upload to okcupid.com. If an info object is provided, its thumbnail positioning will be used by default.
- **caption** – The caption to add to the photo.
- **thumb_nail_left** – For thumb nail positioning.
- **thumb_nail_top** – For thumb nail positioning.
- **thumb_nail_right** – For thumb nail positioning.
- **thumb_nail_bottom** – For thumb nail positioning.

upload_by_filename(*filename*)

upload_file(*file_object*, *image_type='jpeg'*)

upload_from_info(*info*)

class **okcupyd.__init__.Session**(*requests_session*)

Bases: `object`

A *requests.Session* with convenience methods for interacting with okcupid.com

build_path(*path*, *secure=None*)

default_login_headers
= {'user-agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_4) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/37.0.2062.94 Safari/537.36'}

do_login(*username*, *password*)

get_current_user_profile()

Get the *okcupyd.profile.Profile* associated with the supplied username.

Parameters: **username** – The username of the profile to retrieve.

get_profile(*username*)

Get the profile associated with the supplied username :param username: The username of the profile to retrieve.

classmethod **login**(*username=None*, *password=None*, *requests_session=None*)

Get a session that has authenticated with okcupid.com. If no username and password is supplied, the ones stored in `okcupyd.settings` will be used.

Parameters:
- **username** (*str*) – The username to log in with.
- **password** (*str*) – The password to log in with.

okc_delete(*path*, *secure=None*, ***kwargs*)

okc_get(*path*, *secure=None*, ***kwargs*)

okc_post(*path*, *secure=None*, ***kwargs*)

okc_put(*path*, *secure=None*, ***kwargs*)

`attractiveness_finder` Module

class **okcupyd.attractiveness_finder.AttractivenessFinderDecorator**(*attractiveness_finder=None*)
[source]

Bases: `object`

class **okcupyd.attractiveness_finder.CheckForExistenceAttractivenessFinder**(*attractiveness_finder=None*) [source]

Bases: `okcupyd.attractiveness_finder.AttractivenessFinderDecorator`

find_attractiveness(*username*, **args*, ***kwargs*) [source]

class **okcupyd.attractiveness_finder.RoundedAttractivenessFinder**(*attractiveness_finder=None*) [source]

Bases: `okcupyd.attractiveness_finder.AttractivenessFinderDecorator`

find_attractiveness(**args*, ***kwargs*) [source]

class **okcupyd.attractiveness_finder.CachedAttractivenessFinder**(*attractiveness_finder=None*) [source]

Bases: `okcupyd.attractiveness_finder.AttractivenessFinderDecorator`

find_attractiveness(*username*, ***kwargs*) [source]

`details` Module

class **okcupyd.details.Detail**(*id_name=None*, *presenter=None*, *updater=None*) [source]

Bases: `object`

Represent a detail belonging to an okcupid.com profile.

NO_DEFAULT= *<object object at 0x7ff8d7089c10>*

classmethod **comma_separated_presenter**(*text*) [source]

classmethod **mapping_multi_updater**(*mapping*) [source]

classmethod **auto_indexed_updater**(**options*) [source]

classmethod **mapping_updater**(*mapping*, *id_name=None*) [source]

static **default_updater**(*id_name*, *value*) [source]

id_name [source]

update(*value*) [source]

class **okcupyd.details.DeclarativeDetail** [source]

Bases: `object`

updater = *None*

presenter = *None*

class **okcupyd.details.Details**(*profile*) [source]

Bases: `object`

Represent the details belonging to an okcupid.com profile.

classmethod **name_detail_pairs**() [source]

refresh() [source]

id_to_display_name_value [source]

as_dict [source]

convert_and_update(*data*) [source]

update(*data*) [source]

bodytype

The bodytype detail of an okcupid.com user's profile.

orientation

The orientation detail of an okcupid.com user's profile.

`smokes`

The smoking detail of an okcupid.com user's profile.

`drugs`

The drugs detail of an okcupid.com user's profile.

`drinks`

The drinking detail of an okcupid.com user's profile.

`job`

The job detail of an okcupid.com user's profile.

`status`

The status detail of an okcupid.com user's profile.

`monogamy`

The monogamous detail of an okcupid.com user's profile.

`children`

The children detail of an okcupid.com user's profile.

`education`

The education detail of an okcupid.com user's profile.

`pets`

The pets detail of an okcupid.com user's profile.

`diet`

The diet detail of an okcupid.com user's profile.

`religion`

The religion detail of an okcupid.com user's profile.

`sign`

The sign detail of an okcupid.com user's profile.

height

The height detail of an okcupid.com user's profile.

ethnicities [source]

The ethnicities detail of an okcupid.com user's profile.

income [source]

The income detail of an okcupid.com user's profile.

languages [source]

The languages detail of an okcupid.com user's profile.

okcupyd.details.declarative_detail

alias of `languages`

okcupyd.details.is_declarative_detail(*x*)

okcupyd.details.detail

The status detail of an okcupid.com user's profile.

`errors` Module

exception **okcupyd.errors.AuthenticationError** [source]

Bases: `exceptions.Exception`

exception **okcupyd.errors.NoCorrespondentError** [source]

Bases: `exceptions.Exception`

`essay` Module

class **okcupyd.essay.Essays**(*profile*) [source]

Bases: `object`

Interface to reading and writing essays.

static **build_essay_property**(*essay_index*, *essay_name*) [source]

essay_names
= ['self_summary', 'my_life', 'good_at', 'people_first_notice', 'favorites', 'six_things', 'think_about', 'friday_night', 'private_admission', 'message_me_if']

A list of the attribute names that are used to store the text of of essays on instances of this class.

short_name_to_title [source]

refresh() [source]

favorites

friday_night

good_at

message_me_if

my_life

people_first_notice

private_admission

self_summary

six_things

think_about

`filter` Module

class **okcupyd.filter.Filters** [source]

Bases: `object`

Registrar for functions that construct filters for submission in requests to okcupid.com

build_documentation_lines() [source]

Build a parameter documentation string that can appended to the docstring of a function that uses this `Filters` instance to build filters.

build_paramter_string(*key*) [source]

static **any_decider**(*function*, *incoming*, *accepted_keys*) [source]

static **all_decider**(*function*, *incoming*, *accepted_keys*) [source]

static **all_not_none_decider**(*function*, *incoming*, *accepted_keys*) [source]

static **any_not_none_decider**(*function*, *incoming*, *accepted_keys*) [source]

filters(***kwargs*) [source]

build(***kwargs*) [source]

register_filter_builder [source]

Register a filter function with this `Filters` instance. This function is curried with `curry` – that is, it can be invoked partially before it is fully evaluated. This allows us to pass kwargs to this function when it is used as a decorator:

```
@register_filter_builder(keys=('real_name',),
                         decider=Filters.any_decider)
def my_filter_function(argument):
    return '4,{0}'.format(argument)
```

Parameters:
- **function** – The filter function to register.
- **keys** – Keys that should be used as the argument names for *function*, if none are provided, the filter functions argument names will be used instead.

- **decider** – a function of signature *(function, incoming_keys, accepted_keys)* that returns True if the filter function should be called and False otherwise. Defaults to `all_not_none_decider()`
- **acceptable_values** – A list of acceptable values for the parameter of the filter function (or a list of lists if the filter function takes multiple parameters)
- **types** – The type of the parameter accepted by the incoming filter function (or a list of types if the function takes multiple parameters)
- **descriptions** – A description for the incoming filter function's argument (or a list of descriptions if the filter function takes multiple arguments)

okcupyd.filter.gentation_filter(*gentation*) [source]

okcupyd.filter.age_filter(*age_min=18*, *age_max=99*) [source]

okcupyd.filter.location_filter(*radius*) [source]

`helpers` Module

class **okcupyd.helpers.MessageInfo**

Bases: `tuple`

MessageInfo(thread_id, message_id)

message_id

Alias for field number 1

thread_id

Alias for field number 0

class **okcupyd.helpers.Messager**(*session*) [source]

Bases: `object`

Send Messages to an okcupid user.

message_request_parameters(*username*, *message*, *thread_id*, *authcode*) [source]

send(*username*, *message*, *authcode=None*, *thread_id=None*) [source]

okcupyd.helpers.parse_fancydate(*fancydate_text*) [source]

okcupyd.helpers.parse_date_updated(*date_updated_text*) [source]

okcupyd.helpers.parse_contextual_date(*date_updated_text*) [source]

okcupyd.helpers.parse_slashed_date(*date_updated_text*) [source]

okcupyd.helpers.parse_abbreviated_date(*date_updated_text*) [source]

okcupyd.helpers.parse_time(*date_updated_text*) [source]

okcupyd.helpers.parse_day_of_the_week(*date_updated_text*) [source]

okcupyd.helpers.date_from_weekday(*weekday*) [source]

okcupyd.helpers.datetime_to_string(*a_datetime*) [source]

okcupyd.helpers.get_locid(*session*, *location*) [source]

Make a request to locquery resource to translate a string location search into an int locid. Returns ———- int

An int that OKCupid maps to a particular geographical location.

okcupyd.helpers.format_last_online(*last_online*) [source]

Return the upper limit in seconds that a profile may have been online. If last_online is an int, return that int. Otherwise if last_online is a str, convert the string into an int. Returns ———- int

okcupyd.helpers.update_looking_for(*profile_tree*, *looking_for*) [source]

Update looking_for attribute of a Profile.

okcupyd.helpers.update_details(*profile_tree*, *details*) [source]

Update details attribute of a Profile.

okcupyd.helpers.get_default_gentation(*gender*, *orientation*) [source]

Return the default gentation for the given gender and orientation.

okcupyd.helpers.replace_chars(*astring*) [source]

Replace certain unicode characters to avoid errors when trying to read various strings. Returns
———- str

okcupyd.helpers.add_newlines(*tree*) [source]

Add a newline character to the end of each
 element.

looking_for Module

okcupyd.looking_for.status_filter(*status*) [source]

class **okcupyd.looking_for.LookingFor**(*profile*) [source]

Bases: object

Represent the looking for attributes belonging to an okcupid.com profile.

raw_fields [source]

gentation [source]

The sex/orientation that the user is looking for.

ages [source]

The age range that the user is interested in.

single [source]

Whether or not the user is only interested in people that are single.

near_me [source]

Whether the user is only interested in people that are close to them.

kinds [source]

The kinds of relationship tha the user is looking for.

update(*ages=None*, *single=None*, *near_me=None*, *kinds=None*, *gentation=None*) [source]

Update the looking for attributes of the logged in user.

Parameters:
- **ages** (*tuple*) – The ages that the logged in user is interested in.
- **single** (*bool*) – Whether or not the user is only interested in people that are single.
- **near_me** (*bool*) – Whether or not the user is only interested in people that are near them.
- **kinds** (*list*) – What kinds of relationships the user should be updated to be interested in.
- **gentation** (*str*) – The sex/orientation of people the user is interested in.

class **Ages**

Bases: `tuple`

Ages(min, max)

max

Alias for field number 1

min

Alias for field number 0

`magicnumbers` Module

class **okcupyd.magicnumbers.maps** [source]

Bases: `object`

bodytype= *<okcupyd.util.REMap object at 0x7ff8d5f4fcd0>*

orientation= *<okcupyd.util.REMap object at 0x7ff8d5f5b290>*

smokes= *<okcupyd.util.REMap object at 0x7ff8d5f65550>*

drugs= *<okcupyd.util.REMap object at 0x7ff8d5f655d0>*

drinks= *<okcupyd.util.REMap object at 0x7ff8d5f65750>*

ethnicities= *<okcupyd.util.REMap object at 0x7ff8d5f65790>*

job= *<okcupyd.util.REMap object at 0x7ff8d5f657d0>*

status= *<okcupyd.util.REMap object at 0x7ff8d5f65810>*

monogamy= *<okcupyd.util.REMap object at 0x7ff8d5f65850>*

strictness= *<okcupyd.util.REMap object at 0x7ff8d5f65890>*

has_kids= *<okcupyd.util.REMap object at 0x7ff8d5f658d0>*

wants_kids= *<okcupyd.util.REMap object at 0x7ff8d5f65910>*

education_status= *<okcupyd.util.REMap object at 0x7ff8d5f65950>*

education_level= *<okcupyd.util.REMap object at 0x7ff8d5f65990>*

religion= *<okcupyd.util.REMap object at 0x7ff8d5f659d0>*

seriousness= *<okcupyd.util.REMap object at 0x7ff8d5f65a10>*

sign= *<okcupyd.util.REMap object at 0x7ff8d5f65a50>*

importance= *<okcupyd.util.REMap object at 0x7ff8d5f65a90>*

dogs= *<okcupyd.util.REMap object at 0x7ff8d5f65ad0>*

cats= *<okcupyd.util.REMap object at 0x7ff8d5f65b10>*

language_level= *<okcupyd.util.REMap object at 0x7ff8d5f65b50>*

diet_strictness= *<okcupyd.util.REMap object at 0x7ff8d5f65b90>*

diet= *<okcupyd.util.REMap object at 0x7ff8d5f65bd0>*

income= *<okcupyd.util.REMap object at 0x7ff8d5f65c10>*

class **okcupyd.magicnumbers.MappingUpdater**(*mapping*) [source]

Bases: `object`

class **okcupyd.magicnumbers.SimpleFilterBuilder**(*filter_number*, *mapping*, *offset=0*) [source]

Bases: `object`

get_number(*values*) [source]

get_filter(*values*) [source]

class **okcupyd.magicnumbers.filters** [source]

Bases: `object`

bodytype= *<okcupyd.magicnumbers.SimpleFilterBuilder object at 0x7ff8d5f65d10>*

smokes= *<okcupyd.magicnumbers.SimpleFilterBuilder object at 0x7ff8d5f65d50>*

drinks= *<okcupyd.magicnumbers.SimpleFilterBuilder object at 0x7ff8d5f65d90>*

drugs= *<okcupyd.magicnumbers.SimpleFilterBuilder object at 0x7ff8d5f65dd0>*

education_level= *<okcupyd.magicnumbers.SimpleFilterBuilder object at 0x7ff8d5f65e10>*

job= *<okcupyd.magicnumbers.SimpleFilterBuilder object at 0x7ff8d5f65e50>*

income= *<okcupyd.magicnumbers.SimpleFilterBuilder object at 0x7ff8d5f65e90>*

religion= *<okcupyd.magicnumbers.SimpleFilterBuilder object at 0x7ff8d5f65ed0>*

monogamy= *<okcupyd.magicnumbers.SimpleFilterBuilder object at 0x7ff8d5f65f10>*

diet= *<okcupyd.magicnumbers.SimpleFilterBuilder object at 0x7ff8d5f65f50>*

sign= *<okcupyd.magicnumbers.SimpleFilterBuilder object at 0x7ff8d5f65f90>*

ethnicities= *<okcupyd.magicnumbers.SimpleFilterBuilder object at 0x7ff8d5f65fd0>*

dogs= *<okcupyd.magicnumbers.SimpleFilterBuilder object at 0x7ff8d1856050>*

cats= *<okcupyd.magicnumbers.SimpleFilterBuilder object at 0x7ff8d1856090>*

okcupyd.magicnumbers.inches_to_centimeters(*inches*) [source]

okcupyd.magicnumbers.get_height_filter(*height_min=None*, *height_max=None*) [source]

okcupyd.magicnumbers.parse_height_string(*height_string*) [source]

okcupyd.magicnumbers.get_kids_filter(*has_kids=()*, *wants_kids=()*) [source]

okcupyd.magicnumbers.get_kids_int(*has_kids*, *wants_kids*) [source]

okcupyd.magicnumbers.subtract_has_kids_exponents(*value*) [source]

okcupyd.magicnumbers.yield_exponents_of_two(*value*) [source]

okcupyd.magicnumbers.get_language_query(*language*) [source]

okcupyd.magicnumbers.get_join_date_filter(*join_date*) [source]

okcupyd.magicnumbers.get_question_filter(*question*, *question_answers=None*) [source]

`messaging` Module

okcupyd.messaging.ThreadFetcher(*session*, *mailbox_number*) [source]

class okcupyd.messaging.ThreadHTMLFetcher(*session*, *mailbox_number*) [source]

Bases: `object`

fetch(*start_at*) [source]

class okcupyd.messaging.MessageFetcher(*session*, *message_thread*, *read_messages=False*) [source]

Bases: `object`

params [source]

messages_tree [source]

refresh() [source]

fetch() [source]

message_elements [source]

class okcupyd.messaging.Message(*message_element*, *message_thread*) [source]

Bases: `object`

Represent a message sent on okcupid.com

id [source]

Returns: The id assigned to this message by okcupid.com.

sender [source]

Returns: A `Profile` instance belonging to the sender of this message.

recipient [source]

Returns: A `Profile` instance belonging to the recipient of this message.

content [source]

Returns: The text body of the message.

time_sent [source]

class **okcupyd.messaging.MessageThread**(*session*, *thread_element*) [source]

Bases: `object`

Represent a message thread between two users.

classmethod **delete_threads**(*session*, *thread_ids_or_threads*, *authcode=None*) [source]

Parameters:
- **session** – A logged in `Session`.
- **thread_ids_or_threads** – A list whose members are either `MessageThread` instances or okc_ids of message threads.
- **authcode** – Authcode to use for this request. If none is provided A request to the logged in user's messages page will be made to retrieve one.

messages= *None*

A `Fetchable` of `Message` objects.

id [source]

Returns: The id assigned to this message by okcupid.com.

correspondent_id [source]

Returns: The id assigned to the correspondent of this message.

correspondent [source]

Returns: The username of the user with whom the logged in user is conversing in this `MessageThread`.

`read` [source]

Returns: Whether or not the user has read all the messages in this `MessageThread`.

`date` [source]

`datetime` [source]

`with_deleted_user` [source]

`initiator` [source]

Returns: A `Profile` instance belonging to the initiator of this `MessageThread`.

`respondent` [source]

Returns: A `Profile` instance belonging to the respondent of this `MessageThread`.

`correspondent_profile` [source]

Returns: The `Profile` of the user with whom the logged in user is conversing in this `MessageThread`.

`user_profile` [source]

Returns: A `Profile` belonging to the logged in user.

`message_count` [source]

`has_messages` [source]

`got_response` [source]

Returns: Whether or not the `MessageThread`. has received a response.

`delete()` [source]

Delete this thread for the logged in user.

`photo` Module

class **okcupyd.photo.PhotoUploader**(*session=None*, *user_id=None*, *authcode=None*) [source]

Bases: `object`

Upload photos to okcupid.com.

upload(*incoming*) [source]

upload_from_info(*info*) [source]

upload_by_filename(*filename*) [source]

upload_file(*file_object*, *image_type='jpeg'*) [source]

confirm(*pic_id*, ***kwargs*) [source]

upload_and_confirm(*incoming*, ***kwargs*) [source]

Upload the file to okcupid and confirm, among other things, its thumbnail position.

Parameters:
- **incoming** – A filepath string, `Info` object or a file like object to upload to okcupid.com. If an info object is provided, its thumbnail positioning will be used by default.
- **caption** – The caption to add to the photo.
- **thumb_nail_left** – For thumb nail positioning.
- **thumb_nail_top** – For thumb nail positioning.
- **thumb_nail_right** – For thumb nail positioning.
- **thumb_nail_bottom** – For thumb nail positioning.

delete(*photo_id*, *album_id=0*) [source]

Delete a photo from the logged in users account.

Parameters:
- **photo_id** – The okcupid id of the photo to delete.
- **album_id** – The album from which to delete the photo.

class **okcupyd.photo.Info**(*photo_id*, *tnl*, *tnt*, *tnr*, *tnb*) [source]

Bases: `object`

Represent a photo that appears on a okcupid.com user's profile.

base_uri= *'https://k0.okccdn.com/php/load_okc_image.php/images/'*

cdn_re= *<_sre.SRE_Pattern object at 0x162bb90>*

classmethod **from_cdn_uri**(*cdn_uri*) [source]

thumb_nail_left= *None*

The horizontal position of the left side of this photo's thumbnail.

thumb_nail_top= *None*

The vertical position of the top side of this photo's thumbnail.

thumb_nail_right= *None*

The horizontal position of the right side of this photo's thumbnail.

thumb_nail_bottom= *None*

The vertical position of the bottom side of this photo's thumbnail.

jpg_uri [source]

Returns: A uri from which this photo can be downloaded in jpg format.

`profile` Module

class **okcupyd.profile.Profile**(*session*, *username*, ***kwargs*) [source]

Bases: `object`

Represent the profile of an okcupid user.

Many of the attributes on this object are `cached_property` instances which lazily load their values, and cache them once they have been accessed. This makes it so that this object avoids making unnecessary HTTP requests to retrieve the same piece of information twice.

Because of this caching behavior, care must be taken to invalidate cached attributes on the object if an up to date view of the profile is needed. It is recommended that you call `refresh()` to accomplish this, but it is also possible to use `bust_self()` to bust individual properties if necessary.

username= *None*

The username of the user to whom this profile belongs.

questions= *None*

A `Fetchable` of `Question` instances, each corresponding to a question that has been answered by the user to whom this profile belongs. The fetchable consists of `UserQuestion` instead when the profile belongs to the logged in user.

details= *None*

A `Details` instance belonging to the same user that this profile belongs to.

refresh(*reload=False*) [source]

Parameters: **reload** – Make the request to return a new profile tree. This will result in the caching of the profile_tree attribute. The new profile_tree will be returned.

is_logged_in_user [source]

Returns: *True* if this profile and the session it was created with belong to the same user and False otherwise.

profile_tree [source]

Returns: a `lxml.etree` created from the html of the profile page of the account associated with the username that this profile was insantiated with.

message_request_parameters(*content*, *thread_id*) [source]

authcode [source]

photo_infos [source]

Returns: list of `Info` instances for each photo displayed on okcupid.

looking_for [source]

Returns: A `LookingFor` instance associated with this profile.

rating [source]

Deprecated. Use `liked()` instead.

Returns: the rating that the logged in user has given this user or 0 if no rating has been given.

liked [source]

Returns: Whether or not the logged in user liked this profile

contacted [source]

Retuns: A boolean indicating whether the logged in user has contacted the owner of this profile.

responds [source]

Returns: The frequency with which the user associated with this profile responds to messages.

id [source]

Returns: The id that okcupid.com associates with this profile.

essays [source]

Returns: A `Essays` instance that is associated with this profile.

age [source]

Returns: The age of the user associated with this profile.

match_percentage [source]

Returns: The match percentage of the logged in user and the user associated with this object.

enemy_percentage [source]

Returns: The enemy percentage of the logged in user and the user associated with this object.

location [source]

Returns: The location of the user associated with this profile.

gender [source]

The gender of the user associated with this profile.

orientation [source]

The sexual orientation of the user associated with this profile.

message [source]

Message the user associated with this profile.

Parameters:
- **message** – The message to send to this user.
- **thread_id** – The id of the thread to respond to, if any.

attractiveness [source]

Returns: The average attractiveness rating given to this profile by the okcupid.com community.

toggle_like() [source]

Toggle whether or not the logged in user likes this profile.

like() [source]

Like this profile.

unlike() [source]

Unlike this profile.

rate(*rating*) [source]

Rate this profile as the user that was logged in with the session that this object was instantiated with.

Parameters: **rating** – The rating to give this user.

find_question(*question_id*, *question_fetchable=None*) [source]

Parameters:
- **question_id** – The id of the question to search for
- **question_fetchable** – The question fetchable to iterate through if none is provided *self.questions* will be used.

question_fetchable(***kwargs*) [source]

Returns: A `Fetchable` instance that contains objects representing the answers that the user associated with this profile has given to okcupid.com match questions.

`authcode_get`(*path*, ***kwargs*) [source]

Perform an HTTP GET to okcupid.com using this profiles session where the authcode is automatically added as a query parameter.

`authcode_post`(*path*, ***kwargs*) [source]

Perform an HTTP POST to okcupid.com using this profiles session where the authcode is automatically added as a form item.

`profile_copy` Module

class `okcupyd.profile_copy.Copy`(*source_profile_or_user*, *dest_user*) [source]

Bases: `object`

Copy photos, essays and other attributes from one profile to another.

`copy_methods`= *['photos', 'essays', 'looking_for', 'details', 'questions']*

`questions`() [source]

Copy questions to the destination user. When this class was initialized with a `Profile`, this will delete any existing questions answers on the destination account.

`photos`() [source]

Copy photos to the destination user.

`essays`() [source]

Copy essays from the source profile to the destination profile.

`looking_for`() [source]

Copy looking for attributes from the source profile to the destination profile.

`details`() [source]

Copy details from the source profile to the destination profile.

`all`() [source]

Invoke all of `questions()`, `details()`, `essays()`, `photos()`, `looking_for()`

`question` Module

class **okcupyd.question.BaseQuestion**(*question_element*) [source]

Bases: `object`

The abstract abse class of `Question` and `UserQuestion`. Contains all the shared functinality of the aforementined classes. The two are quite different in some ways and can not always be used interchangably. See their respective docstrings for more details.

answered [source]

id [source]

Returns: The integer id given to this question by okcupid.com.

text [source]

class **okcupyd.question.Question**(*question_element*) [source]

Bases: `okcupyd.question.BaseQuestion`

Represent a question answered by a user other than the logged in user.

Note: Because of the way that okcupid presents question data it is actually not very easy to get the index of the answer to a question that belongs to a user other than the logged in user. It is possible to retrieve this value (see `okcupyd.user.User.get_question_answer_id()` and `get_answer_id_for_question()`), but it can take quite a few requests to do so. For this reason, the answer_id is NOT included as an attribute on this object, despite its inclusion in `UserQuestion`.

return_none_if_unanswered(*function*) [source]

their_answer [source]

The answer that the user whose `Profile` this question was retrieved from provided to this `Question`.

my_answer [source]

The answer that the user whose `Session` was used to create this `Question` provided.

their_answer_matches [source]

Returns: whether or not the answer provided by the user answering the question is acceptable to the logged in user.

Return type: rbool

my_answer_matches [source]

Returns: bool indicating whether or not the answer provided by the logged in user is acceptable to the user answering the question.

their_note [source]

Returns: The note the answering user provided as explanation for their answer to this question.

my_note [source]

Returns: The note the logged in user provided as an explanation for their answer to this question.

class **okcupyd.question.UserQuestion**(*question_element*) [source]

Bases: `okcupyd.question.BaseQuestion`

Represent a question answered by the logged in user.

get_answer_id_for_question(*question*) [source]

Get the answer_id corresponding to the answer given for question by looking at this `UserQuestion`'s answer_options. The given `Question` instance must have the same id as this `UserQuestion`.

That this method exists is admittedly somewhat weird. Unfortunately, it seems to be the only way to retrieve this information.

answer_id [source]

answer_options [source]

Returns: A list of `AnswerOption` instances representing the available answers to this question.

explanation [source]

Returns: The explanation written by the logged in user for this question (if any).

answer_text_to_option [source]

answer [source]

Returns: A `AnswerOption` instance corresponding to the answer the user gave to this question.

class **okcupyd.question.AnswerOption**(*option_element*) [source]

Bases: `object`

is_users [source]

Returns: Whether or not this was the answer selected by the logged in user.

is_match [source]

Returns: Whether or not this was the answer is acceptable to the logged in user.

text [source]

Returns: The text of this answer.

id [source]

Returns: The integer index associated with this answer.

class **okcupyd.question.Questions**(*session*, *importances=('not_important', 'little_important', 'somewhat_important', 'very_important', 'mandatory')*, *user_id=None*) [source]

Bases: `object`

Interface to accessing and answering questions belonging to the logged in user.

headers
*= {'origin': 'https://www.okcupid.com', 'accept-language': 'en-US,en;q=0.8', 'accept-encoding': 'gzip,deflate', 'accept': 'application/json, text/javascript, */*; q=0.01', 'referer': 'https://www.okcupid.com/questions', 'x-requested-with': 'XMLHttpRequest', 'content-type': 'application/x-www-form-urlencoded; charset=UTF-8'}*

importance_name_to_number
= {'not_important': 5, 'little_important': 4, 'mandatory': 0, 'very_important': 1, 'somewhat_important': 3}

Human readable importance name to integer used to represent them on okcupid.com

path= *'questions/ask'*

respond_from_user_question(*user_question*, *importance*) [source]

Respond to a question in exactly the way that is described by the given user_question.

Parameters:
- **user_question** (`UserQuestion`) – The user question to respond with.
- **importance** (int see `importance_name_to_number`) – The importance that should be used in responding to the question.

respond_from_question(*question*, *user_question*, *importance*) [source]

Copy the answer given in *question* to the logged in user's profile.

Parameters:
- **question** – A `Question` instance to copy.
- **user_question** – An instance of `UserQuestion` that corresponds to the same question as *question*. This is needed to retrieve the answer id from the question text answer on question.
- **importance** – The importance to assign to the response to the answered question.

respond(*question_id*, *user_response_ids*, *match_response_ids*, *importance*, *note=''*, *is_public=1*, *is_new=1*) [source]

Respond to an okcupid.com question.

Parameters:
- **question_id** – The okcupid id used to identify this question.
- **user_response_ids** – The answer id(s) to provide to this question.
- **match_response_ids** – The answer id(s) that the user considers acceptable.
- **importance** – The importance to attribute to this question. See `importance_name_to_number` for details.
- **note** – The explanation note to add to this question.
- **is_public** – Whether or not the question answer should be made public.

clear() [source]

USE WITH CAUTION. Delete the answer to every question that the logged in user has responded to.

okcupyd.question.QuestionProcessor(*question_class*) [source]

class okcupyd.question.QuestionHTMLFetcher(*session*, *uri*, ***additional_parameters*) [source]

Bases: object

classmethod from_username(*session*, *username*, ***kwargs*) [source]

fetch(*start_at*) [source]

okcupyd.question.QuestionFetcher(*session*, *username*, *question_class=<class 'okcupyd.question.Question'>*, *is_user=False*, ***kwargs*) [source]

search Module

okcupyd.search.search_filters
= Filters([<function option_filter at 0x7ff8d171a410>, <function option_filter at 0x7ff8d171ab90>, <function option_filter at 0x7ff8d171ad70>, <function option_filter at 0x7ff8d171a8c0>, <function attractiveness_filter at 0x7ff8d171a140>, <function question_count_filter at 0x7ff8d171a1b8>, <function option_filter at 0x7ff8d171a9b0>, <function get_kids_filter at 0x7ff8d18541b8>, <function status_filter at 0x7ff8d171a2a8>, <function get_question_filter at 0x7ff8d1854488>, <function get_language_query at 0x7ff8d1854398>, <function age_filter at 0x7ff8d1854c08>, <function get_join_date_filter at 0x7ff8d1854410>, <function option_filter at 0x7ff8d172b0c8>, <function location_filter at 0x7ff8d1854c80>, <function get_height_filter at 0x7ff8d18540c8>, <function option_filter at 0x7ff8d171ac80>, <function option_filter at 0x7ff8d171a500>, <function last_online_filter at 0x7ff8d171a230>, <function gentation_filter at 0x7ff8d1854578>, <function option_filter at 0x7ff8d171a5f0>, <function option_filter at 0x7ff8d171ae60>, <function option_filter at 0x7ff8d171a6e0>, <function option_filter at 0x7ff8d171a7d0>, <function option_filter at 0x7ff8d171af50>, <function option_filter at 0x7ff8d171aaa0>])

A Filters object that stores all of the filters that are accepted by SearchFetchable().

okcupyd.search.attractiveness_filter(*attractiveness_min*, *attractiveness_max*) [source]

okcupyd.search.question_count_filter(*question_count_min*) [source]

okcupyd.search.last_online_filter(*last_online*) [source]

okcupyd.search.status_filter(*status*) [source]

okcupyd.search.build_option_filter(*key*) [source]

class **okcupyd.search.MatchCardExtractor**(*div*) [source]

Bases: `object`

id [source]

username [source]

age [source]

location [source]

match_percentage [source]

enemy_percentage [source]

contacted [source]

as_dict [source]

okcupyd.search.SearchFetchable(*session=None*, ***kwargs*) [source]

Search okcupid.com with the given parameters. Parameters are registered to this function through `register_filter_builder()` of `search_filters`.

Returns: A `Fetchable` of `Profile` instances.

Parameters:

- **session** (`Session`) – A logged in session.
- **location** – A location string which will be used to filter results.
- **gender** – The gender of the user performing the search.
- **keywords** – A list or space delimeted string of words to search for.
- **order_by** – The criteria to use for ordering results. expected_values: ‘match’, ‘online’, ‘special_blend’
- **age_max** (*int*) – The maximum age of returned search results.
- **age_min** (*int*) – The minimum age of returned search results.
- **attractiveness_max** (*int*) – The maximum attractiveness of returned search results.
- **attractiveness_min** (*int*) – The minimum attractiveness of returned search results.
- **bodytype** (*str*) – expected values: ‘jacked’, ‘rather not say’, ‘fit’, ‘athletic’, ‘used up’, ‘average’, ‘full figured’, ‘overweight’, ‘curvy’, ‘thin’, ‘a little extra’, ‘skinny’

- **cats** (*str*) – expected values: ‘likes cats’, ‘dislikes cats’, ‘has cats’
- **diet** (*str*) – expected values: ‘anything’, ‘vegetarian’, ‘vegan’, ‘kosher’, ‘other’, ‘halal’
- **dogs** (*str*) – expected values: ‘dislikes dogs’, ‘has dogs’, ‘likes dogs’
- **drinks** (*str*) – expected values: ‘desperately’, ‘often’, ‘socially’, ‘very often’, ‘not at all’, ‘rarely’
- **drugs** (*str*) – expected values: ‘never’, ‘often’, ‘sometimes’
- **education_level** (*str*) – expected values: ‘law school’, ‘two[-]year college’, ‘university’, ‘space camp’, ‘high ?school’, ‘college’, ‘ph.d program’, ‘med school’, ‘masters program’
- **ethnicities** (*str*) – expected values: ‘latin’, ‘pacific islander’, ‘middle eastern’, ‘hispanic’, ‘indian’, ‘black’, ‘asian’, ‘white’, ‘other’, ‘native american’
- **gentation** (*str*) – The gentation of returned search results. expected values: ‘men and women who like bi women’, ‘gay girls only’, ‘bi guys only’, ‘gay guys only’, ‘’, ‘straight girls only’, ‘women who like men’, ‘bi men and women’, ‘guys who like girls’, ‘straight guys only’, ‘bi girls only’, ‘men and women who like bi men’, ‘guys and girls who like bi guys’, ‘both who like bi women’, ‘gay women only’, ‘gay men only’, ‘women’, ‘everybody’, ‘straight men only’, ‘girls who like girls’, ‘bi men only’, ‘both who like bi men’, ‘bi women only’, ‘guys who like guys’, ‘bi guys and girls’, ‘guys and girls who like bi girls’, ‘both who like bi guys’, ‘men who like women’, ‘girls who like guys’, ‘women who like women’, ‘both who like bi girls’, ‘straight women only’, ‘men who like men’
- **has_kids** – expected values: ‘has a kid’, “doesn’t have kids”, ‘has kids’
- **height_max** – The maximum height of returned search results. expected values: ‘A height int in inches’, ‘An imperial height string e.g. 5‘4”’, ‘A metric height string e.g. 1.54m’
- **height_min** – The minimum height of returned search results. expected values: ‘A height int in inches’, ‘An imperial height string e.g. 5‘4”’, ‘A metric height string e.g. 1.54m’
- **income** (*str*) – expected values: ‘$30,000-$40,000’, ‘$20,000-$30,000’, ‘$80,000-$100,000’, ‘$100,000-$150,000’, ‘$250,000-$500,000’, ‘less than $20,000’, ‘More than $1,000,000’, ‘$500,000-$1,000,000’, ‘$60,000-$70,000’, ‘$70,000-$80,000’, ‘$40,000-$50,000’, ‘$50,000-$60,000’, ‘$150,000-$250,000’
- **job** (*str*) – expected values: ‘art’, ‘sales’, ‘engineering’, ‘politics’, ‘education’, ‘technology’, ‘management’, ‘entertainment’, ‘media’, ‘administration’, ‘writing’, ‘other’, ‘music’, ‘medicine’, ‘transportation’, ‘finance’, ‘retired’, ‘government’, ‘marketing’, ‘unemployed’, ‘construction’, ‘student’, ‘hospitality’, ‘law’, ‘rather not say’, ‘science’, ‘banking’, ‘military’
- **join_date** (*int*) – expected values: ‘week’, ‘year’, ‘day’, ‘hour’, ‘month’
- **language** – expected values: ‘portuguese’, ‘irish’, ‘chinese’, ‘czech’, ‘slovenian’, ‘sign language’, ‘hebrew’, ‘indonesian’, ‘rotuman’, ‘spanish’, ‘maori’, ‘slovak’, ‘mongolian’,

'basque', 'urdu', 'polish', 'arabic', 'hungarian', 'esperanto', 'breton', 'italian', 'belarusan', 'icelandic', 'estonian', 'gujarati', 'occitan', 'serbian', 'sardinian', 'ancient greek', 'german', 'other', 'chechen', 'dutch', 'sanskrit', 'korean', 'farsi', 'hindi', 'danish', 'bulgarian', 'latin', 'khmer', 'latvian', 'hawaiian', 'ukrainian', 'welsh', 'georgian', 'lithuanian', 'malay', 'french', 'japanese', 'catalan', 'armenian', 'yiddish', 'swedish', 'russian', 'vietnamese', 'thai', 'afrikaans', 'tamil', 'cebuano', 'tagalog', 'finnish', 'norwegian', 'lisp', 'albanian', 'turkish', 'ilongo', 'romanian', 'c++', 'greek', 'persian', 'tibetan', 'frisian', 'english', 'croatian', 'swahili', 'bengali'

- **last_online** (*str*) – expected values: 'day', 'today', 'week', 'month', 'year', 'decade'
- **monogamy** (*str*) – expected values: '(:?[^\-]monogamous)|(:?^monogamous)', 'non-monogamous'
- **question** (`UserQuestion`) – A question whose answer should be used to match search results, or a question id. If a question id, *question_answers* must be supplied.
- **question_answers** (*list*) – A list of acceptable question answer indices.
- **question_count_min** (*int*) – The minimum number of questions answered by returned search results.
- **radius** (*int*) – The maximum distance from the specified location of returned search results.
- **religion** (*str*) – expected values: 'judaism', 'catholicism', 'buddhism', 'christianity', 'atheism', 'agnosticism', 'other', 'hinduism', 'islam'
- **sign** (*str*) – expected values: 'libra', 'sagittarius', 'cancer', 'scorpio', 'aquarius', 'taurus', 'leo', 'virgo', 'capricorn', 'gemini', 'aries', 'pisces'
- **smokes** (*str*) – expected values: 'yes', 'sometimes', 'trying to quit', 'no', 'when drinking'
- **status** (*str*) – The relationship status of returned search results. expected values: 'not single', 'married', 'single', 'any'
- **wants_kids** – expected values: 'wants', "doesn't want", 'might want'

class **okcupyd.search.SearchHTMLFetcher**(*session=None*, ***options*) [source]

Bases: `object`

fetch(*start_at=None*, *count=None*) [source]

okcupyd.search.search(*session=None*, *count=1*, ***kwargs*) [source]

`session` Module

class **okcupyd.session.Session**(*requests_session*) [source]

Bases: `object`

A *requests.Session* with convenience methods for interacting with okcupid.com

default_login_headers
= {'user-agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_4) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/37.0.2062.94 Safari/537.36'}

classmethod **login**(*username=None*, *password=None*, *requests_session=None*) [source]

Get a session that has authenticated with okcupid.com. If no username and password is supplied, the ones stored in `okcupyd.settings` will be used.

Parameters:
- **username** (*str*) – The username to log in with.
- **password** (*str*) – The password to log in with.

do_login(*username*, *password*) [source]

build_path(*path*, *secure=None*) [source]

get_profile(*username*) [source]

Get the profile associated with the supplied username :param username: The username of the profile to retrieve.

get_current_user_profile() [source]

Get the *okcupyd.profile.Profile* associated with the supplied username.

Parameters: **username** – The username of the profile to retrieve.

okc_delete(*path*, *secure=None*, ***kwargs*)

okc_get(*path*, *secure=None*, ***kwargs*)

okc_post(*path*, *secure=None*, ***kwargs*)

okc_put(*path*, *secure=None*, ***kwargs*)

okcupyd.session.build_okc_method(*method_name*) [source]

`settings` Module

Module where the default username and password for logging in to okcupid are housed.

okcupyd.settings.USERNAME= *None*

The username that will be used to log in to okcupid

okcupyd.settings.PASSWORD= *None*

The password that will be used to log in to okcupid

`statistics` Module

class **okcupyd.statistics.Statistics**(*user*, *message_threads=None*, *filters=()*, *attractiveness_finder=None*) [source]

Bases: `object`

threads [source]

has_messages [source]

has_response [source]

no_responses [source]

initiated [source]

received [source]

has_attractiveness [source]

time_filter(*min_date=None*, *max_date=None*) [source]

attractiveness_filter(*attractiveness_finder=None*, *min_attractiveness=0*, *max_attractiveness=10000*) [source]

`with_filters`(**filters*, ***kwargs*) [source]

`count` [source]

`response_rate` [source]

`average_first_message_length` [source]

`average_conversation_length` [source]

`average_attractiveness` [source]

`portion_initiated` [source]

`portion_received` [source]

`user` Module

class `okcupyd.user.User`(*session=None*) [source]

Bases: `object`

Encapsulate a logged in okcupid user.

classmethod `from_credentials`(*username*, *password*) [source]

Parameters:
- **username** (*str*) – The username to log in with.
- **password** (*str*) – The password to log in with.

`profile`= *None*

A `Profile` object belonging to the logged in user.

`inbox`= *None*

A `Fetchable` of `MessageThread` objects corresponding to messages that are currently in the user's inbox.

outbox= *None*

A `Fetchable` of `MessageThread` objects corresponding to messages that are currently in the user's outbox.

drafts= *None*

A `Fetchable` of `MessageThread` objects corresponding to messages that are currently in the user's drafts folder.

visitors= *None*

A `Fetchable` of `Profile` objects of okcupid.com users that have visited the user's profile.

questions= *None*

A `Questions` object that is instantiated with the owning `User` instance's session.

attractiveness_finder= *None*

An `_AttractivenessFinder` object that is instantiated with the owning `User` instance's session.

photo= *None*

A `PhotoUploader` that is instantiated with the owning `User` instance's session.

get_profile(*username*) [source]

Get the `Profile` associated with the supplied username.

Parameters: **username** – The username of the profile to retrieve.

username [source]

Return the username associated with the `User`.

message(*username*, *message_text*) [source]

Message an okcupid user. If an existing conversation between the logged in user and the target user can be found, reply to that thread instead of starting a new one.

Parameters:
- **username** (*str*) – The username of the user to which the message should be sent.
- **message_text** (*str*) – The body of the message.

search(***kwargs*) [source]

Call `SearchFetchable()` to get a `Fetchable` object that will lazily perform okcupid searches to provide `Profile` objects matching the search criteria.

Defaults for *gender*, *gentation*, *location* and *radius* will be provided if none are given.

Parameters: **kwargs** – See the `SearchFetchable()` docstring for details about what parameters are available.

delete_threads(*thread_ids_or_threads*) [source]

Call `delete_threads()`.

Parameters: **thread_ids_or_threads** – A list whose members are either `MessageThread` instances or okc_ids of message threads.

get_user_question(*question*, *fast=False*, *bust_questions_cache=False*) [source]

Get a `UserQuestion` corresponding to the given `Question`.

HUGE CAVEATS: If the logged in user has not answered the relevant question, it will automatically be answered with whatever the first answer to the question is.

For the sake of reducing the number of requests made when this function is called repeatedly this function does not bust the cache of this `User`'s `okcupyd.profile.Profile.questions` attribute. That means that a question that HAS been answered could still be answered by this function if this `User`'s `questions` was populated previously (This population happens automatically – See `Fetchable` for details about when and how this happens).

Parameters:

- **question** (`BaseQuestion`) – The question for which a `UserQuestion` should be retrieved.
- **fast** (*bool*) – Don't try to look through the users existing questions to see if arbitrarily answering the question can be avoided.
- **bust_questions_cache** (*bool*) – clear the `questions` attribute of this users `Profile` before looking for an existing answer. Be aware that even this does not eliminate all race conditions.

get_question_answer_id(*question*, *fast=False*, *bust_questions_cache=False*) [source]

Get the index of the answer that was given to *question*

See the documentation for `get_user_question()` for important caveats about the use of this function.

Parameters:
- **question** (`BaseQuestion`) – The question whose *answer_id* should be retrieved.
- **fast** (*bool*) – Don't try to look through the users existing questions to see if arbitrarily answering the question can be avoided.
- **bust_questions_cache** (*bool*) –

  param bust_questions_cache:

  clear the

  `questions` attribute of this users `Profile` before looking for an existing answer. Be aware that even this does not eliminate all race conditions.

quickmatch() [source]

Return a `Profile` obtained by visiting the quickmatch page.

copy(*profile_or_user*) [source]

Create a `Copy` instance with the provided object as the source and this `User` as the destination.

Parameters: **profile_or_user** – A `User` or `Profile` object.

`xpath` Module

class **okcupyd.xpath.XPathBuilder**(*nodes=()*, *relative=True*, *direct_child=False*) [source]

Bases: `object`

xpath [source]

or_ [source]

text_ [source]

add_node(***kwargs*) [source]

update_final_node(*updated_final_node*) [source]

attribute_contains(*attribute*, *contains_string*) [source]

with_classes(**classes*) [source]

select_attribute_(*attribute*, *elem=None*) [source]

text_contains_(*contained_text*) [source]

with_class(**classes*)

apply_(*tree*) [source]

one_(*tree*) [source]

get_text_(*tree*) [source]

class **okcupyd.xpath.XPathNode**(*element='*'*, *attributes=None*, *predicates=None*, *direct_child=False*, *use_or=False*, *selected_attribute=None*) [source]

Bases: `object`

text= *<object object at 0x7ff8d7089c30>*

static **contains_class**(*class_attribute*, *contained_class*) [source]

static **contains_attribute**(*attribute*, *contained_string*) [source]

static **attribute_equal**(*attribute*, *value*) [source]

make_or [source]

separator [source]

xpath [source]

predicate_joiner [source]

predicate_string [source]

selected_attribute_string [source]

with_classes(*classes*) [source]

add_contains_predicates(*kv_pairs*) [source]

text_contains(*contained_text*) [source]

okcupyd.xpath.xpb

Subpackages

db Package

db Package

class okcupyd.db.Query(*entities*, *session=None*) [source]

Bases: sqlalchemy.orm.query.Query

class okcupyd.db.txn(*session_class=None*) [source]

Bases: object

okcupyd.db.with_txn(*function*, *instance*, *args*, *kwargs*) [source]

class okcupyd.db.OKCBase(***kwargs*) [source]

Bases: sqlalchemy.ext.declarative.api.Base

okc_id= *Column(None, StringBackedInteger(), table=None, nullable=False)*

okcupyd.db.reset_engine(*engine*) [source]

okcupyd.db.set_sqlite_db_file(*file_path*) [source]

adapters Module

class **okcupyd.db.adapters.UserAdapter**(*profile*) [source]

Bases: object

build(*session*) [source]

get_no_txn(*session*) [source]

get(*session*)

class **okcupyd.db.adapters.ThreadAdapter**(*thread*) [source]

Bases: object

add_messages() [source]

get_thread() [source]

mailbox Module

class **okcupyd.db.mailbox.Sync**(*user*) [source]

Bases: object

Sync messages from a users inbox to the okc database.

all() [source]

update_mailbox [source]

Update the mailbox associated with the given mailbox name.

inbox

Update the mailbox associated with the given mailbox name.

outbox

Update the mailbox associated with the given mailbox name.

types Module

class **okcupyd.db.types.JSONType**(**args*, ***kwargs*) [source]

Bases: sqlalchemy.sql.type_api.TypeDecorator

impl

alias of VARCHAR

process_bind_param(*value*, *dialect*) [source]

process_result_value(*value*, *dialect*) [source]

class **okcupyd.db.types.StringBackedInteger**(**args*, ***kwargs*) [source]

Bases: sqlalchemy.sql.type_api.TypeDecorator

impl

alias of VARCHAR

process_bind_param(*value*, *dialect*) [source]

process_result_value(*value*, *dialect*) [source]

user Module

okcupyd.db.user.have_messaged_by_username_no_txn(*session*, *username_one*, *username_two*) [source]

okcupyd.db.user.have_messaged_by_username(*session*, *username_one*, *username_two*) [source]

okcupyd.db.user.and_(*clauses)

Produce a conjunction of expressions joined by `AND`.

E.g.:

```
from sqlalchemy import and_

stmt = select([users_table]).where(
                and_(
                    users_table.c.name == 'wendy',
                    users_table.c.enrolled == True
                )
            )
```

The `and_()` conjunction is also available using the Python `&` operator (though note that compound expressions need to be parenthesized in order to function with Python operator precedence behavior):

```
stmt = select([users_table]).where(
                (users_table.c.name == 'wendy') &
                (users_table.c.enrolled == True)
            )
```

The `and_()` operation is also implicit in some cases; the `Select.where()` method for example can be invoked multiple times against a statement, which will have the effect of each clause being combined using `and_()`:

```
stmt = select([users_table]).\
            where(users_table.c.name == 'wendy').\
            where(users_table.c.enrolled == True)
```

See also

`or_()`

okcupyd.db.user.or_(*clauses)

Produce a conjunction of expressions joined by `OR`.

E.g.:

```
from sqlalchemy import or_
```

```
stmt = select([users_table]).where(
                or_(
                    users_table.c.name == 'wendy',
                    users_table.c.name == 'jack'
                )
            )
```

The `or_()` conjunction is also available using the Python `|` operator (though note that compound expressions need to be parenthesized in order to function with Python operator precedence behavior):

```
stmt = select([users_table]).where(
                (users_table.c.name == 'wendy') |
                (users_table.c.name == 'jack')
            )
```

See also

`and_()`

Subpackages

model Package¶

`model` Package¶

`message` Module¶

class **okcupyd.db.model.message.Message**(***kwargs*) [source]

Bases: `okcupyd.db.OKCBase`

time_sent

message_thread_id

sender_id

sender

recipient_id

recipient

`text`

`thread_index`

`created_at`

`id`

`okc_id`

okcupyd.db.model.message.relationship(*argument*, *secondary=None*, *primaryjoin=None*, *secondaryjoin=None*, *foreign_keys=None*, *uselist=None*, *order_by=False*, *backref=None*, *back_populates=None*, *post_update=False*, *cascade=False*, *extension=None*, *viewonly=False*, *lazy=True*, *collection_class=None*, *passive_deletes=False*, *passive_updates=True*, *remote_side=None*, *enable_typechecks=True*, *join_depth=None*, *comparator_factory=None*, *single_parent=False*, *innerjoin=False*, *distinct_target_key=None*, *doc=None*, *active_history=False*, *cascade_backrefs=True*, *load_on_pending=False*, *strategy_class=None*, *_local_remote_pairs=None*, *query_class=None*, *info=None*)

Provide a relationship between two mapped classes.

This corresponds to a parent-child or associative table relationship. The constructed class is an instance of `RelationshipProperty`.

A typical `relationship()`, used in a classical mapping:

```
mapper(Parent, properties={
  'children': relationship(Child)
})
```

Some arguments accepted by `relationship()` optionally accept a callable function, which when called produces the desired value. The callable is invoked by the parent `Mapper` at “mapper initialization” time, which happens only when mappers are first used, and is assumed to be after all mappings have been constructed. This can be used to resolve order-of-declaration and other dependency issues, such as if `Child` is declared below `Parent` in the same file:

```
mapper(Parent, properties={
    "children":relationship(lambda: Child,
                        order_by=lambda: Child.id)
})
```

When using the *declarative_toplevel* extension, the Declarative initializer allows string arguments to be passed to `relationship()`. These string arguments are converted into callables that evaluate the string as Python code, using the Declarative class-registry as a namespace. This allows the lookup of related classes to be automatic via their string name, and removes the need to import related classes at all into the local module space:

```
from sqlalchemy.ext.declarative import declarative_base

Base = declarative_base()

class Parent(Base):
    __tablename__ = 'parent'
    id = Column(Integer, primary_key=True)
    children = relationship("Child", order_by="Child.id")
```

See also

relationship_config_toplevel - Full introductory and reference documentation for `relationship()`.

orm_tutorial_relationship - ORM tutorial introduction.

Parameters:

- **argument** –
  a mapped class, or actual `Mapper` instance, representing the target of the relationship.

  :paramref:`~.relationship.argument` may also be passed as a callable function which is evaluated at mapper initialization time, and may be passed as a Python-evaluable string when using Declarative.

  See also

  declarative_configuring_relationships - further detail on relationship configuration when using Declarative.

- **secondary** –
  for a many-to-many relationship, specifies the intermediary table, and is typically an instance of `Table`. In less common circumstances, the argument may also be specified as an `Alias` construct, or even a `Join` construct.

  :paramref:`~.relationship.secondary` may also be passed as a callable function which is evaluated at mapper initialization time. When using Declarative, it may also be a string argument noting the name of a `Table`

that is present in the `MetaData` collection associated with the parent-mapped `Table`.

The :paramref:`~.relationship.secondary` keyword argument is typically applied in the case where the intermediary `Table` is not otherwise exprssed in any direct class mapping. If the "secondary" table is also explicitly mapped elsewhere (e.g. as in *association_pattern*), one should consider applying the :paramref:`~.relationship.viewonly` flag so that this `relationship()` is not used for persistence operations which may conflict with those of the association object pattern.

See also

relationships_many_to_many - Reference example of "many to many".

orm_tutorial_many_to_many - ORM tutorial introduction to many-to-many relationships.

self_referential_many_to_many - Specifics on using many-to-many in a self-referential case.

declarative_many_to_many - Additional options when using Declarative.

association_pattern - an alternative to :paramref:`~.relationship.secondary` when composing association table relationships, allowing additional attributes to be specified on the association table.

composite_secondary_join - a lesser-used pattern which in some cases can enable complex `relationship()` SQL conditions to be used.

New in version 0.9.2: :paramref:`~.relationship.secondary` works more effectively when referring to a `Join` instance.

- **active_history=False** – When `True`, indicates that the "previous" value for a many-to-one reference should be loaded when replaced, if not already loaded. Normally, history tracking logic for simple many-to-ones only needs to be aware of the "new" value in order to perform a flush. This flag is available for applications that make use of `attributes.get_history()` which also need to know the "previous" value of the attribute.
- **backref** –
  indicates the string name of a property to be placed on the related mapper's class that will handle this relationship in the other direction. The other property will be created automatically when the mappers are configured. Can also be passed as a `backref()` object to control the configuration of the new relationship.

See also

relationships_backref - Introductory documentation and examples.

:paramref:`~.relationship.back_populates` - alternative form of backref specification.

`backref()` - allows control over `relationship()` configuration when using :paramref:`~.relationship.backref`.

- **back_populates** –
  Takes a string name and has the same meaning as :paramref:`~.relationship.backref`, except the complementing property is **not** created automatically, and instead must be configured explicitly on the other mapper. The complementing property should also indicate :paramref:`~.relationship.back_populates` to this relationship to ensure proper functioning.

  See also

  relationships_backref - Introductory documentation and examples.

  :paramref:`~.relationship.backref` - alternative form of backref specification.

- **cascade** –
  a comma-separated list of cascade rules which determines how Session operations should be “cascaded” from parent to child. This defaults to `False`, which means the default cascade should be used - this default cascade is `"save-update, merge"`.

  The available cascades are `save-update`, `merge`, `expunge`, `delete`, `delete-orphan`, and `refresh-expire`. An additional option, `all` indicates shorthand for `"save-update, merge, refresh-expire, expunge, delete"`, and is often used as in `"all, delete-orphan"` to indicate that related objects should follow along with the parent object in all cases, and be deleted when de-associated.

  See also

  unitofwork_cascades - Full detail on each of the available cascade options.

  tutorial_delete_cascade - Tutorial example describing a delete cascade.

- **cascade_backrefs=True** –
  a boolean value indicating if the `save-update` cascade should operate along an assignment event intercepted by a backref. When set to `False`, the attribute managed by this relationship will not cascade an incoming transient object into the session of a persistent parent, if the event is received via backref.

See also

backref_cascade - Full discussion and examples on how the :paramref:`~.relationship.cascade_backrefs` option is used.

- **collection_class** –
  a class or callable that returns a new list-holding object. will be used in place of a plain list for storing elements.

  See also

  custom_collections - Introductory documentation and examples.

- **comparator_factory** –
  a class which extends `RelationshipProperty.Comparator` which provides custom SQL clause generation for comparison operations.

  See also

  `PropComparator` - some detail on redefining comparators at this level.

  custom_comparators - Brief intro to this feature.

- **distinct_target_key=None** –
  Indicate if a “subquery” eager load should apply the DISTINCT keyword to the innermost SELECT statement. When left as `None`, the DISTINCT keyword will be applied in those cases when the target columns do not comprise the full primary key of the target table. When set to `True`, the DISTINCT keyword is applied to the innermost SELECT unconditionally.

  It may be desirable to set this flag to False when the DISTINCT is reducing performance of the innermost subquery beyond that of what duplicate innermost rows may be causing.

  New in version 0.8.3: - :paramref:`~.relationship.distinct_target_key` allows the subquery eager loader to apply a DISTINCT modifier to the innermost SELECT.

  Changed in version 0.9.0: - :paramref:`~.relationship.distinct_target_key` now defaults to `None`, so that the feature enables itself automatically for those cases where the innermost query targets a non-unique key.

  See also

  loading_toplevel - includes an introduction to subquery eager loading.

doc – docstring which will be applied to the resulting descriptor.

- **extension** –
  an `AttributeExtension` instance, or list of extensions, which will be prepended to the list of attribute listeners for the resulting descriptor placed on the class.

  Deprecated since version 0.7: Please see `AttributeEvents`.

- **foreign_keys** –
  a list of columns which are to be used as “foreign key” columns, or columns which refer to the value in a remote column, within the context of this `relationship()` object’s :paramref:`~.relationship.primaryjoin` condition. That is, if the :paramref:`~.relationship.primaryjoin` condition of this `relationship()` is `a.id == b.a_id`, and the values in `b.a_id` are required to be present in `a.id`, then the “foreign key” column of this `relationship()` is `b.a_id`.

  In normal cases, the :paramref:`~.relationship.foreign_keys` parameter is **not required.** `relationship()` will automatically determine which columns in the :paramref:`~.relationship.primaryjoin` conditition are to be considered “foreign key” columns based on those `Column` objects that specify `ForeignKey`, or are otherwise listed as referencing columns in a `ForeignKeyConstraint` construct. :paramref:`~.relationship.foreign_keys` is only needed when:

  - There is more than one way to construct a join from the local table to the remote table, as there are multiple foreign key references present. Setting `foreign_keys` will limit the `relationship()` to consider just those columns specified here as “foreign”.
    Changed in version 0.8: A multiple-foreign key join ambiguity can be resolved by setting the :paramref:`~.relationship.foreign_keys` parameter alone, without the need to explicitly set :paramref:`~.relationship.primaryjoin` as well.
  - The `Table` being mapped does not actually have `ForeignKey` or `ForeignKeyConstraint` constructs present, often because the table was reflected from a database that does not support foreign key reflection (MySQL MyISAM).
  - The :paramref:`~.relationship.primaryjoin` argument is used to construct a non-standard join condition, which makes use of columns or expressions that do not normally refer to their “parent” column, such as a join condition expressed by a complex comparison using a SQL function.

  The `relationship()` construct will raise informative error messages that suggest the use of the :paramref:`~.relationship.foreign_keys` parameter when presented with an ambiguous condition. In typical cases, if `relationship()` doesn’t raise any exceptions, the :paramref:`~.relationship.foreign_keys` parameter is usually not needed.

:paramref:`~.relationship.foreign_keys` may also be passed as a callable function which is evaluated at mapper initialization time, and may be passed as a Python-evaluable string when using Declarative.

See also

relationship_foreign_keys

relationship_custom_foreign

`foreign()` - allows direct annotation of the “foreign” columns within a :paramref:`~.relationship.primaryjoin` condition.

New in version 0.8: The `foreign()` annotation can also be applied directly to the :paramref:`~.relationship.primaryjoin` expression, which is an alternate, more specific system of describing which columns in a particular :paramref:`~.relationship.primaryjoin` should be considered “foreign”.

- **info** –
  Optional data dictionary which will be populated into the `MapperProperty.info` attribute of this object.

  New in version 0.8.

- **innerjoin=False** –
  when `True`, joined eager loads will use an inner join to join against related tables instead of an outer join. The purpose of this option is generally one of performance, as inner joins generally perform better than outer joins.

  This flag can be set to `True` when the relationship references an object via many-to-one using local foreign keys that are not nullable, or when the reference is one-to-one or a collection that is guaranteed to have one or at least one entry.

  If the joined-eager load is chained onto an existing LEFT OUTER JOIN, `innerjoin=True` will be bypassed and the join will continue to chain as LEFT OUTER JOIN so that the results don’t change. As an alternative, specify the value `"nested"`. This will instead nest the join on the right side, e.g. using the form “a LEFT OUTER JOIN (b JOIN c)”.

  New in version 0.9.4: Added `innerjoin="nested"` option to support nesting of eager “inner” joins.

  See also

  what_kind_of_loading - Discussion of some details of various loader options.

  :paramref:`.joinedload.innerjoin` - loader option version

- **join_depth** –

when non-`None`, an integer value indicating how many levels deep “eager” loaders should join on a self-referring or cyclical relationship. The number counts how many times the same Mapper shall be present in the loading condition along a particular join branch. When left at its default of `None`, eager loaders will stop chaining when they encounter a the same target mapper which is already higher up in the chain. This option applies both to joined- and subquery- eager loaders.

See also

self_referential_eager_loading - Introductory documentation and examples.

- **lazy=’select’** –
  specifies how the related items should be loaded. Default value is `select`. Values include:
  - `select` - items should be loaded lazily when the property is first accessed, using a separate SELECT statement, or identity map fetch for simple many-to-one references.
  - `immediate` - items should be loaded as the parents are loaded, using a separate SELECT statement, or identity map fetch for simple many-to-one references.
  - `joined` - items should be loaded “eagerly” in the same query as that of the parent, using a JOIN or LEFT OUTER JOIN. Whether the join is “outer” or not is determined by the :paramref:`~.relationship.innerjoin` parameter.
  - `subquery` - items should be loaded “eagerly” as the parents are loaded, using one additional SQL statement, which issues a JOIN to a subquery of the original statement, for each collection requested.
  - `noload` - no loading should occur at any time. This is to support “write-only” attributes, or attributes which are populated in some manner specific to the application.
  - `dynamic` - the attribute will return a pre-configured `Query` object for all read operations, onto which further filtering operations can be applied before iterating the results. See the section *dynamic_relationship* for more details.
  - True - a synonym for ‘select’
  - False - a synonym for ‘joined’
  - None - a synonym for ‘noload’

See also

`/orm/loading` - Full documentation on relationship loader configuration.

dynamic_relationship - detail on the `dynamic` option.

- **load_on_pending=False** –
  Indicates loading behavior for transient or pending parent objects.

  When set to `True`, causes the lazy-loader to issue a query for a parent object that is not persistent, meaning it has never been flushed. This may take effect for a pending object when autoflush is disabled, or for a transient object that has been "attached" to a `Session` but is not part of its pending collection.

  The :paramref:`~.relationship.load_on_pending` flag does not improve behavior when the ORM is used normally - object references should be constructed at the object level, not at the foreign key level, so that they are present in an ordinary way before a flush proceeds. This flag is not not intended for general use.

  See also

  `Session.enable_relationship_loading()` - this method establishes "load on pending" behavior for the whole object, and also allows loading on objects that remain transient or detached.

- **order_by** –
  indicates the ordering that should be applied when loading these items. :paramref:`~.relationship.order_by` is expected to refer to one of the `Column` objects to which the target class is mapped, or the attribute itself bound to the target class which refers to the column.

  :paramref:`~.relationship.order_by` may also be passed as a callable function which is evaluated at mapper initialization time, and may be passed as a Python-evaluable string when using Declarative.

- **passive_deletes=False** –
  Indicates loading behavior during delete operations.

  A value of True indicates that unloaded child items should not be loaded during a delete operation on the parent. Normally, when a parent item is deleted, all child items are loaded so that they can either be marked as deleted, or have their foreign key to the parent set to NULL. Marking this flag as True usually implies an ON DELETE <CASCADE|SET NULL> rule is in place which will handle updating/deleting child rows on the database side.

  Additionally, setting the flag to the string value 'all' will disable the "nulling out" of the child foreign keys, when there is no delete or delete-orphan cascade enabled. This is typically used when a triggering or error raise scenario is in place on the database side. Note that the foreign key attributes on in-session child objects will not be changed after a flush occurs so this is a very special use-case setting.

  See also

passive_deletes - Introductory documentation and examples.

- **passive_updates=True** –
  Indicates loading and INSERT/UPDATE/DELETE behavior when the source of a foreign key value changes (i.e. an “on update” cascade), which are typically the primary key columns of the source row.

  When True, it is assumed that ON UPDATE CASCADE is configured on the foreign key in the database, and that the database will handle propagation of an UPDATE from a source column to dependent rows. Note that with databases which enforce referential integrity (i.e. PostgreSQL, MySQL with InnoDB tables), ON UPDATE CASCADE is required for this operation. The relationship() will update the value of the attribute on related items which are locally present in the session during a flush.

  When False, it is assumed that the database does not enforce referential integrity and will not be issuing its own CASCADE operation for an update. The relationship() will issue the appropriate UPDATE statements to the database in response to the change of a referenced key, and items locally present in the session during a flush will also be refreshed.

  This flag should probably be set to False if primary key changes are expected and the database in use doesn’t support CASCADE (i.e. SQLite, MySQL MyISAM tables).

  See also

  passive_updates - Introductory documentation and examples.

  :paramref:`.mapper.passive_updates` - a similar flag which takes effect for joined-table inheritance mappings.

- **post_update** –
  this indicates that the relationship should be handled by a second UPDATE statement after an INSERT or before a DELETE. Currently, it also will issue an UPDATE after the instance was UPDATEd as well, although this technically should be improved. This flag is used to handle saving bi-directional dependencies between two individual rows (i.e. each row references the other), where it would otherwise be impossible to INSERT or DELETE both rows fully since one row exists before the other. Use this flag when a particular mapping arrangement will incur two rows that are dependent on each other, such as a table that has a one-to-many relationship to a set of child rows, and also has a column that references a single child row within that list (i.e. both tables contain a foreign key to each other). If a flush operation returns an error that a “cyclical dependency” was detected, this is a cue that you might want to use :paramref:`~.relationship.post_update` to “break” the cycle.

  See also

post_update - Introductory documentation and examples.

- **primaryjoin** –
  a SQL expression that will be used as the primary join of this child object against the parent object, or in a many-to-many relationship the join of the primary object to the association table. By default, this value is computed based on the foreign key relationships of the parent and child tables (or association table).

  :paramref:`~.relationship.primaryjoin` may also be passed as a callable function which is evaluated at mapper initialization time, and may be passed as a Python-evaluable string when using Declarative.

  See also

  relationship_primaryjoin

- **remote_side** –
  used for self-referential relationships, indicates the column or list of columns that form the “remote side” of the relationship.

  :paramref:`.relationship.remote_side` may also be passed as a callable function which is evaluated at mapper initialization time, and may be passed as a Python-evaluable string when using Declarative.

  Changed in version 0.8: The `remote()` annotation can also be applied directly to the `primaryjoin` expression, which is an alternate, more specific system of describing which columns in a particular `primaryjoin` should be considered “remote”.

  See also

  self_referential - in-depth explanation of how :paramref:`~.relationship.remote_side` is used to configure self-referential relationships.

  `remote()` - an annotation function that accomplishes the same purpose as :paramref:`~.relationship.remote_side`, typically when a custom :paramref:`~.relationship.primaryjoin` condition is used.

- **query_class** –
  a `Query` subclass that will be used as the base of the “appender query” returned by a “dynamic” relationship, that is, a relationship that specifies `lazy="dynamic"` or was otherwise constructed using the `orm.dynamic_loader()` function.

  See also

dynamic_relationship - Introduction to “dynamic” relationship loaders.

- **secondaryjoin** –
  a SQL expression that will be used as the join of an association table to the child object. By default, this value is computed based on the foreign key relationships of the association and child tables.

  :paramref:`~.relationship.secondaryjoin` may also be passed as a callable function which is evaluated at mapper initialization time, and may be passed as a Python-evaluable string when using Declarative.

  See also

  relationship_primaryjoin

- **single_parent** –
  when True, installs a validator which will prevent objects from being associated with more than one parent at a time. This is used for many-to-one or many-to-many relationships that should be treated either as one-to-one or one-to-many. Its usage is optional, except for `relationship()` constructs which are many-to-one or many-to-many and also specify the `delete-orphan` cascade option. The `relationship()` construct itself will raise an error instructing when this option is required.

  See also

  unitofwork_cascades - includes detail on when the :paramref:`~.relationship.single_parent` flag may be appropriate.

- **uselist** –
  a boolean that indicates if this property should be loaded as a list or a scalar. In most cases, this value is determined automatically by `relationship()` at mapper configuration time, based on the type and direction of the relationship - one to many forms a list, many to one forms a scalar, many to many is a list. If a scalar is desired where normally a list would be present, such as a bi-directional one-to-one relationship, set :paramref:`~.relationship.uselist` to False.

  The :paramref:`~.relationship.uselist` flag is also available on an existing `relationship()` construct as a read-only attribute, which can be used to determine if this `relationship()` deals with collections or scalar attributes:

  ```
  >>> User.addresses.property.uselist
  True
  ```

  See also

relationships_one_to_one - Introduction to the “one to one” relationship pattern, which is typically when the :paramref:`~.relationship.uselist` flag is needed.

- **viewonly=False** – when set to True, the relationship is used only for loading objects, and not for any persistence operation. A `relationship()` which specifies :paramref:`~.relationship.viewonly` can work with a wider range of SQL operations within the :paramref:`~.relationship.primaryjoin` condition, including operations that feature the use of a variety of comparison operators as well as SQL functions such as `cast()`. The :paramref:`~.relationship.viewonly` flag is also of general use when defining any kind of `relationship()` that doesn’t represent the full set of related objects, to prevent modifications of the collection from resulting in persistence operations.

`message_thread` Module¶

class **okcupyd.db.model.message_thread.MessageThread**(*****kwargs***) [source]

Bases: `okcupyd.db.OKCBase`

initiator_id

initiator

respondent_id

respondent

messages

created_at

id

okc_id

okcupyd.db.model.message_thread.relationship(*argument*, *secondary=None*,

primaryjoin=None, secondaryjoin=None, foreign_keys=None, uselist=None, order_by=False, backref=None, back_populates=None, post_update=False, cascade=False, extension=None, viewonly=False, lazy=True, collection_class=None, passive_deletes=False, passive_updates=True, remote_side=None, enable_typechecks=True, join_depth=None, comparator_factory=None, single_parent=False, innerjoin=False, distinct_target_key=None, doc=None, active_history=False, cascade_backrefs=True, load_on_pending=False, strategy_class=None, _local_remote_pairs=None, query_class=None, info=None)

Provide a relationship between two mapped classes.

This corresponds to a parent-child or associative table relationship. The constructed class is an instance of `RelationshipProperty`.

A typical `relationship()`, used in a classical mapping:

```
mapper(Parent, properties={
  'children': relationship(Child)
})
```

Some arguments accepted by `relationship()` optionally accept a callable function, which when called produces the desired value. The callable is invoked by the parent `Mapper` at "mapper initialization" time, which happens only when mappers are first used, and is assumed to be after all mappings have been constructed. This can be used to resolve order-of-declaration and other dependency issues, such as if `Child` is declared below `Parent` in the same file:

```
mapper(Parent, properties={
    "children":relationship(lambda: Child,
                        order_by=lambda: Child.id)
})
```

When using the *declarative_toplevel* extension, the Declarative initializer allows string arguments to be passed to `relationship()`. These string arguments are converted into callables that evaluate the string as Python code, using the Declarative class-registry as a namespace. This allows the lookup of related classes to be automatic via their string name, and removes the need to import related classes at all into the local module space:

```
from sqlalchemy.ext.declarative import declarative_base

Base = declarative_base()

class Parent(Base):
    __tablename__ = 'parent'
    id = Column(Integer, primary_key=True)
    children = relationship("Child", order_by="Child.id")
```

See also

relationship_config_toplevel - Full introductory and reference documentation for `relationship()`.

orm_tutorial_relationship - ORM tutorial introduction.

Parameters:

- **argument** –
  a mapped class, or actual `Mapper` instance, representing the target of the relationship.

  :paramref:`~.relationship.argument` may also be passed as a callable function which is evaluated at mapper initialization time, and may be passed as a Python-evaluable string when using Declarative.

  See also

  declarative_configuring_relationships - further detail on relationship configuration when using Declarative.

- **secondary** –
  for a many-to-many relationship, specifies the intermediary table, and is typically an instance of `Table`. In less common circumstances, the argument may also be specified as an `Alias` construct, or even a `Join` construct.

  :paramref:`~.relationship.secondary` may also be passed as a callable function which is evaluated at mapper initialization time. When using Declarative, it may also be a string argument noting the name of a `Table` that is present in the `MetaData` collection associated with the parent-mapped `Table`.

  The :paramref:`~.relationship.secondary` keyword argument is typically applied in the case where the intermediary `Table` is not otherwise exprssed in any direct class mapping. If the “secondary” table is also explicitly mapped elsewhere (e.g. as in *association_pattern*), one should consider applying the :paramref:`~.relationship.viewonly` flag so that this `relationship()` is not used for persistence operations which may conflict with those of the association object pattern.

  See also

  relationships_many_to_many - Reference example of “many to many”.

  orm_tutorial_many_to_many - ORM tutorial introduction to many-to-many relationships.

  self_referential_many_to_many - Specifics on using many-to-many in a self-referential case.

declarative_many_to_many - Additional options when using Declarative.

association_pattern - an alternative to :paramref:`~.relationship.secondary` when composing association table relationships, allowing additional attributes to be specified on the association table.

composite_secondary_join - a lesser-used pattern which in some cases can enable complex `relationship()` SQL conditions to be used.

New in version 0.9.2: :paramref:`~.relationship.secondary` works more effectively when referring to a `Join` instance.

- **active_history=False** – When `True`, indicates that the "previous" value for a many-to-one reference should be loaded when replaced, if not already loaded. Normally, history tracking logic for simple many-to-ones only needs to be aware of the "new" value in order to perform a flush. This flag is available for applications that make use of `attributes.get_history()` which also need to know the "previous" value of the attribute.
- **backref** –
  indicates the string name of a property to be placed on the related mapper's class that will handle this relationship in the other direction. The other property will be created automatically when the mappers are configured. Can also be passed as a `backref()` object to control the configuration of the new relationship.

  See also

  relationships_backref - Introductory documentation and examples.

  :paramref:`~.relationship.back_populates` - alternative form of backref specification.

  `backref()` - allows control over `relationship()` configuration when using :paramref:`~.relationship.backref`.

- **back_populates** –
  Takes a string name and has the same meaning as :paramref:`~.relationship.backref`, except the complementing property is **not** created automatically, and instead must be configured explicitly on the other mapper. The complementing property should also indicate :paramref:`~.relationship.back_populates` to this relationship to ensure proper functioning.

  See also

  relationships_backref - Introductory documentation and examples.

:paramref:`~.relationship.backref` - alternative form of backref specification.

- **cascade** –
  a comma-separated list of cascade rules which determines how Session operations should be “cascaded” from parent to child. This defaults to `False`, which means the default cascade should be used - this default cascade is `"save-update, merge"`.

  The available cascades are `save-update`, `merge`, `expunge`, `delete`, `delete-orphan`, and `refresh-expire`. An additional option, `all` indicates shorthand for `"save-update, merge, refresh-expire, expunge, delete"`, and is often used as in `"all, delete-orphan"` to indicate that related objects should follow along with the parent object in all cases, and be deleted when de-associated.

  See also

  unitofwork_cascades - Full detail on each of the available cascade options.

  tutorial_delete_cascade - Tutorial example describing a delete cascade.

- **cascade_backrefs=True** –
  a boolean value indicating if the `save-update` cascade should operate along an assignment event intercepted by a backref. When set to `False`, the attribute managed by this relationship will not cascade an incoming transient object into the session of a persistent parent, if the event is received via backref.

  See also

  backref_cascade - Full discussion and examples on how the :paramref:`~.relationship.cascade_backrefs` option is used.

- **collection_class** –
  a class or callable that returns a new list-holding object. will be used in place of a plain list for storing elements.

  See also

  custom_collections - Introductory documentation and examples.

- **comparator_factory** –
  a class which extends `RelationshipProperty.Comparator` which provides

custom SQL clause generation for comparison operations.

See also

`PropComparator` - some detail on redefining comparators at this level.

custom_comparators - Brief intro to this feature.

- **distinct_target_key=None** –
  Indicate if a “subquery” eager load should apply the DISTINCT keyword to the innermost SELECT statement. When left as `None`, the DISTINCT keyword will be applied in those cases when the target columns do not comprise the full primary key of the target table. When set to `True`, the DISTINCT keyword is applied to the innermost SELECT unconditionally.

  It may be desirable to set this flag to False when the DISTINCT is reducing performance of the innermost subquery beyond that of what duplicate innermost rows may be causing.

  New in version 0.8.3: - :paramref:`~.relationship.distinct_target_key` allows the subquery eager loader to apply a DISTINCT modifier to the innermost SELECT.

  Changed in version 0.9.0: - :paramref:`~.relationship.distinct_target_key` now defaults to `None`, so that the feature enables itself automatically for those cases where the innermost query targets a non-unique key.

  See also

  loading_toplevel - includes an introduction to subquery eager loading.

- **doc** – docstring which will be applied to the resulting descriptor.
- **extension** –
  an `AttributeExtension` instance, or list of extensions, which will be prepended to the list of attribute listeners for the resulting descriptor placed on the class.

  Deprecated since version 0.7: Please see `AttributeEvents`.

- **foreign_keys** –
  a list of columns which are to be used as “foreign key” columns, or columns which refer to the value in a remote column, within the context of this `relationship()` object’s :paramref:`~.relationship.primaryjoin` condition. That is, if the :paramref:`~.relationship.primaryjoin` condition of this `relationship()` is `a.id == b.a_id`, and the values in `b.a_id` are required to be present in `a.id`, then the “foreign key” column of this `relationship()` is `b.a_id`.

  In normal cases, the :paramref:`~.relationship.foreign_keys` parameter is **not required.** `relationship()` will automatically determine which columns

in the :paramref:`~.relationship.primaryjoin` condititon are to be considered “foreign key” columns based on those `Column` objects that specify `ForeignKey`, or are otherwise listed as referencing columns in a `ForeignKeyConstraint` construct. :paramref:`~.relationship.foreign_keys` is only needed when:

- There is more than one way to construct a join from the local table to the remote table, as there are multiple foreign key references present. Setting `foreign_keys` will limit the `relationship()` to consider just those columns specified here as “foreign”.
  Changed in version 0.8: A multiple-foreign key join ambiguity can be resolved by setting the :paramref:`~.relationship.foreign_keys` parameter alone, without the need to explicitly set :paramref:`~.relationship.primaryjoin` as well.
- The `Table` being mapped does not actually have `ForeignKey` or `ForeignKeyConstraint` constructs present, often because the table was reflected from a database that does not support foreign key reflection (MySQL MyISAM).
- The :paramref:`~.relationship.primaryjoin` argument is used to construct a non-standard join condition, which makes use of columns or expressions that do not normally refer to their “parent” column, such as a join condition expressed by a complex comparison using a SQL function.

The `relationship()` construct will raise informative error messages that suggest the use of the :paramref:`~.relationship.foreign_keys` parameter when presented with an ambiguous condition. In typical cases, if `relationship()` doesn’t raise any exceptions, the :paramref:`~.relationship.foreign_keys` parameter is usually not needed.

:paramref:`~.relationship.foreign_keys` may also be passed as a callable function which is evaluated at mapper initialization time, and may be passed as a Python-evaluable string when using Declarative.

See also

relationship_foreign_keys

relationship_custom_foreign

`foreign()` - allows direct annotation of the “foreign” columns within a :paramref:`~.relationship.primaryjoin` condition.

New in version 0.8: The `foreign()` annotation can also be applied directly to the :paramref:`~.relationship.primaryjoin` expression, which is an alternate, more specific system of describing which columns in a particular :paramref:`~.relationship.primaryjoin` should be considered “foreign”.

- **info** –

Optional data dictionary which will be populated into the `MapperProperty.info` attribute of this object.

New in version 0.8.

- **innerjoin=False** –
  when `True`, joined eager loads will use an inner join to join against related tables instead of an outer join. The purpose of this option is generally one of performance, as inner joins generally perform better than outer joins.

  This flag can be set to `True` when the relationship references an object via many-to-one using local foreign keys that are not nullable, or when the reference is one-to-one or a collection that is guaranteed to have one or at least one entry.

  If the joined-eager load is chained onto an existing LEFT OUTER JOIN, `innerjoin=True` will be bypassed and the join will continue to chain as LEFT OUTER JOIN so that the results don't change. As an alternative, specify the value `"nested"`. This will instead nest the join on the right side, e.g. using the form "a LEFT OUTER JOIN (b JOIN c)".

  New in version 0.9.4: Added `innerjoin="nested"` option to support nesting of eager "inner" joins.

  See also

  what_kind_of_loading - Discussion of some details of various loader options.

  :paramref:`.joinedload.innerjoin` - loader option version

- **join_depth** –
  when non-`None`, an integer value indicating how many levels deep "eager" loaders should join on a self-referring or cyclical relationship. The number counts how many times the same Mapper shall be present in the loading condition along a particular join branch. When left at its default of `None`, eager loaders will stop chaining when they encounter a the same target mapper which is already higher up in the chain. This option applies both to joined- and subquery- eager loaders.

  See also

  self_referential_eager_loading - Introductory documentation and examples.

- **lazy='select'** –
  specifies how the related items should be loaded. Default value is `select`. Values include:

  - `select` - items should be loaded lazily when the property is first accessed,

using a separate SELECT statement, or identity map fetch for simple many-to-one references.

- `immediate` - items should be loaded as the parents are loaded, using a separate SELECT statement, or identity map fetch for simple many-to-one references.
- `joined` - items should be loaded “eagerly” in the same query as that of the parent, using a JOIN or LEFT OUTER JOIN. Whether the join is “outer” or not is determined by the :paramref:`~.relationship.innerjoin` parameter.
- `subquery` - items should be loaded “eagerly” as the parents are loaded, using one additional SQL statement, which issues a JOIN to a subquery of the original statement, for each collection requested.
- `noload` - no loading should occur at any time. This is to support “write-only” attributes, or attributes which are populated in some manner specific to the application.
- `dynamic` - the attribute will return a pre-configured `Query` object for all read operations, onto which further filtering operations can be applied before iterating the results. See the section *dynamic_relationship* for more details.
- True - a synonym for ‘select’
- False - a synonym for ‘joined’
- None - a synonym for ‘noload’

See also

`/orm/loading` - Full documentation on relationship loader configuration.

dynamic_relationship - detail on the `dynamic` option.

- **load_on_pending=False** –
  Indicates loading behavior for transient or pending parent objects.

  When set to `True`, causes the lazy-loader to issue a query for a parent object that is not persistent, meaning it has never been flushed. This may take effect for a pending object when autoflush is disabled, or for a transient object that has been “attached” to a `Session` but is not part of its pending collection.

  The :paramref:`~.relationship.load_on_pending` flag does not improve behavior when the ORM is used normally - object references should be constructed at the object level, not at the foreign key level, so that they are present in an ordinary way before a flush proceeds. This flag is not not intended for general use.

See also

`Session.enable_relationship_loading()` - this method establishes “load on pending” behavior for the whole object, and also allows loading on

objects that remain transient or detached.

- **order_by** –
  indicates the ordering that should be applied when loading these items. :paramref:`~.relationship.order_by` is expected to refer to one of the `Column` objects to which the target class is mapped, or the attribute itself bound to the target class which refers to the column.

  :paramref:`~.relationship.order_by` may also be passed as a callable function which is evaluated at mapper initialization time, and may be passed as a Python-evaluable string when using Declarative.

- **passive_deletes=False** –
  Indicates loading behavior during delete operations.

  A value of True indicates that unloaded child items should not be loaded during a delete operation on the parent. Normally, when a parent item is deleted, all child items are loaded so that they can either be marked as deleted, or have their foreign key to the parent set to NULL. Marking this flag as True usually implies an ON DELETE <CASCADE|SET NULL> rule is in place which will handle updating/deleting child rows on the database side.

  Additionally, setting the flag to the string value 'all' will disable the "nulling out" of the child foreign keys, when there is no delete or delete-orphan cascade enabled. This is typically used when a triggering or error raise scenario is in place on the database side. Note that the foreign key attributes on in-session child objects will not be changed after a flush occurs so this is a very special use-case setting.

  See also

  passive_deletes - Introductory documentation and examples.

- **passive_updates=True** –
  Indicates loading and INSERT/UPDATE/DELETE behavior when the source of a foreign key value changes (i.e. an "on update" cascade), which are typically the primary key columns of the source row.

  When True, it is assumed that ON UPDATE CASCADE is configured on the foreign key in the database, and that the database will handle propagation of an UPDATE from a source column to dependent rows. Note that with databases which enforce referential integrity (i.e. PostgreSQL, MySQL with InnoDB tables), ON UPDATE CASCADE is required for this operation. The relationship() will update the value of the attribute on related items which are locally present in the session during a flush.

  When False, it is assumed that the database does not enforce referential integrity and will not be issuing its own CASCADE operation for an update. The relationship() will issue the appropriate UPDATE statements

to the database in response to the change of a referenced key, and items locally present in the session during a flush will also be refreshed.

This flag should probably be set to False if primary key changes are expected and the database in use doesn't support CASCADE (i.e. SQLite, MySQL MyISAM tables).

See also

passive_updates - Introductory documentation and examples.

:paramref:`.mapper.passive_updates` - a similar flag which takes effect for joined-table inheritance mappings.

- **post_update** –
  this indicates that the relationship should be handled by a second UPDATE statement after an INSERT or before a DELETE. Currently, it also will issue an UPDATE after the instance was UPDATEd as well, although this technically should be improved. This flag is used to handle saving bi-directional dependencies between two individual rows (i.e. each row references the other), where it would otherwise be impossible to INSERT or DELETE both rows fully since one row exists before the other. Use this flag when a particular mapping arrangement will incur two rows that are dependent on each other, such as a table that has a one-to-many relationship to a set of child rows, and also has a column that references a single child row within that list (i.e. both tables contain a foreign key to each other). If a flush operation returns an error that a "cyclical dependency" was detected, this is a cue that you might want to use :paramref:`~.relationship.post_update` to "break" the cycle.

  See also

  post_update - Introductory documentation and examples.

- **primaryjoin** –
  a SQL expression that will be used as the primary join of this child object against the parent object, or in a many-to-many relationship the join of the primary object to the association table. By default, this value is computed based on the foreign key relationships of the parent and child tables (or association table).

  :paramref:`~.relationship.primaryjoin` may also be passed as a callable function which is evaluated at mapper initialization time, and may be passed as a Python-evaluable string when using Declarative.

  See also

  relationship_primaryjoin

- **remote_side** –
  used for self-referential relationships, indicates the column or list of columns that form the “remote side” of the relationship.

  :paramref:`.relationship.remote_side` may also be passed as a callable function which is evaluated at mapper initialization time, and may be passed as a Python-evaluable string when using Declarative.

  Changed in version 0.8: The `remote()` annotation can also be applied directly to the `primaryjoin` expression, which is an alternate, more specific system of describing which columns in a particular `primaryjoin` should be considered “remote”.

  See also

  self_referential - in-depth explanation of how :paramref:`~.relationship.remote_side` is used to configure self-referential relationships.

  `remote()` - an annotation function that accomplishes the same purpose as :paramref:`~.relationship.remote_side`, typically when a custom :paramref:`~.relationship.primaryjoin` condition is used.

- **query_class** –
  a `Query` subclass that will be used as the base of the “appender query” returned by a “dynamic” relationship, that is, a relationship that specifies `lazy="dynamic"` or was otherwise constructed using the `orm.dynamic_loader()` function.

  See also

  dynamic_relationship - Introduction to “dynamic” relationship loaders.

- **secondaryjoin** –
  a SQL expression that will be used as the join of an association table to the child object. By default, this value is computed based on the foreign key relationships of the association and child tables.

  :paramref:`~.relationship.secondaryjoin` may also be passed as a callable function which is evaluated at mapper initialization time, and may be passed as a Python-evaluable string when using Declarative.

  See also

  relationship_primaryjoin

- **single_parent** –
  when True, installs a validator which will prevent objects from being

associated with more than one parent at a time. This is used for many-to-one or many-to-many relationships that should be treated either as one-to-one or one-to-many. Its usage is optional, except for `relationship()` constructs which are many-to-one or many-to-many and also specify the `delete-orphan` cascade option. The `relationship()` construct itself will raise an error instructing when this option is required.

See also

unitofwork_cascades - includes detail on when the :paramref:`~.relationship.single_parent` flag may be appropriate.

- **uselist** –
  a boolean that indicates if this property should be loaded as a list or a scalar. In most cases, this value is determined automatically by `relationship()` at mapper configuration time, based on the type and direction of the relationship - one to many forms a list, many to one forms a scalar, many to many is a list. If a scalar is desired where normally a list would be present, such as a bi-directional one-to-one relationship, set :paramref:`~.relationship.uselist` to False.

  The :paramref:`~.relationship.uselist` flag is also available on an existing `relationship()` construct as a read-only attribute, which can be used to determine if this `relationship()` deals with collections or scalar attributes:

```
>>> User.addresses.property.uselist
True
```

See also

relationships_one_to_one - Introduction to the “one to one” relationship pattern, which is typically when the :paramref:`~.relationship.uselist` flag is needed.

- **viewonly=False** – when set to True, the relationship is used only for loading objects, and not for any persistence operation. A `relationship()` which specifies :paramref:`~.relationship.viewonly` can work with a wider range of SQL operations within the :paramref:`~.relationship.primaryjoin` condition, including operations that feature the use of a variety of comparison operators as well as SQL functions such as `cast()`. The :paramref:`~.relationship.viewonly` flag is also of general use when defining any kind of `relationship()` that doesn’t represent the full set of related objects, to prevent modifications of the collection from resulting in persistence operations.

user Module¶

class **okcupyd.db.model.user.User**(***kwargs*) [source]

Bases: okcupyd.db.OKCBase

classmethod **from_profile**(*profile*) [source]

handle

age

location

created_at

id

okc_id

class **okcupyd.db.model.user.OKCupydUser**(***kwargs*) [source]

Bases: sqlalchemy.ext.declarative.api.Base

inbox_last_updated

outbox_last_updated

user_id

user

created_at

id

okcupyd.db.model.user.relationship(*argument, secondary=None, primaryjoin=None,*

secondaryjoin=None, foreign_keys=None, uselist=None, order_by=False, backref=None, back_populates=None, post_update=False, cascade=False, extension=None, viewonly=False, lazy=True, collection_class=None, passive_deletes=False, passive_updates=True, remote_side=None, enable_typechecks=True, join_depth=None, comparator_factory=None, single_parent=False, innerjoin=False, distinct_target_key=None, doc=None, active_history=False, cascade_backrefs=True, load_on_pending=False, strategy_class=None, _local_remote_pairs=None, query_class=None, info=None)

Provide a relationship between two mapped classes.

This corresponds to a parent-child or associative table relationship. The constructed class is an instance of `RelationshipProperty`.

A typical `relationship()`, used in a classical mapping:

```
mapper(Parent, properties={
  'children': relationship(Child)
})
```

Some arguments accepted by `relationship()` optionally accept a callable function, which when called produces the desired value. The callable is invoked by the parent `Mapper` at “mapper initialization” time, which happens only when mappers are first used, and is assumed to be after all mappings have been constructed. This can be used to resolve order-of-declaration and other dependency issues, such as if `Child` is declared below `Parent` in the same file:

```
mapper(Parent, properties={
    "children":relationship(lambda: Child,
                        order_by=lambda: Child.id)
})
```

When using the *declarative_toplevel* extension, the Declarative initializer allows string arguments to be passed to `relationship()`. These string arguments are converted into callables that evaluate the string as Python code, using the Declarative class-registry as a namespace. This allows the lookup of related classes to be automatic via their string name, and removes the need to import related classes at all into the local module space:

```
from sqlalchemy.ext.declarative import declarative_base

Base = declarative_base()

class Parent(Base):
    __tablename__ = 'parent'
    id = Column(Integer, primary_key=True)
    children = relationship("Child", order_by="Child.id")
```

See also

relationship_config_toplevel - Full introductory and reference documentation for `relationship()`.

orm_tutorial_relationship - ORM tutorial introduction.

Parameters:

- **argument** –
  a mapped class, or actual `Mapper` instance, representing the target of the relationship.

  :paramref:`~.relationship.argument` may also be passed as a callable function which is evaluated at mapper initialization time, and may be passed as a Python-evaluable string when using Declarative.

  See also

  declarative_configuring_relationships - further detail on relationship configuration when using Declarative.

- **secondary** –
  for a many-to-many relationship, specifies the intermediary table, and is typically an instance of `Table`. In less common circumstances, the argument may also be specified as an `Alias` construct, or even a `Join` construct.

  :paramref:`~.relationship.secondary` may also be passed as a callable function which is evaluated at mapper initialization time. When using Declarative, it may also be a string argument noting the name of a `Table` that is present in the `MetaData` collection associated with the parent-mapped `Table`.

  The :paramref:`~.relationship.secondary` keyword argument is typically applied in the case where the intermediary `Table` is not otherwise exprssed in any direct class mapping. If the “secondary” table is also explicitly mapped elsewhere (e.g. as in *association_pattern*), one should consider applying the :paramref:`~.relationship.viewonly` flag so that this `relationship()` is not used for persistence operations which may conflict with those of the association object pattern.

  See also

  relationships_many_to_many - Reference example of “many to many”.

  orm_tutorial_many_to_many - ORM tutorial introduction to many-to-many relationships.

  self_referential_many_to_many - Specifics on using many-to-many in a self-referential case.

declarative_many_to_many - Additional options when using Declarative.

association_pattern - an alternative to :paramref:`~.relationship.secondary` when composing association table relationships, allowing additional attributes to be specified on the association table.

composite_secondary_join - a lesser-used pattern which in some cases can enable complex `relationship()` SQL conditions to be used.

New in version 0.9.2: :paramref:`~.relationship.secondary` works more effectively when referring to a `Join` instance.

- **active_history=False** – When `True`, indicates that the "previous" value for a many-to-one reference should be loaded when replaced, if not already loaded. Normally, history tracking logic for simple many-to-ones only needs to be aware of the "new" value in order to perform a flush. This flag is available for applications that make use of `attributes.get_history()` which also need to know the "previous" value of the attribute.
- **backref** –

  indicates the string name of a property to be placed on the related mapper's class that will handle this relationship in the other direction. The other property will be created automatically when the mappers are configured. Can also be passed as a `backref()` object to control the configuration of the new relationship.

  See also

  relationships_backref - Introductory documentation and examples.

  :paramref:`~.relationship.back_populates` - alternative form of backref specification.

  `backref()` - allows control over `relationship()` configuration when using :paramref:`~.relationship.backref`.

- **back_populates** –

  Takes a string name and has the same meaning as :paramref:`~.relationship.backref`, except the complementing property is **not** created automatically, and instead must be configured explicitly on the other mapper. The complementing property should also indicate :paramref:`~.relationship.back_populates` to this relationship to ensure proper functioning.

  See also

  relationships_backref - Introductory documentation and examples.

:paramref:`~.relationship.backref` - alternative form of backref specification.

- **cascade** –
  a comma-separated list of cascade rules which determines how Session operations should be “cascaded” from parent to child. This defaults to `False`, which means the default cascade should be used - this default cascade is `"save-update, merge"`.

  The available cascades are `save-update`, `merge`, `expunge`, `delete`, `delete-orphan`, and `refresh-expire`. An additional option, `all` indicates shorthand for `"save-update, merge, refresh-expire, expunge, delete"`, and is often used as in `"all, delete-orphan"` to indicate that related objects should follow along with the parent object in all cases, and be deleted when de-associated.

  See also

  unitofwork_cascades - Full detail on each of the available cascade options.

  tutorial_delete_cascade - Tutorial example describing a delete cascade.

- **cascade_backrefs=True** –
  a boolean value indicating if the `save-update` cascade should operate along an assignment event intercepted by a backref. When set to `False`, the attribute managed by this relationship will not cascade an incoming transient object into the session of a persistent parent, if the event is received via backref.

  See also

  backref_cascade - Full discussion and examples on how the :paramref:`~.relationship.cascade_backrefs` option is used.

- **collection_class** –
  a class or callable that returns a new list-holding object. will be used in place of a plain list for storing elements.

  See also

  custom_collections - Introductory documentation and examples.

- **comparator_factory** –
  a class which extends `RelationshipProperty.Comparator` which provides

custom SQL clause generation for comparison operations.

See also

`PropComparator` - some detail on redefining comparators at this level.

custom_comparators - Brief intro to this feature.

- **distinct_target_key=None** –
  Indicate if a “subquery” eager load should apply the DISTINCT keyword to the innermost SELECT statement. When left as `None`, the DISTINCT keyword will be applied in those cases when the target columns do not comprise the full primary key of the target table. When set to `True`, the DISTINCT keyword is applied to the innermost SELECT unconditionally.

  It may be desirable to set this flag to False when the DISTINCT is reducing performance of the innermost subquery beyond that of what duplicate innermost rows may be causing.

  New in version 0.8.3: - :paramref:`~.relationship.distinct_target_key` allows the subquery eager loader to apply a DISTINCT modifier to the innermost SELECT.

  Changed in version 0.9.0: - :paramref:`~.relationship.distinct_target_key` now defaults to `None`, so that the feature enables itself automatically for those cases where the innermost query targets a non-unique key.

  See also

  loading_toplevel - includes an introduction to subquery eager loading.

- **doc** – docstring which will be applied to the resulting descriptor.
- **extension** –
  an `AttributeExtension` instance, or list of extensions, which will be prepended to the list of attribute listeners for the resulting descriptor placed on the class.

  Deprecated since version 0.7: Please see `AttributeEvents`.

- **foreign_keys** –
  a list of columns which are to be used as “foreign key” columns, or columns which refer to the value in a remote column, within the context of this `relationship()` object’s :paramref:`~.relationship.primaryjoin` condition. That is, if the :paramref:`~.relationship.primaryjoin` condition of this `relationship()` is `a.id == b.a_id`, and the values in `b.a_id` are required to be present in `a.id`, then the “foreign key” column of this `relationship()` is `b.a_id`.

  In normal cases, the :paramref:`~.relationship.foreign_keys` parameter is **not required.** `relationship()` will automatically determine which columns

in the :paramref:`~.relationship.primaryjoin` condtition are to be considered “foreign key” columns based on those `Column` objects that specify `ForeignKey`, or are otherwise listed as referencing columns in a `ForeignKeyConstraint` construct. :paramref:`~.relationship.foreign_keys` is only needed when:

- There is more than one way to construct a join from the local table to the remote table, as there are multiple foreign key references present. Setting `foreign_keys` will limit the `relationship()` to consider just those columns specified here as “foreign”.
  Changed in version 0.8: A multiple-foreign key join ambiguity can be resolved by setting the :paramref:`~.relationship.foreign_keys` parameter alone, without the need to explicitly set :paramref:`~.relationship.primaryjoin` as well.
- The `Table` being mapped does not actually have `ForeignKey` or `ForeignKeyConstraint` constructs present, often because the table was reflected from a database that does not support foreign key reflection (MySQL MyISAM).
- The :paramref:`~.relationship.primaryjoin` argument is used to construct a non-standard join condition, which makes use of columns or expressions that do not normally refer to their “parent” column, such as a join condition expressed by a complex comparison using a SQL function.

The `relationship()` construct will raise informative error messages that suggest the use of the :paramref:`~.relationship.foreign_keys` parameter when presented with an ambiguous condition. In typical cases, if `relationship()` doesn’t raise any exceptions, the :paramref:`~.relationship.foreign_keys` parameter is usually not needed.

:paramref:`~.relationship.foreign_keys` may also be passed as a callable function which is evaluated at mapper initialization time, and may be passed as a Python-evaluable string when using Declarative.

See also

relationship_foreign_keys

relationship_custom_foreign

`foreign()` - allows direct annotation of the “foreign” columns within a :paramref:`~.relationship.primaryjoin` condition.

New in version 0.8: The `foreign()` annotation can also be applied directly to the :paramref:`~.relationship.primaryjoin` expression, which is an alternate, more specific system of describing which columns in a particular :paramref:`~.relationship.primaryjoin` should be considered “foreign”.

- **info** –

Optional data dictionary which will be populated into the `MapperProperty.info` attribute of this object.

New in version 0.8.

- **innerjoin=False** –
  when `True`, joined eager loads will use an inner join to join against related tables instead of an outer join. The purpose of this option is generally one of performance, as inner joins generally perform better than outer joins.

  This flag can be set to `True` when the relationship references an object via many-to-one using local foreign keys that are not nullable, or when the reference is one-to-one or a collection that is guaranteed to have one or at least one entry.

  If the joined-eager load is chained onto an existing LEFT OUTER JOIN, `innerjoin=True` will be bypassed and the join will continue to chain as LEFT OUTER JOIN so that the results don't change. As an alternative, specify the value `"nested"`. This will instead nest the join on the right side, e.g. using the form "a LEFT OUTER JOIN (b JOIN c)".

  New in version 0.9.4: Added `innerjoin="nested"` option to support nesting of eager "inner" joins.

  See also

  what_kind_of_loading - Discussion of some details of various loader options.

  :paramref:`.joinedload.innerjoin` - loader option version

- **join_depth** –
  when non-`None`, an integer value indicating how many levels deep "eager" loaders should join on a self-referring or cyclical relationship. The number counts how many times the same Mapper shall be present in the loading condition along a particular join branch. When left at its default of `None`, eager loaders will stop chaining when they encounter a the same target mapper which is already higher up in the chain. This option applies both to joined- and subquery- eager loaders.

  See also

  self_referential_eager_loading - Introductory documentation and examples.

- **lazy='select'** –
  specifies how the related items should be loaded. Default value is `select`. Values include:

  - `select` - items should be loaded lazily when the property is first accessed,

using a separate SELECT statement, or identity map fetch for simple many-to-one references.

- `immediate` - items should be loaded as the parents are loaded, using a separate SELECT statement, or identity map fetch for simple many-to-one references.
- `joined` - items should be loaded “eagerly” in the same query as that of the parent, using a JOIN or LEFT OUTER JOIN. Whether the join is “outer” or not is determined by the :paramref:`~.relationship.innerjoin` parameter.
- `subquery` - items should be loaded “eagerly” as the parents are loaded, using one additional SQL statement, which issues a JOIN to a subquery of the original statement, for each collection requested.
- `noload` - no loading should occur at any time. This is to support “write-only” attributes, or attributes which are populated in some manner specific to the application.
- `dynamic` - the attribute will return a pre-configured `Query` object for all read operations, onto which further filtering operations can be applied before iterating the results. See the section *dynamic_relationship* for more details.
- True - a synonym for ‘select’
- False - a synonym for ‘joined’
- None - a synonym for ‘noload’

See also

`/orm/loading` - Full documentation on relationship loader configuration.

dynamic_relationship - detail on the `dynamic` option.

- **load_on_pending=False** –
  Indicates loading behavior for transient or pending parent objects.

  When set to `True`, causes the lazy-loader to issue a query for a parent object that is not persistent, meaning it has never been flushed. This may take effect for a pending object when autoflush is disabled, or for a transient object that has been “attached” to a `Session` but is not part of its pending collection.

  The :paramref:`~.relationship.load_on_pending` flag does not improve behavior when the ORM is used normally - object references should be constructed at the object level, not at the foreign key level, so that they are present in an ordinary way before a flush proceeds. This flag is not not intended for general use.

See also

`Session.enable_relationship_loading()` - this method establishes “load on pending” behavior for the whole object, and also allows loading on

objects that remain transient or detached.

- **order_by** –
  indicates the ordering that should be applied when loading these items. :paramref:`~.relationship.order_by` is expected to refer to one of the `Column` objects to which the target class is mapped, or the attribute itself bound to the target class which refers to the column.

  :paramref:`~.relationship.order_by` may also be passed as a callable function which is evaluated at mapper initialization time, and may be passed as a Python-evaluable string when using Declarative.

- **passive_deletes=False** –
  Indicates loading behavior during delete operations.

  A value of True indicates that unloaded child items should not be loaded during a delete operation on the parent. Normally, when a parent item is deleted, all child items are loaded so that they can either be marked as deleted, or have their foreign key to the parent set to NULL. Marking this flag as True usually implies an ON DELETE <CASCADE|SET NULL> rule is in place which will handle updating/deleting child rows on the database side.

  Additionally, setting the flag to the string value 'all' will disable the "nulling out" of the child foreign keys, when there is no delete or delete-orphan cascade enabled. This is typically used when a triggering or error raise scenario is in place on the database side. Note that the foreign key attributes on in-session child objects will not be changed after a flush occurs so this is a very special use-case setting.

  See also

  passive_deletes - Introductory documentation and examples.

- **passive_updates=True** –
  Indicates loading and INSERT/UPDATE/DELETE behavior when the source of a foreign key value changes (i.e. an "on update" cascade), which are typically the primary key columns of the source row.

  When True, it is assumed that ON UPDATE CASCADE is configured on the foreign key in the database, and that the database will handle propagation of an UPDATE from a source column to dependent rows. Note that with databases which enforce referential integrity (i.e. PostgreSQL, MySQL with InnoDB tables), ON UPDATE CASCADE is required for this operation. The relationship() will update the value of the attribute on related items which are locally present in the session during a flush.

  When False, it is assumed that the database does not enforce referential integrity and will not be issuing its own CASCADE operation for an update. The relationship() will issue the appropriate UPDATE statements

to the database in response to the change of a referenced key, and items locally present in the session during a flush will also be refreshed.

This flag should probably be set to False if primary key changes are expected and the database in use doesn't support CASCADE (i.e. SQLite, MySQL MyISAM tables).

See also

passive_updates - Introductory documentation and examples.

:paramref:`.mapper.passive_updates` - a similar flag which takes effect for joined-table inheritance mappings.

- **post_update** –
  this indicates that the relationship should be handled by a second UPDATE statement after an INSERT or before a DELETE. Currently, it also will issue an UPDATE after the instance was UPDATEd as well, although this technically should be improved. This flag is used to handle saving bi-directional dependencies between two individual rows (i.e. each row references the other), where it would otherwise be impossible to INSERT or DELETE both rows fully since one row exists before the other. Use this flag when a particular mapping arrangement will incur two rows that are dependent on each other, such as a table that has a one-to-many relationship to a set of child rows, and also has a column that references a single child row within that list (i.e. both tables contain a foreign key to each other). If a flush operation returns an error that a "cyclical dependency" was detected, this is a cue that you might want to use :paramref:`~.relationship.post_update` to "break" the cycle.

  See also

  post_update - Introductory documentation and examples.

- **primaryjoin** –
  a SQL expression that will be used as the primary join of this child object against the parent object, or in a many-to-many relationship the join of the primary object to the association table. By default, this value is computed based on the foreign key relationships of the parent and child tables (or association table).

  :paramref:`~.relationship.primaryjoin` may also be passed as a callable function which is evaluated at mapper initialization time, and may be passed as a Python-evaluable string when using Declarative.

  See also

  relationship_primaryjoin

- **remote_side** –
  used for self-referential relationships, indicates the column or list of columns that form the “remote side” of the relationship.

  :paramref:`.relationship.remote_side` may also be passed as a callable function which is evaluated at mapper initialization time, and may be passed as a Python-evaluable string when using Declarative.

  Changed in version 0.8: The `remote()` annotation can also be applied directly to the `primaryjoin` expression, which is an alternate, more specific system of describing which columns in a particular `primaryjoin` should be considered “remote”.

  See also

  self_referential - in-depth explanation of how :paramref:`~.relationship.remote_side` is used to configure self-referential relationships.

  `remote()` - an annotation function that accomplishes the same purpose as :paramref:`~.relationship.remote_side`, typically when a custom :paramref:`~.relationship.primaryjoin` condition is used.

- **query_class** –
  a `Query` subclass that will be used as the base of the “appender query” returned by a “dynamic” relationship, that is, a relationship that specifies `lazy="dynamic"` or was otherwise constructed using the `orm.dynamic_loader()` function.

  See also

  dynamic_relationship - Introduction to “dynamic” relationship loaders.

- **secondaryjoin** –
  a SQL expression that will be used as the join of an association table to the child object. By default, this value is computed based on the foreign key relationships of the association and child tables.

  :paramref:`~.relationship.secondaryjoin` may also be passed as a callable function which is evaluated at mapper initialization time, and may be passed as a Python-evaluable string when using Declarative.

  See also

  relationship_primaryjoin

- **single_parent** –
  when True, installs a validator which will prevent objects from being

associated with more than one parent at a time. This is used for many-to-one or many-to-many relationships that should be treated either as one-to-one or one-to-many. Its usage is optional, except for `relationship()` constructs which are many-to-one or many-to-many and also specify the `delete-orphan` cascade option. The `relationship()` construct itself will raise an error instructing when this option is required.

See also

unitofwork_cascades - includes detail on when the :paramref:`~.relationship.single_parent` flag may be appropriate.

- **uselist** –
  a boolean that indicates if this property should be loaded as a list or a scalar. In most cases, this value is determined automatically by `relationship()` at mapper configuration time, based on the type and direction of the relationship - one to many forms a list, many to one forms a scalar, many to many is a list. If a scalar is desired where normally a list would be present, such as a bi-directional one-to-one relationship, set :paramref:`~.relationship.uselist` to False.

  The :paramref:`~.relationship.uselist` flag is also available on an existing `relationship()` construct as a read-only attribute, which can be used to determine if this `relationship()` deals with collections or scalar attributes:

```
>>> User.addresses.property.uselist
True
```

See also

relationships_one_to_one - Introduction to the “one to one” relationship pattern, which is typically when the :paramref:`~.relationship.uselist` flag is needed.

- **viewonly=False** – when set to True, the relationship is used only for loading objects, and not for any persistence operation. A `relationship()` which specifies :paramref:`~.relationship.viewonly` can work with a wider range of SQL operations within the :paramref:`~.relationship.primaryjoin` condition, including operations that feature the use of a variety of comparison operators as well as SQL functions such as `cast()`. The :paramref:`~.relationship.viewonly` flag is also of general use when defining any kind of `relationship()` that doesn’t represent the full set of related objects, to prevent modifications of the collection from resulting in persistence operations.

tasks Package

`tasks` Package

`copy` Module

okcupyd.tasks.copy.build_copy(*source_credentials_or_username*, *dest_credentials*) [source]

okcupyd.tasks.copy.build_copy_task(*method*) [source]

`db` Module

`site` Module

util Package

`util` Package

okcupyd.util.makelist(*value*) [source]

okcupyd.util.makelist_decorator(*function*) [source]

class **okcupyd.util.cached_property**(*func*) [source]

Bases: `object`

Descriptor that caches the result of the first call to resolve its contents.

bust_self(*obj*) [source]

Remove the value that is being stored on *obj* for this `cached_property` object.

Parameters: **obj** – The instance on which to bust the cache.

classmethod **bust_caches**(*obj*, *excludes=()*) [source]

Bust the cache for all `cached_property` objects on *obj*

Parameters: **obj** – The instance on which to bust the caches.

classmethod **get_cached_properties**(*obj*) [source]

class **okcupyd.util.CallableMap**(*func_value_pairs=()*) [source]

Bases: `object`

add(*func*, *value*) [source]

class **okcupyd.util.REMap**(*re_value_pairs=()*, *default=<object object at 0x7ff8d7089c40>*) [source]

Bases: `object`

A mapping object that matches regular expressions to values.

NO_DEFAULT= *<object object at 0x7ff8d7089c40>*

classmethod **from_string_pairs**(*string_value_pairs*, ***kwargs*) [source]

Build an `REMap` from str, value pairs by applying *re.compile* to each string and calling the __init__ of `REMap`

add(*re*, *value*) [source]

pattern_to_value [source]

values() [source]

class **okcupyd.util.GetAttrGetItem** [source]

Bases: `type`

okcupyd.util.IndexedREMap(**re_strings*, ***kwargs*) [source]

Build a `REMap` from the provided regular expression string. Each string will be associated with the index corresponding to its position in the argument list.

Parameters:
- **re_strings** – The re_strings that will serve as keys in the map.
- **default** – The value to return if none of the regular expressions match
- **offset** – The offset at which to start indexing for regular expressions defaults to 1.

`compose` Module

okcupyd.util.compose.make_single_arity(*function*) [source]

okcupyd.util.compose.force_args_return(*function*) [source]

okcupyd.util.compose.tee(**functions*) [source]

`currying` Module

okcupyd.util.currying.curry [source]

Curry a function or method.

Applying `curry` to a function creates a callable with the same functionality that can be invoked with an incomplete argument list to create a partial application of the original function.

```
@curry
def greater_than(x, y):
   return x > y

>>> less_than_40 = greater_than(40)
>>> less_than_40(39)
True
>>> less_than_40(50)
False
```

`curry` allows functions to be partially invoked an arbitary number of times:

```
@curry
def add_5_things(a, b, c, d, e):
    return a + b + c + d + e

# All of the following invocations of add_5_things
>>> add_5_things(1)(1)(1)(1)(1)
5

one_left = add_5_things(1, 1)(3)(4) # A one place function that will
# add 1 + 1 + 3 + 4 = 9 to whatever is provided as its argument.

>>>> one_left(5)
14
>>> one_left(6)
15
```

A particular compelling use case for `curry` is the creation of decorators that take optional arguments:

```
@curry
def add_n(function, n=1):
    def wrapped(*args, **kwargs):
        return function(*args, **kwargs) + n
    return wrapped

@add_n(n=12)
def multiply_plus_twelve(x, y):
    return x * y

@add_n
def multiply_plus_one(x, y):
    return x * y

>>> multiply_plus_one(1, 1)
2
>>> multiply_plus_twelve(1, 1)
13
```

Notice that we were able to apply *add_n* regardless of whether or not an optional argument had been supplied earlier.

The version of curry that is available for import has been curried itself. That is, its constructor can be invoked partially:

```
@curry(evaluation_checker=lambda *args, **kwargs: len(args) > 2)
def args_taking_function(*args):
    return reduce(lambda x, y: x*y, args)

>>> args_taking_function(1, 2)
2
>>> args_taking_function(2)(3)
6
>>> args_taking_function(2, 2, 2, 2)
16
```

`fetchable` Module

Most of the collection objects that are returned from function invocations in the okcupyd library are instances of `Fetchable`. In most cases, it is fine to treat these objects as though they are lists because they can be iterated over, sliced and accessed by index, just like lists:

```
for question in user.profile.questions:
    print(question.answer.text)

a_random_question = user.profile.questions[2]
for question in questions[2:4]:
    print(question.answer_options[0])
```

However, in some cases, it is important to be aware of the subtle differences between `Fetchable` objects and python lists. `Fetchable` construct the elements that they "contain" lazily. In most of its uses in the okcupyd library, this means that http requests can be made to populate `Fetchable` instances as its elments are requested.

The `questions` `Fetchable` that is used in the example above fetches the pages that are used to construct its contents in batches of 10 questions. This means that the actual call to retrieve data is made when iteration starts. If you enable the request logger when you run this code snippet, you get output that illustrates this fact:

```
2014-10-29 04:25:04 Livien-MacbookAir requests.packages.urllib3.connectionpool[82461] DEBUG
"GET /profile/ShrewdDrew/questions?leanmode=1&low=11 HTTP/1.1" 200 None
 Yes
 Yes
 Kiss someone.
 Yes.
 Yes
 Sex.
 Both equally
 No, I wouldn't give it as a gift.
 Maybe, I want to know all the important stuff.
 Once or twice a week
 2014-10-29 04:25:04 Livien-MacbookAir requests.packages.urllib3.connectionpool[82461] DEBUG
"GET /profile/ShrewdDrew/questions?leanmode=1&low=21 HTTP/1.1" 200 None
 No.
 No
 No
 Yes
 Rarely / never
 Always.
 Discovering your shared interests
 The sun
 Acceptable.
 No.
```

Some fetchables will continue fetching content for quite a long time. The search fetchable, for example, will fetch content until okcupid runs out of search results. As such, things like:

```
for profile in user.search():
    profile.message("hey!")
```

should be avoided, as they are likely to generate a massive number of requests to okcupid.com.

Another subtlety of the `Fetchable` class is that its instances cache its contained results. This means that the second iteration over `okcupyd.profile.Profile.questions` in the example below does not result in any http requests:

```
for question in user.profile.questions:
    print(question.text)

for question in user.profile.questions:
    print(question.answer)
```

It is important to understand that this means that the contents of a `Fetchable` are not guarenteed to be in sync with okcupid.com the second time they are requested. Calling `refresh()` will cause the `Fetchable` to request new data from okcupid.com when its contents are requested. The code snippet that follows prints out all the questions that the logged in user has answered roughly once per hour, including ones that are answered while the program is running.

```
import time

while True:
    for question in user.profile.questions:
        print(question.text)
    user.profile.questions.refresh()
    time.sleep(3600)
```

Without the call to *user.profile.questions.refresh()*, this program would never update the user.profile.questions instance, and thus what would be printed to the screen with each iteration of the for loop.

class **okcupyd.util.fetchable.Fetchable**(*fetcher*, ***kwargs*) [source]

Bases: `object`

List-like container object that lazily loads its contained items.

classmethod **fetch_marshall**(*fetcher*, *processor*) [source]

refresh(*nice_repr=True*, ***kwargs*) [source]

Parameters:
- **nice_repr** (*bool*) – Append the repr of a list containing the items that have been fetched to this point by the fetcher.
- **kwargs** – kwargs that should be passed to the fetcher when its fetch method is called. These are merged with the values provided to the constructor, with the ones provided here taking precedence if there is a conflict.

class **okcupyd.util.fetchable.FetchMarshall**(*fetcher*, *processor*, *terminator=None*, *start_at=1*) [source]

Bases: `object`

static **simple_decider**(*pos*, *last*, *text_response*) [source]

fetch(*start_at=None*) [source]

class **okcupyd.util.fetchable.SimpleProcessor**(*session*, *object_factory*, *element_xpath*) [source]

Bases: `object`

Applies object_factory to each element found with element_xpath

Accepts session merely to be consistent with the FetchMarshall interface.

process(*text_response*) [source]

class **okcupyd.util.fetchable.PaginationProcessor**(*object_factory*, *element_xpb*, *current_page_xpb*, *total_page_xpb*) [source]

Bases: `object`

process(*text_response*) [source]

class **okcupyd.util.fetchable.GETFetcher**(*session*, *path*, *query_param_builder=<function <lambda> at 0x7ff8d5fafa28>*) [source]

Bases: `object`

fetch(**args*, ***kwargs*) [source]

`misc` Module

okcupyd.util.misc.enable_logger(*log_name*, *level=10*) [source]

okcupyd.util.misc.get_credentials() [source]

okcupyd.util.misc.add_command_line_options(*add_argument*, *use_short_options=True*) [source]

Parameters:
- **add_argument** – The add_argument method of an ArgParser.
- **use_short_options** – Whether or not to add short options.

okcupyd.util.misc.handle_command_line_options(*args*) [source]

Parameters: **args** – The args returned from an ArgParser

okcupyd.util.misc.update_settings_with_module(*module_name*) [source]

okcupyd.util.misc.save_file(*filename*, *data*) [source]

okcupyd.util.misc.find_all(*a_str*, *sub*) [source]

okcupyd.util.misc.replace_all_case_insensitive(*a_str*, *sub*, *replacement*) [source]

Indices and tables

- *Index*
- *Module Index*
- *Search Page*