Pychievements

Release 0.1.2

Apr 17, 2017

Contents

Pychievements is a framework for creating and tracking achievements within a Python application. It includes functions specifically for creating commandline applications, though it is flexible enough to be used for any application such as web applications.

See the examples to get a good feel for what Pychievements offers. Source can be found on github.

CHAPTER 1

Getting Started

Getting Started will guide you through creating your first achievements an introducing you to the pieces of the system. From there, visit the *Reference* for in-depth information.

Installing Pychievements

Pychievements can be installed with `pip`:

```
$ pip install pychievements
```

Introduction to Pychievments

Pychievements has a number of modules that you'll need to be at least familiar with:

tracker The default achievement tracker for pychievements is instantiated at import and is used to track all registered achievements for specific *tracked_ids*.

Achievement Base Achievment class.

icons.Icon Base Icon class. Icons are used to know what to display for a given goal within an achievement and have two states, achieved and unachieved.

backends.AchievementBackend Pychievements have pluggable backends for storing tracked achievement data. The default `AchievementBackend` simply keeps everyting in memory, meaning it will be lost when the application is closed. The backend the tracker is using can be updated with the `set_backend` method.

signals You can register functions as callbacks that can *recieve* Pychievement signals. Signals can be generated when a level is changed, when a new goal is reached, or when all goals have been achieved for a given achievement.

Achievements

At the core of Pychievements is the `Achievement` class. It is used to define goals that are obtained at specified *levels*. Levels are simply an integer. At a minimum, achievements must have the following attributes defined:

- name : Display name of your achievement
- category : Defaults to "achievements"
- goals : A tuple of *goals*. A *goal* is a dictionary with the following keys: *["level", "name", "icon", "description"]*

An example Achievement class:

```
class MyAchievement(Achievement):
    name = "My Achievement"
    category = "achievements"
    keywords = ("my", "achievement")
    goals = (
        {"level": 10, "name": "Level 1", "icon": icons.star, "description": "Level One
↪" },
        {"level": 20, "name": "Level 2", "icon": icons.star, "description": "Level Two
↪" },
        {"level": 30, "name": "Level 3", "icon": icons.star, "description": "Level␣
↪Three" },
    )
```

An achievements current level for an id can tracked with either the `increment` or `evaluate` functions, which the achievment can override to provide custom level manipulation.

The Tracker

A singleton `tracker` is created on import that is available as `pychievements.tracker`. The tracker provides an interface for interacting with registered achievements for a given `tracked_id`. The tracker lets you:

- increment level of an achievement for a tracked_id
- evaluate level of an achievement for a tracked_id
- query all achievements by category or keywords
- query all achieved goals of an achievement for a tracked_id
- query all unachieved goals of an achievement for a tracked_id
- query the current goal being worked towards of an achievement for a tracked_id

The tracker works with the configured backend to store and retrieve all of the tracked levels.

Icons

Icons are simple classes that provide the "icon" (or what is displayed) for an achievment goal. It must define what to display for when the goal has been achieved (`achieved`) or not (`unachieved`)

Examples

The easiest way to get started to check out the examples in the examples folder in the the repository. Then check out the *Reference* for more information.

CHAPTER 2

Reference

Achievements

class pychievements.achievements.**Achievement**(*current=0*)

Base Achievement class.

An achievement primarily consists of 'goals', being levels that can be reached. Instances of Achievements are used to track progress, and the current level for individual IDs. For this, an Achievement implements a number of functions to interact with the current level. Achievements can also have a `category` (string) and `keywords` (tuple of strings) that can be used to filter Achievements.

Goals are defined as a tuple of tuples with the format:

```
goals = (
    {'level': 10, 'name': 'Level 1', 'icon': icons.star, 'description': 'Level One
↪'},
    {'level': 20, 'name': 'Level 2', 'icon': icons.star, 'description': 'Level Two
↪'},
    {'level': 30, 'name': 'Level 3', 'icon': icons.star, 'description': 'Level␣
↪Three'},
)
```

Arguments:

level A positive integer that must be reached (greater than or equal) to be considered 'met'

name A short name for the level

icon The `Icon` to represent the level before it has been achieved. This must be an *pychievements.icons.Icon* class.

Note: There are simple ASCII icons available from *pychievements.icons*

---

description A longer description of the level.

Achievements can be updated in two ways: `increment` and `evaluate`. Increment increments the current level given an optional set of arguments, where evaluate performs a custom evaluation a sets the current level based on that evaluation.

Increment is best used when the application is aware of achievement tracking, and calls to increment can be placed throughout the application.

Evaluate is best used when actions may happen externally, and cannot be tracked using repeated calls to increment. Evaluate will also return the list of achieved goals after it has performed its evaluation.

An Achievement can be initialized with a `current` level, for example when restoring for a saved state.

achieved
Returns a list of achieved goals

current
Returns the current level being achieved (meaning haven't achieved yet) as a tuple:

:: (current_level, (required_level, name, icon, description))

If all achievements have been achieved, the current level is returned with a None:

:: (current_level, None)

evaluate (**args*, ***kwargs*)
Performs a custom evaluation to set the current level of an achievement. Returns a list of achieved goals after the level is determined.

increment (*amount=1*, **args*, ***kwargs*)
Increases the current level. Achievements can redefine this function to take options to increase the level based on given arguments. By default, this will simply increment the current count by `amount` (which defaults to 1).

set_level (*level*)
Overrides the current level with the given level

unachieved
Returns a list of goals that have not been met yet

Icons

pychievments.icons includes the `Icon` class as well as a number of pre-defined icons useful for CLI applications.

- unicodeCheck
- unicdeCheckBox
- star

class `pychievements.icons.`**Icon** (*unachieved=''*, *achieved=''*)
Simple class to represent an `Icon` for an achievement. It provides to functions, `achieved`, and `unachieved`, which will return the displayable icon for the appropriate state.

The base Icon class can be used without modification to create simple text Icons, e.g.:

```
star = Icon(unachieved=' No ', achieved=' Yes ')
```

achieved (*tracked_id=None*, *achievement=None*)
Returns the achieved icon

unachieved (*tracked_id=None*, *achievement=None*)
Returns the unachieved icon

Trackers

class pychievements.trackers.**AchievementTracker**(*backend=None*)

AchievementTracker tracks achievements and current levels for `tracked_id` using a configured achievement backend.

A default instance of Achievement tracker is created as a singleton when pycheivements is imported as `pychievements.tracker`. Most often, this is what you will want to use.

Arguments:

backend: The backend to use for storing/retrieving achievement data. If `None`, the default `AchievementBackend` will be used, which stores all data in memory.

Note: The backend the tracker is using can be updated at any time using the *`set_backend()`* function.

---

achieved(*tracked_id*, *achievement*)

Returns `achieved` for a given tracked_id. See :ref:`Achievement`

achievement_for_id(*tracked_id*, *achievement*)

Returns `Achievement` for a given `tracked_id`. Achievement can be an `Achievement` class or a string of the name of an achievement class that has been registered with this tracker.

Raises NotRegistered if the given achievement is not registered with the tracker.

If `tracked_id` has not been tracked yet by this tracker, it will be created.

achievements(*category=None*, *keywords=[]*)

Returns all registered achievements.

Arguments:

category Filters returned achievements by category. This is a strict string match.

keywords Filters returned achievements by keywords. Returned achievements will match all given keywords

achievements_for_id(*tracked_id*, *category=None*, *keywords=[]*)

Returns all of the achievements for tracked_id that match the given category and keywords

current(*tracked_id*, *achievement*)

Returns `current` for a given tracked_id. See :ref:`Achievement`

evaluate(*tracked_id*, *achievement*, **args*, ***kwargs*)

Evaluates an achievement for a given `tracked_id`. Achievement can be an `Achievement` class or a string of the name of an achievement class that has been registered with this tracker.

Raises NotRegistered if the given achievement is not registered with the tracker.

If `tracked_id` has not been tracked yet by this tracker, it will be created before evaluating.

Returns list of achieved goals for the given achievement after evaluation

get_tracked_ids()

Returns all tracked ids

increment(*tracked_id*, *achievement*, *amount=1*, **args*, ***kwargs*)

Increments an achievement for a given `tracked_id`. Achievement can be an `Achievement` class or a string of the name of an achievement class that has been registered with this tracker.

Raises NotRegistered if the given achievement is not registered with the tracker.

If `tracked_id` has not been tracked yet by this tracker, it will be created before incrementing.

Returns an list of achieved goals if a new goal was reached, or False

is_registered(*achievement*)
Check if an achievement is registered with this *AchievementTracker*

register(*achievement_or_iterable*, ***options*)
Registers the given achievement(s) to be tracked.

remove_id(*tracked_id*)
Remove all tracked information for tracked_id

set_backend(*backend*)
Configures a new backend for storing achievement data.

set_level(*tracked_id*, *achievement*, *level*)
Returns `set_level` for a given tracked_id. See :ref:`Achievement`

unachieved(*tracked_id*, *achievement*)
Returns `unachieved` for a given tracked_id. See :ref:`Achievement`

unregister(*achievement_or_iterable*)
Un-registers the given achievement(s).

If an achievement isn't already registered, this will raise NotRegistered.

Signals

class `pychievements.signals.`**Signal**
Base class for all signals

Internal attributes:

receivers { receiverkey(id): receiver }

connect(*receiver*, *sender=None*, *dispatch_uid=None*)
Connect receiver to sender for signal.

Arguments:

receiver A function or an instance method which is to recieve signals.

sender The sender to which the receiver should respond. Must be None to recieve events from any sender.

dispatch_uid An identifier used to uniquely identify a particular instance of a receiver. This will usually be a string, though it may be anything hashable.

disconnect(*receiver=None*, *sender=None*, *dispatch_uid=None*)
Disconnect receiver from sender for signal.

Arguments:

receiver The registered receiver to disconnect. May be none if dispatch_uid is specified.

sender The registered sender to disconnect

dispatch_uid the unique identifier of the receiver to disconnect

send(*sender*, ***named*)
Send signal from sender to all connected receivers.

If any receiver raises an error, the error propagates back through send, terminating the dispatch loop, so it is quite possible to not have all receivers called if a raises an error.

Arguments:

sender The sender of the signal Either a specific object or None.

named Named arguments which will be passed to receivers.

Returns a list of tuple pairs [(receiver, response), ...].

send_robust(*sender*, ***named*)

Send signal from sender to all connected receivers catching errors.

Arguments:

sender The sender of the signal. Can be any python object (normally one registered with a connect if you actually want something to occur).

named Named arguments which will be passed to receivers. These arguments must be a subset of the argument names defined in providing_args.

Return a list of tuple pairs [(receiver, response), ...].

If any receiver raises an error (specifically any subclass of Exception), the error instance is returned as the result for that receiver. The traceback is always attached to the error at `__traceback__`.

`pychievements.signals.`**receiver**(*signal*, ***kwargs*)

A decorator for connecting receivers to signals. Used by passing in the signal (or list of signals) and keyword arguments to connect:

```
@receiver(goal_achieved)
def signal_receiver(sender, **kwargs):
    ...

@receiver([goal_achieved, level_increased], sender=tracker)
def signals_receiver(sender, **kwargs):
    ...
```

Backends

class `pychievements.backends.`**AchievementBackend**

Achievement backends implement the getting/setting/updating of achievements for `tracked_id`. Achievements in the system are tracked for a specific, unique ID, `tracked_id`.

AchievementBackend is the most basic implementation of an AchievementBackend, storing all tracked information in memory and never persisting it. All of the functions of an AchievementBackend work to retrieve an `Achievement` instance for a given `tracked_id`, and run the appropriate function on it, storing the results. In the least, storing results for a specific achievement, for a specific `target_id` should include the `target_id`, the `Achievement` class name (`Achievement.__name__`), and the current level (`Achievement.current`)

Note: AchievementBackend is NOT thread safe

---

achievement_for_id(*tracked_id*, *achievement*)

Retrieves the current `Achievement` for the given `tracked_id`. If the given `tracked_id` does not exist yet, it should be created. Also, if the given `tracked_id` hasn't tracked the given `Achievement` yet, a new instance of the `Achievement` should be created for the given `tracked_id`

achievements_for_id(*tracked_id*, *achievements*)
Returns the current achievement for each achievement in `achievements` for the given tracked_id

remove_id(*tracked_id*)
Removes *tracked_id* from the backend

set_level_for_id(*tracked_id*, *achievement*, *level*)
Set the `level` for an `Achievement` for the given `tracked_id`

class `pychievements.backends.`**SQLiteAchievementBackend**(*dbfile*)
Stores achievement data in a SQLite database.

Arguments:

dbfile The full path and file name to store the SQLite database

To use, create the backend and then use the `set_backend()` method of the tracker.

```
mybackend = SQLiteAchievementBackend('/some/db.file')
tracker.set_backend(mybackend)
```

CLI

`pychievements.cli.`**print_goal**(*goal*, *achieved=False*, *level=None*, *indent=2*)
Print a goals description with its icon. Achieved (True/False) will choose the correct icon from the goal. If a level is specified, a tracker line will be added under the icon showing the current level out of the required level for the goal. If level is > the required level, achieved will be set to true.

`pychievements.cli.`**print_goals**(*achievement_or_iter*, *indent=2*)
Displays all of the available goals registered for the given achievement(s)

`pychievements.cli.`**print_goals_for_tracked**(*tracked_id*, *achievement_or_iter=None*, *achieved=True*, *unachieved=False*, *only_current=False*, *level=False*, *category=None*, *keywords=[]*, *indent=2*, *tracker=None*)
Prints goals for a specific `tracked_id` from as tracked by a `tracker`. By default, this will print out all achieved goals for every achievement in the `tracker`.

Arguments:

achievment_or_iter If `None`, this will print goals for all achievements registered with the `tracker`. Otherwise an `Achievement` or list of achievements can be given to show goals for.

achieved If True, prints out goals that have allready been achieved.

unachieved If True, prints out goals that have not been achieved.

only_current If True, only prints the goal currently being worked on (next to be achieved). This will override the `achieved` and `unachieved` options.

category Category to filter achievements from the tracker.

keywords Keywords to filter achievements from the tracker.

level If True, show the current level with the achievements

tracker The tracker to use for getting information about achievements and `tracked_id`. If `tracker` is `None`, this will default to using the default tracker.

CHAPTER 3

Contribute

If you'd like to contribute, simply fork the repository, commit your changes to the **master** branch (or branch off of it), and send a pull request. Make sure you add yourself to AUTHORS.

CHAPTER 4

License

Pychievements is license under the MIT license. You can find it in github, LICENSE

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

S

U