
PythonTop40 Documentation

Release 0.1.6

Danny Goodall

January 03, 2015

1	Contents	1
1.1	Installation	1
1.2	Exploring the Demo Code	1
1.3	PythonTop40 top40	10
1.4	PythonTop40 errors	15
1.5	Change Log for PythonTop40	16
2	PythonTop40	21
2.1	Usage	21
3	Features	23
4	Installation	25
5	Documentation	27
6	API - Application Programming Interface	29
6.1	Tests	29
6.2	Changes	29
6.3	Indices and tables	29
	Python Module Index	31

1.1 Installation

PythonTop40 can be found on the Python Package Index [PyPi here](#). It can be installed using `pip`, like so.

```
pip install pythontop40
```

1.2 Exploring the Demo Code

1.2.1 Our example program

Let's look at an example program, and examine in detail what it is doing and how it works.

```
from pythontop40 import Top40

top40 = Top40()

albums = top40.albums

for album in albums:
    print(
        album.position,
        album.title,
        "BY",
        album.artist,
        album.previousPosition,
        album.numWeeks
    )
```

1.2.2 Importing the PythonTop40 module

The first line in our program

```
from pythontop40 import Top40

top40 = Top40()

albums = top40.albums
```

```
for album in albums:
    print(
        album.position,
        album.title,
        "BY",
        album.artist,
        album.previousPosition,
        album.numWeeks
    )
```

uses the Python `import` command to bring the `Top40` class from the `top40` module into our code.

This `import` command means that our program can now use the `Top40` class, to get the list of Top 40 singles and albums. The `import` command is how we tell Python that we want to use a feature that isn't included in the Python standard library.

1.2.3 Creating a Top40 instance

The next line in our program creates a variable called `top40` which becomes the way we will talk to the remote server where the lists of Top 40 singles and albums information is held.

```
from pythontop40 import Top40
```

```
top40 = Top40()
```

```
albums = top40.albums
```

```
for album in albums:
    print(
        album.position,
        album.title,
        "BY",
        album.artist,
        album.previousPosition,
        album.numWeeks
    )
```

Behind the Scenes

Technically speaking this code creates an *instance* of the `Top40` class, and behind the scenes it is this that manages the communication with the remote server that contains the list of singles and albums.

We don't really need to worry about that, as all of this complexity is hidden from us. Instead we simply interact with the data and capabilities that the `top40` variable provides us.

We can think of the `top40` variable as providing us with a number of ways to access the Top 40 charts for albums and singles.

`top40` does this through a number of *properties* that each returns different results to our program.

If we were to use the `top40.singles` property instead of the `top40.albums` property, then as you might expect our program would receive a python `list` of singles instead of a `list` of albums.

Other properties that we could use are `top40.singles_chart` and `top40.albums_chart` which both return a little bit more information about the chart itself - such as the `date` it was published and the date it was `retrieved` from the server.

1.2.4 Retrieving the Top40 albums

The following line of code creates a variable called `albums` and assigns to it the value returned from the `top40.albums` *property*.

```
from pythontop40 import Top40

top40 = Top40()

albums = top40.albums

for album in albums:
    print(
        album.position,
        album.title,
        "BY",
        album.artist,
        album.previousPosition,
        album.numWeeks
    )
```

When this piece of code is executed, behind the scenes our `top40` variable *magically* makes contact with a server over the Internet, asks it for the list of the Top 40 albums, and returns this list *list* of information to our `albums` variable.

1.2.5 The format of the returned data

If we could see the value returned to the `albums` variable in the above code, it would look *something* like this.

```
albums = [
    Entry(
        position = 1,
        artist = "One Direction"
        ...
    ),
    Entry(
        position = 2,
        artist = "Ed Sheeran"
        ...
    ),
    Entry(
        position = 3,
        artist = "Sam Smith"
        ...
    )
]
```

Note: The `...` in the above example shows that there are more pieces of information in the `Entry`, but these are not shown to make the example easier to understand.

The data is enclosed in `[]` square brackets, which tells us that we have a Python *list* of ‘*things*’. But what are the things in the list? Well, because we have a *list* of things, we can access the first (or 0th item) in the list by placing ‘`[0]`’ after the name of a *list*.

```
print(albums[0])
Entry(position = 1, artist = "One Direction"...)
```

Behind the Scenes

Whilst you will never have to do this yourself, an `Entry` instance is created by passing *named arguments* to the `Entry` class. If we were to manually create the `Entry` instance, it might look something like this.

```
entry = Entry(
    position = 3,
    previousPosition = 4,
    numWeeks = 26,
    artist = "Sam Smith",
    title = "In The Lonely Hour",
    Change(
        direction = "up",
        amount = 1,
        actual = 1
    )
)
```

If we then asked Python to print the `position` attribute of the `entry` variable, we would get the following result

```
print(entry.position)
3
```

Likewise if we wanted to see how many weeks this entry had been in the chart we could access it like this.

```
print(entry.numWeeks)
26
```

So you should be able to see that inside our `Entry` object, we have another object called `Change`. This means that to access the `Change` object that is inside the `Entry` object, we would do the following.

```
print(entry.change)
<Change (
  amount=1,
  actual=1,
  direction='up'
)>
```

And finally, to access the `direction` of the change since last week's chart, we can see that we would have to access the `direction` attribute of the `Change` object that is embedded in the `Entry` object. And to do that, we could type the following.

```
print(entry.change.direction)
up
```

1.2.6 Accessing the information within each chart entry

This tells us that we have a list of things of type `Entry`. There is one `Entry` for every album in our Top 40 chart. The example data above only shows the first 3 entries, but given that this is the Top 40 we are dealing with, we would expect to see 40 entries in our list.

Each entry is represented by a Python *object* called `Entry`. The `Entry` class has been created as part of the **Python-Top40** project to hold the details of albums or singles in the chart.

As you'd expect from looking at the example code, the `Entry` class can hold information about the `position` of this entry, the name of the `artist`, the `title` of the album or single.

In addition, the number of weeks the album or single has been in the chart is accessed via the `numWeeks` attribute and the position that the entry occupied last week can be found by using the `previousPosition` attribute.

So in our original example, the next part the code loops through each of the album entries in the chart using the `for` statement, and then inside the loop, the value of `album` is set to each of the `albums` in our list.

This means that we can use the `print()` function to print the position, title and artist of each of the albums in our chart.

```
from pythontop40 import Top40

top40 = Top40()

albums = top40.albums

for album in albums:
    print(
        album.position,
        album.title,
        "BY",
        album.artist,
        album.previousPosition,
        album.numWeeks
    )
```

1.2.7 Printing extra information about the chart entry

If we wanted to extend our demo program to print the number of weeks that the album had been in the chart, as well as the chart position it occupied in the previous week's chart, we could do this by accessing the `numWeeks` and `previousPosition` attributes respectively.

The following code would achieve that.

```
from pythontop40 import Top40

top40 = Top40()

albums = top40.albums

for album in albums:
    print(
        album.position,
        album.title,
        "BY",
        album.artist,
        album.numWeeks,
        album.previousPosition
    )
```

If this code is run, it would result in something similar to this.

```
1 Never Been Better BY Olly Murs 1 0
2 X BY Ed Sheeran 23 2
3 FOUR BY One Direction 2 1
4 In The Lonely Hour BY Sam Smith 27 3
5 The Endless River BY Pink Floyd 3 4
6 Wanted On Voyage BY George Ezra 22 8
.
.
.
40 The London Sessions BY Mary J. Blige 1 0
```

1.2.8 Formatting the output columns

It's not easy to see the information, but you can now see that there are two numbers at the end of each line that represent the `numWeeks` and `previousPosition` attributes respectively.

So if we now wanted to make the formatting a little easier to read, we can make use of the `format()` function that allows us to carry out formatting on a string. The description of the `format()` function is outside the scope of this tutorial, but hopefully the following code will be relatively simple to follow.

```
from pythontop40 import Top40

top40 = Top40()

albums = top40.albums

for album in albums:
    print(
        "{:5} {:50} by {:50} {:5} {:5}".format(
            album.position,
            album.title,
            album.artist,
            album.numWeeks,
            album.previousPosition
        )
    )
```

When this code is run, it produces a column-based list of album entries that is much easier to understand.

```
1 Never Been Better                by Olly Murs
2 X                                by Ed Sheeran
3 FOUR                             by One Direction
4 In The Lonely Hour               by Sam Smith
5 The Endless River               by Pink Floyd
6 Wanted On Voyage                 by George Ezra
.
.
.
40 The London Sessions             by Mary J. Blige
```

Hopefully you can see that the format string features a series of place markers - represented by the `{ }` braces, and that each place marker brace corresponds with a data value in the list `format()` variables that follow.

```
from pythontop40 import Top40

top40 = Top40()

albums = top40.albums

for album in albums:
    print(
        "{:5} {:50} by {:50} {:5} {:5}".format(
            album.position,
            album.title,
            album.artist,
            album.numWeeks,
            album.previousPosition
        )
    )
```

Again, it will probably be clear that the text inside each of the braces such as `{:5}` tells the `format()` function how many columns that specific entry will take up.

So `{:5}` at the beginning of the format string, tells the `format()` function to use 5 columns for the first variable, and as `album.position` is the first in the list of variables inside the `format()` function, the position of the album in the chart will take up the first 5 columns.

The second `{}` brace contains `{:50}` which means it will occupy 50 columns, and the second variable is `album.title`, so the album title will occupy the next 50 columns, and so on...

Notice that in amongst all those `{}` braces, the format string actually contains the word `by`, because it's fine to put other things in the format string alongside the `{}` braces - even spaces! If it isn't a `{}` brace then it just gets produced as is.

1.2.9 Accessing the change information

As mentioned above the album `Entry` object has a `Change` object embedded within it.

```
entry = Entry(
    position = 3,
    previousPosition = 4,
    numWeeks = 26,
    artist = "Sam Smith",
    title = "In The Lonely Hour",
    Change(
        direction = "up",
        amount = 1,
        actual = 1
    )
)
```

The `Change` object actually describes the change since last week's chart in a little bit more detail. It provides access to the following pieces of information about the chart `Entry`.

- The `amount` of change in position since last week's chart. This is an `absolute` value - i.e. it describes the amount of change, but not the direction. So unless it is zero, it is always positive.
- The `actual` amount of change in positions since last week's chart. This can be negative, positive or zero.
- The `direction` of the change since last week. This is a `:py:func'str'` and is either `up` or `down`.

1.2.10 Printing the change information

So if we wanted to alter our program so that we started printed a summary of whether the album had gone up or down since last week, we could do so as follows.

```
from pythontop40 import Top40

top40 = Top40()

albums = top40.albums

for album in albums:
    print(
        "{:5} {:50} by {:50} {:5} {:5} - {:4}({:4})".format(
            album.position,
            album.title,
            album.artist,
```

```
        album.numWeeks,  
        album.previousPosition,  
        album.change.direction,  
        album.change.amount  
    )  
)
```

You'll see that we've added the following `{ }` braces to the format string

```
"{:4}({:4})"
```

and we've also added two more variables to the `format()` function.

```
album.change.direction,  
album.change.amount
```

These changes result in the following text output when the code is run.

```
1 Never Been Better          by Olly Murs  
2 X                          by Ed Sheeran  
3 FOUR                       by One Direction  
4 In The Lonely Hour        by Sam Smith  
5 The Endless River         by Pink Floyd  
6 Wanted On Voyage          by George Ezra  
.  
.  
.  
40 The London Sessions      by Mary J. Blige
```

1.2.11 Some finishing touches

Finally, we'll make some significant changes to the program to add column headings, column formatting, and to alter the text that describes the change since last week.

The output of the new program looks like this.

	No.		Title		Artist
	-----		-----		-----
	1		Never Been Better		Olly Murs
	2		X		Ed Sheeran
	3		FOUR		One Direction
	4		In The Lonely Hour		Sam Smith
	5		The Endless River		Pink Floyd
	6		Wanted On Voyage		George Ezra
	7		1989		Taylor Swift
	8		Listen		David Guetta
	9		Sonic Highways		Foo Fighters
	10		It's The Girls		Bette Midler
	11		Partners		Barbra Streisand
	12		Love In Venice		André Rieu
	13		Hope		Susan Boyle
	14		Dublin To Detroit		Boyzone
	15		No Sound Without Silence		The Script
	16		Forever		Queen
	17		Christmas		Michael Bublé
	18		Motion		Calvin Harris
	19		Blue Smoke - The Best Of		Dolly Parton
	20		Home Sweet Home		Katherine Jenkins

21	The Greatest Hits	Luther Vandross
22	Strictly Come Dancing	Dave Arch & The Strictly Come Dancing
23	Melody Road	Neil Diamond
24	A Perfect Contradiction	Paloma Faith
25	Sirens Of Song	Jools Holland & His Rhythm & Blues Or
26	Chapter One	Ella Henderson
27	Serenata	Alfie Boe
28	My Dream Duets	Barry Manilow
29	Aquostic (Stripped Bare)	Status Quo
30	Nothing Has Changed (The Best of David Bowie)	David Bowie
31	Love In The Future	John Legend
32	Stand Beside Me: Live In Concert	Daniel O'Donnell
33	Royal Blood	Royal Blood
34	5 Seconds Of Summer	5 Seconds of Summer
35	Caustic Love	Paolo Nutini
36	Nostalgia	Annie Lennox
37	No Fixed Address	Nickelback
38	If Everyone Was Listening	Michael Ball
39	+	Ed Sheeran
40	The London Sessions	Mary J. Blige

And below is the complete program that produced the output above.

```
from pythontop40 import Top40
```

```
top40 = Top40()
format_string = "| {:5} | {:50} | {:50} | {:8} | {:8} | {:22} |"
up_arrow = "^"
down_arrow = " v"
```

```
# Print the column headings
```

```
print(
    format_string.format(
        " No.",
        "Title",
        "Artist",
        " Weeks",
        "Previous",
        "Change since last week"
    )
)
```

```
# Print the heading underline
```

```
print(
    format_string.format(
        "-----",
        "-----",
        "-----",
        "-----",
        "-----",
        "-----"
    )
)
```

```
albums = top40.albums
```

```
for album in albums:
```

```
# Create the string that describes that change since last week
# If the amount of change since last week's chart is 0, or previous position in the chart was 0
# entry to the chart), then we should set the change_text to be empty.
if album.change.amount == 0:
    change_text = ''
elif album.previousPosition == 0:
    change_text = '    **NEW ENTRY**'
else:
    # We now know that there was a change in position since last week

    # We want to use the word place if there is only 1 place change, but if there is more than one
    # then we want to use the word places. To do this we will use a Python conditional assignment
    places_text = "place" if album.change.amount == 1 else "places"

    # We want to use the up arrow text if the album has moved up since last week, and the down arrow
    # has moved down. To do this we will also use a Python conditional assignment
    arrow_text = up_arrow if album.change.direction == "up" else down_arrow

    # Now let's build the change_text variable from the three components
    # - The arrow text
    # - The amount of change since last week
    # - The place text - using the correct plural term
    change_text = "{} by {} {}".format(
        arrow_text,
        album.change.amount,
        places_text
    )

# Print the output using the same format string that we used for the heading and underline
print(
    format_string.format(
        album.position,
        album.title,
        album.artist,
        album.numWeeks,
        album.previousPosition,
        change_text
    )
)
```

It might be worth spending a little time looking at the program and the output that it produces, to see if you can see which changes in the code produce which changes in the output.

1.3 PythonTop40 top40

The `top40` module contains the high level classes that are used to package the returned data such as `Entry`, `Chart` and `Change`.

In addition the `Top40` class provides the main external interface into the module. Once an instance of the `Top40` class has been instantiated it can be used to return data from the remote API to the called program:

```
from pythontop40 import Top40

top40 = Top40()

album_list = top40.albums
```

```
singles_list = top40.singles

albums_chart = top40.albums_chart
singles_chart = top40.singles_chart
```

From there, the returned objects can be interrogated and interacted with:

```
first_album = album_list[0]
print( first_album.position )
print( first_album.artist )

print("The date of the singles chart is", singles_chart.date)
print(The album_chart was retrieved from the server on", albums_chart.retrieved
```

And this, don't forget this:

```
class Repo(Model):
    name = fields.String()
    owner = fields.Embedded(User)

booby = Repo(
    name='Booby',
    owner={
        'login': 'jaimegildesagredo',
        'name': 'Jaime Gil de Sagredo'
    })

print booby.to_json()
'{"owner": {"login": "jaimegildesagredo", "name": "Jaime Gil de Sagredo"}, "name": "Booby"}'
```

class `top40.Change` (***kwargs*)

The Change model that describes the change of this entry since last week's chart.

This class isn't made publicly visible, so it should never really need to be initialised manually. That said, it is initialised by passing a series of keyword arguments, like so:

```
change = Change(
    direction = "down",
    amount = 2,
    actual = -2
)
```

The model does not feature any validation.

direction

str

The direction of the change “up” or “down”.

amount

int

The amount of change in chart position expressed as a positive integer.

actual

int

The amount of the change in chart position expressed as positive or negative (or 0).

Returns `Change`: The Change model instance created from the passed arguments.

```
class top40.Chart(**kwargs)
```

The Chart model that contains the embedded list of entries.

Parameters

- **entries** (`list` of `dict`) – A list of Python dictionaries. Each dictionary describes each `Entry` type in the chart, so the keys in the dictionary should match the properties of the `Entry` class.
- **date** (`int`) – The date of this chart as an integer timestamp containing the total number of seconds.
- **retrieved** (`int`) – The date that this chart was retrieved from the API server as an integer timestamp containing the total number of seconds.
- **current** (`bool`) – **Optional.** A flag used in V2 of the API to signify if the last scheduled read from the BBC's server worked or not. A value `True` means that the returned chart is the latest version that we have tried to read. A value of `False` means that the chart that is being returned is old. In all likelihood the chart is probably still accurate as it is only updated once per week, so this flag only means that the last scheduled read from the BBC's server did not work.

entries

`list` of `Entry`

A list of `Entry` types for each entry in the specific `Chart`. The entries are returned in the `list` with the highest chart position (i.e. the lowest number - #1 in the chart) first.

date

`int`

The date of this chart as an integer timestamp containing the total number of seconds. This value can then be converted to a Python `datetime.datetime` type by `datetime_type = datetime.datetime.fromtimestamp(chart.date)` (assuming that the `chart` variable was of type `Chart`).

retrieved

`int`

The date that this chart was retrieved from the API server as an integer timestamp containing the total number of seconds. This can be converted to a `datetime` type in the same as described for `date` above.

current

`bool`

Optional. A flag used in V2 of the API to signify if the last scheduled read from the BBC's server worked or not. A value `True` means that the returned chart is the latest version that we have tried to read. A value of `False` means that the chart that is being returned is old. In all likelihood the chart is probably still accurate as it is only updated once per week, so this flag only means that the last scheduled read from the BBC's server did not work.

Returns `Chart` – The Chart model instance created from the arguments.

Return type `Chart`

```
class top40.Entry(**kwargs)
```

The Entry model that contains the details about the chart entry, a `Change` Model is embedded in each Entry model.

Parameters

- **position** (`int`) – The position of this entry in the chart.

- **previousPosition** (`int`) – The position of this entry in the previous week’s chart.
- **numWeeks** (`int`) – The number of weeks this entry has been in the chart.
- **artist** (`str`) – The name of the artist for this entry.
- **title** (`str`) – The title of this entry.
- **change** (`Change`) – The embedded change model that describes the change in position.
- **status** (`str`) – **NEW in dev6** The text status from the BBC chart - takes the format of “new” | “up 3” | “down 4” | “non-mover”. Not used in Ben Major’s V1 API - optional

position`int`

The position of this entry in the chart.

previousPosition`int`

The position of this entry in the previous week’s chart.

numWeeks`int`

The number of weeks this entry has been in the chart.

artist`str`

The name of the artist for this entry.

title`str`

The title of this entry.

change`Change`

The embedded change model that describes the change in position.

status`str`

NEW in dev6 The text status from the BBC chart - takes the format of “new” | “up 3” | “down 4” | “non-mover”. Not used in Ben Major’s V1 API - optional

Returns `Entry`: The Entry model instance created from the arguments.

```
class top40.Top40 (base_url='http://ben-major.co.uk/labs/top40/api',          cache_duration=3600,
                  cache_config=None)
```

Provides the programmer with properties that return the Top 40 chart data.

The programmer creates an instance of this object, and then uses the exposed properties to access the data about the singles and albums charts.

Creates and returns the object instance.

All results will be cached for the duration of the existence of this instance in memory. However, if `cache_duration` is specified (not `None`), then results will be persisted to a local sqlite DB for the duration, in seconds, or `cache_duration`. A config for requests cache can also be passed in `cache_config` too, or if `None`, the default setting is used.

Parameters

- **base_url** (*str*) – The base url of the remote API before the specific service details are appended. For example, the base url might be “a.site.com/api/”, and the service “/albums/”, when appended to the base url, creates the total url required to access the album data.
- **cache_duration** (*int*) – If None, then the persistent cache will be disabled. Otherwise the cache duration specified will be used.
- **cache_config** (*dict*) – If None the default config will be used to pass to the `install_cache` method of `requests_cache`, otherwise the config in this parameter will be used. Any ‘expire_after’ key in the cache config will be replaced and the duration set to `cache_duration`.

error_format*str*

The format string to be used when creating error messages.

base_url*str*

The base url used to access the remote api

cache_duration*int*

The duration in seconds that results will be returned from the cache before a fresh read of the external API will replace them.

cache_config*dict*

A dictionary that describes the config that will be passed to the `request_cache` instance - allowing different backends and other options to be set.

Returns **Top40** – The Top40 instance.

Return type `Top40`

albums

A property that returns a `list` of album `Entry` types.

Returns `list`: A `list` of `Entry` instances. Each entry describes an album in the chart.

Raises

- `Top40HTTPError` (`Top40HTTPError`) – If a status code that is not 200 is returned
- `Top40ConnectionError` (`Top40ConnectionError`) – If a connection could not be established to the remote server
- `Top40ReadTimeoutError` (`Top40ReadTimeoutError`) – If the remote server took too long to respond

albums_chart

A property that returns the `Chart` object for the current Top40 albums

Returns `Chart`: The albums’ chart object - an instance of the `Chart` class containing the album information and the and the issue and retrieval dates specific to this chart.

Raises

- `Top40HTTPError` (`Top40HTTPError`) – If a status code that is not 200 is returned

- `Top40ConnectionError` (`Top40ConnectionError`) – If a connection could not be established to the remote server
- `Top40ReadTimeoutError` (`Top40ReadTimeoutError`) – If the remote server took too long to respond

reset_cache (*cache_duration=None*)

Remove any cached singles or albums charts

Because the UK Top40 charts only change once per week, `Top40` will cache the results of singles and albums. This means that during the execution of a program, repeated calls to retrieve singles and albums chart information will only actually call the remote API once. If, for whatever reason you need to ensure that an attempt to access single or album information actually results in a call to the remote API, then calling the `Top40.reset_cache()` method will do this, by clearing down any existing cached chart information.

If a cache is in place, then the results will also be cached across python runtime executions.

Params:

cache_duration (int): If `None` we will uninstall the requests cache and the next read from the API will cause a remote call to be executed. Otherwise it specifies the number of seconds before the persistent cache will expire.

singles

A property that returns a list of single entries.

Returns `list`: A `list` of `Entry` instances. Each entry describes a single in the chart.

Raises

- `Top40HTTPError` (`Top40HTTPError`) – If a status code that is not 200 is returned
- `Top40ConnectionError` (`Top40ConnectionError`) – If a connection could not be established to the remote server
- `Top40ReadTimeoutError` (`Top40ReadTimeoutError`) – If the remote server took too long to respond

singles_chart

A property that returns the `Chart` object for the current Top40 singles

Returns `Chart`: The singles' chart object - an instance of the `Chart` class containing the singles information and the issue and retrieval dates specific to this chart.

Raises

- `Top40HTTPError` (`Top40HTTPError`) – If a status code that is not 200 is returned
- `Top40ConnectionError` (`Top40ConnectionError`) – If a connection could not be established to the remote server
- `Top40ReadTimeoutError` (`Top40ReadTimeoutError`) – If the remote server took too long to respond

1.4 PythonTop40 errors

The errors module containing the exceptions that PythonTop40 uses

exception `errors.Top40ConnectionError`

This is raised when a connection cannot be established to the remote server

exception errors.Top40ConversionError

This is raised when a conversion is specified, but causes an error

exception errors.Top40Error

Base class for all exceptions

exception errors.Top40HTTPError (*message, return_code=0*)

This exception is raised if an HTTP level error was experienced. i.e. no physical or connection error, but a web server error was returned.

exception errors.Top40ReadTimeoutError

This is raised when an ongoing action takes longer than expected

1.5 Change Log for PythonTop40

1.5.1 v0.1.6 3rd January 2105

- Minor change. Bumped version number. Changed cache file generation to use system temp directory instead of CWD.

1.5.2 v0.1.5 29th December 2014

- Minor change. Bumped version number. Updated changes docs. Testing the automated build from a bitbucket push.

1.5.3 v0.1.4 29th December 2014

- Minor change. Removed the embedded test-requirements requirement from the dev-requirements.txt file.

1.5.4 v0.1.3 29th December 2014

- Minor change. Modified install_requires list. Removed nap and munch. Added requests-cache.

1.5.5 v0.1.2 29th December 2014

- Top40HTTPError now defaults to return_code=0
- Added the request-cache sqlite file to .gitignore
- Added requests-cache to requirements.txt
- Removed munch and nap from requirements.txt
- Added httpretty to test-requirements.txt
- Refactored errors returned when reading from the remote server
- Refactored tests away from mock and to use httpretty instead
- Implemented requests-cache. This should be seamless with existing code and will now persist results to local sqlite storage with a cache duration of 3600 seconds (one hour)
- Added option to pass an extended persistent_cache_config dictionary, which will be passed to the request_cache instance - this should allow for other cache types to sqlite.

- Removed the code to recursively create a Munch structure, Python native dicts are now used
- Removed the code to create a nap api object to access the remote server, and instead have replaced it with pure requests access. This was overkill for this project.
- Changed documentation
- Removed utils module and associated documentation
- Changed CHANGES.RST to CHANGES.rst
- Bumped version to 0.1.2

1.5.6 v0.1.1 14th December 2014

- Fix for TypeError ‘encoding’ is an invalid keyword argument for this function - trying to install PythonTop40 on Python 2.7

1.5.7 v0.1.0 13th December 2014

- Minor changes to the documentation relating to Python 2 status
- First official release

1.5.8 v0.1.0.dev9 - 12th December 2014

- Updated docs from previous change
- First cut at a Python 2 version. Need to create more tests so that I have greater coverage, but passing so far
- Changes to demo code so that it will run in V2 or V3
- Removed python3_compat.py
- Created extra dependencies - six and future
- Increased version number to v0.1.0.dev9

1.5.9 v0.1.0.dev8 - 12th December 2014

- Added optional field “current” to the Chart model - used in V2 of the API, but not in V1
- Increased version number to v0.1.0.dev8

1.5.10 v0.1.0.dev7 - 11th December 2014

- Refactoring of Doc generation and setup.py - both now get config information from package_info.json
- Increased version number to v0.1.0.dev7

1.5.11 v0.1.0.dev6 - 11th December 2014

- Added an optional field “status” to the Entry model - This will be filled in using V2 of the API, but will be not present for V1.
- Added Test changes to ensure “status” field can be present or non-present.
- Increased version number to v0.1.0.dev6

1.5.12 v0.1.0.dev5 - 8th December 2014

- Removed requirement for development version of Booby from setup.py
- Removed the trailing slash from the URL for /singles and /albums

1.5.13 v0.1.0.dev4 - 6th December 2014

- Changed the way the setuptools `long_description` is accessed
- Documentation changes
- Removed `demo.py`
 - Changed installation instructions to refer to **PythonTop40** on PyPI
 - Moved changes text into project route directory’s `CHANGES.RST` and included them into `docs/changes.rst`
 - Moved code examples out of the `moredetail.rst` file, and used `literalinclude` instead.
 - Added a link to the ReadTheDocs documentation into the `README.rst` file

1.5.14 v0.1.0.dev3 - 6th December 2014

- Minor change to documentation

ToDo Modify links to PyPI and ReadTheDocs in the rest of the documentation.

1.5.15 v0.1.0.dev2 - 6th December 2014

- Test coverage increased
- Initial documentation complete
- Documentation uploaded to [ReadTheDocs](#)
- **PythonTop40** now installable using `pip - pip install pythontop40`
- **PythonTop40** now uploaded to [PyPi](#)

ToDo Modify links to PyPI and ReadTheDocs in the rest of the documentation.

1.5.16 V0.1.0.dev1 - 4th December 2014

Initial version with working code and some tests.

ToDo:

- Complete tests coverage
- Complete documentation
- Upload documentation to `readthedocs`.
- Make code installable using `setup.py / pip`.
- Make code installable from PyPI using `pip`.

PythonTop40

The **PythonTop40** library is designed to be used in UK schools to provide students with access to data that describes the UK Top 40 singles and albums.

This is part of a wider initiative that I'm referring to as **Code-Further**. The hope is that by providing simple interfaces to information that is relevant to students, they will be able to relate to the data and imagine more ways in which they could consume and use it in their code - and hopefully **Code-Further**.

The data that **PythonTop40** accesses is provided by the excellent work by [@Ben Major](#) and his [UK Top 40 Charts API](#).

PythonTop40 is under active development visit [this blog post](#) for more information. and licensed under the [Apache2 license](#), so feel free to [contribute](#) and [report errors and suggestions](#).

Note: The **PythonTop40** library is designed to be used in UK schools to provide programmatic access to data that describes the UK Top 40 singles and albums. The hope is that by providing simple interfaces to access information that students may have an interest in, they may be inspired to **code-further**. This documentation will therefore most likely be aimed at teachers and education professionals, who may not have a deep knowledge of Python.

Warning: **PythonTop40** is currently designed to work with Python version 3. I have recently carried out some work to make it run on Python 2, but this does need to be more thoroughly tested than my current Nose tests allow. If you [encounter any issues](#), or you'd like to [submit a pull request](#), please contact me on BitBucket.

2.1 Usage

PythonTop40 exposes a very simple API to developers. It is accessed by importing the `Top40` class into your module and creating an instance of this class, like so:

```
from pythontop40 import Top40
top40 = Top40()
```

The `top40` instance exposes a number of properties to the programmer. These include:

- `top40.albums`
- `top40.singles`
- `top40.albums_chart`
- `top40.singles_chart`

The example code below shows how you can use one of these properties to get a list of the current Top 40 albums.:

```
from pythontop40 import Top40

top40 = Top40()

albums = top40.albums

for album in albums:
    print(
        album.position,
        album.title,
        "BY",
        album.artist
    )
```

This short program uses the `albums` property of the `Top40` class to obtain the Python `list` of the current Top 40 UK albums. It then loops through this list, and at each iteration of the loop the variable *album* is set to the next album entry in the list.

A `print()` function then prints the `position`, `title` and `artist` attributes of the album `entry` resulting in something like this::

```
1 Never Been Better BY Olly Murs
2 X BY Ed Sheeran
3 FOUR BY One Direction
4 In The Lonely Hour BY Sam Smith
5 The Endless River BY Pink Floyd
.
.
.
40 The London Sessions BY Mary J. Blige
```

I hope it's pretty clear what is going on, but a more detailed discussion of what the program does on can be found [here](#).

Features

PythonTop40 provides:

- a list of the current Top 40 UK singles using the `singles` property of the `Top40` class.
- a list of the current Top 40 UK albums using the `albums` property of the `Top40` class.
- a `chart` object relating to either singles or albums. The `chart` object contains the:
 - `date` that the chart was published
 - the date that the chart was `retrieved` from the server
 - a `list` containing an `Entry` for each Top 40 single or album
- **PythonTop40** will also cache the results, so that once a result type (singles or albums) has been retrieved from the remote server, it will be returned on subsequent requests from the cache without refreshing from the remote server.
 - **PythonTop40** will use a persistent cache by default. This should ensure that the remote server is not hammered with requests when the data is unlikely to change too frequently. The default duration for the cache is 3600 seconds (1 hour). Unlike the in-memory cache, the persistent cache will survive after the Python interpreter run session ends. The duration can be changed by passing a `cache_duration` value to the `Top40` constructor. Using a value of `None` for `cache_duration` will disable the persistent cache and rely on the in-memory cache only.
 - The cache can be reset using the `reset_cache()` method, so that the next request for albums or singles information will be forced to obtain it by connecting to the remote server.

Installation

PythonTop40 can be found on the Python Package Index [PyPi here](#). It can be installed using `pip`, like so.

```
pip install pythontop40
```

Documentation

The documentation for **PythonTop40** can be found on the [ReadTheDocs](#) site.

API - Application Programming Interface

The full documentation of the classes and functions that make up **PythonTop40** can be found [here](#), and the errors and exceptions can be found [here](#).

6.1 Tests

To run the **PythonTop40** test suite, you should install the test and development requirements and then run nosetests.

```
$ pip install -r dev-requirements.txt
$ nosetests tests
```

6.2 Changes

See [Changes](#).

6.3 Indices and tables

- *genindex*
- *modindex*
- *search*

e

errors, [15](#)

t

top40, [10](#)

A

actual (top40.Change attribute), 11
albums (top40.Top40 attribute), 14
albums_chart (top40.Top40 attribute), 14
amount (top40.Change attribute), 11
artist (top40.Entry attribute), 13

B

base_url (top40.Top40 attribute), 14

C

cache_config (top40.Top40 attribute), 14
cache_duration (top40.Top40 attribute), 14
Change (class in top40), 11
change (top40.Entry attribute), 13
Chart (class in top40), 11
current (top40.Chart attribute), 12

D

date (top40.Chart attribute), 12
direction (top40.Change attribute), 11

E

entries (top40.Chart attribute), 12
Entry (class in top40), 12
error_format (top40.Top40 attribute), 14
errors (module), 15

N

numWeeks (top40.Entry attribute), 13

P

position (top40.Entry attribute), 13
previousPosition (top40.Entry attribute), 13

R

reset_cache() (top40.Top40 method), 15
retrieved (top40.Chart attribute), 12

S

singles (top40.Top40 attribute), 15
singles_chart (top40.Top40 attribute), 15
status (top40.Entry attribute), 13

T

title (top40.Entry attribute), 13
Top40 (class in top40), 13
top40 (module), 10
Top40ConnectionError, 15
Top40ConversionError, 15
Top40Error, 16
Top40HTTPError, 16
Top40ReadTimeoutError, 16