CodeFurther Documentation

Release 0.1.0.dev15

Danny Goodall

March 14, 2015

Contents

Warning: This documentation is currently out of sync with the functionality in the **CodeFurther** package. In addition to the functionality described in this documentation, the following data sources have also been implemented:

- `football`
- `weather`

On top of this, the following means of sharing have been implemented:

- `email`
- `twitter`
- `slack`
- `textmessage`

Many of these services require the presence of a `cf_setting.py` script that contains the following settings:

- `CF_TWITTER_APP_KEY`
- `CF_TWITTER_APP_SECRET`
- `CF_TWITTER_OAUTH_TOKEN`
- `CF_TWITTER_OAUTH_TOKEN_SECRET`
- `CF_SLACK_API_KEY`
- `CF_SMTP_SERVER`
- `CF_SMTP_PORT`
- `CF_SMTP_USERNAME`
- `CF_SMTP_PASSWORD`
- `CF_TWILIO_ACCOUNT`
- `CF_TWILIO_TOKEN`
- `CF_TWILIO_FROM`
- `CF_FORECASTIO_API_KEY`

If this script file is not found, then the corresponding environment variable will be searched for.
The entire **CodeFurther** package is under development, so proceed

CHAPTER 1

Contents

1.1 CodeFurther Installation

CodeFurther can be found on the Python Package Index PyPi here. It can be installed using `pip`, like so.

```
pip install codefurther
```

1.2 top40 - UK Top40 Charts

1.2.1 Introduction

The **Top40** module is part of the **codefurther** Python package, and is designed to be used in UK schools to provide students with access to data that describes the UK Top 40 singles and albums.

Top40 is part of a wider initiative that I'm referring to as **CodeFurther**. 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 **CodeFurther**.

The data that **Top40** accesses is provided by the excellent work by @Ben Major and his UK Top 40 Charts API.

The **Top40** module is under active development as part of the **CodeFurther** package and licensed under the Apache2 license, so feel free to contribute and report errors and suggestions.

Note: The **Top40** 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 **CodeFurther**. This documentation will therefore most likely be aimed at teachers and education professionals, who may not have a deep knowledge of Python.

Warning: **Top40** is currently designed to work with Python version 3. I have recently carried out a small amount of work to make it run on Python 2, but this does need to be more thoroughly tested that my current Nose tests allow. If you encounter any issues, or you'd like to submit a pull request, please contact me on BitBucket.

1.2.2 Features

Top40 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
- **Top40** 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.
- 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.

1.2.3 Usage

Top40 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 codefurther.top40 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 codefurther.top40 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 in the ExploringTheTop40DemoCode.

1.2.4 Exploring the Top40 Demo Code

Our example program

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

```
from codefurther.top40 import Top40

top40 = Top40()

albums = top40.albums

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

Importing the PythonTop40 module

The first line in our program

```
from codefurther.top40 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.

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 codefurther.top40 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.

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 codefurther.top40 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.

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 0 th item) in the list by placing ''[0]' after the name of a `list`.

```
print(albums[0])
Entry(postition = 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
```

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 codefurther.top40 import Top40

top40 = Top40()

albums = top40.albums

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

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 codefurther.top40 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
```

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 codefurther.top40 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 codefurther.top40 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.

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. The 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`.

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 codefurther.top40 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
```

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 Orc
|    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 codefurther.top40 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 on
        # 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 ar
        # 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.

1.2.5 Top40 API

Return the UK Top 40 Single and Album charts

class top40.**Top40**(*base_url_param=None*, *cache_duration=3600*, *cache_config=None*, *cache_prefix='codefurthercache'*)

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

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

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.**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 on 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 liekliehood the chart is probably still in 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 on 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 liekliehood the chart is probably still in 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`

1.3 lyrics - Search for song lyrics

The **Lyrics** module is part of the **codefurther** Python package, and is designed to be used in UK schools to provide students with access to data that describes the words to popular songs.

Lyrics is part of a wider initiative that I'm referring to as **CodeFurther**. 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 **CodeFurther**.

The data that **Lyrics** accesses is provided by Wikia and should be accessed as as part of its Lyrics Wikia site.

1.3.1 Features

Lyrics provides:

- Search for an artist on Lyrics Wikia
- Search for the lyrics of a song by a specific artist
- Search for all of the songs by a specific artist that are present on Lyrics Wikia.

1.3.2 Usage

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

```
from codefurther.lyrics import Lyrics
lyrics_machine = Lyrics()
```

To print out the lyrics to a song, the `song_lyrics()` method is used to to find the song, given the name of the artist and the name of the song.:

```
lyrics_list = lyrics_machine.song_lyrics("billy bragg", "days like these")
```

The lyrics are returned as a `list` of `str` items, and be printed simply, like so.:

```
for line in lyrics_list:
    print( line )
```

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

- `Lyrics.song_lyrics`
- `Lyrics_artist_songs`
- `Lyrics_artist_search`

The example code below shows how you can use these properties. This code, simply returns the lyrics to a song, given the name of the artist and the name of the song.:

```
from codefurther.lyrics import Lyrics

lyrics_machine = Lyrics()

lyrics_list = lyrics_machine.song_lyrics("billy bragg", "days like these")
```

```
for count, line in enumerate(lyrics_list):
    print(
        "{}. {}".format(
            count+1,
            line
        )
    )
```

This results in the following output.:

```
1. The party that became so powerful
2. By sinking foreign boats
3. Is dreaming up new promises
4. Because promises win votes
.
.
.
31. Peace, bread, work, and freedom
32. Is the best we can achieve
33. And wearing badges is not enough
34. In days like these
```

The following code find all of the songs for a given artist.:

```
from codefurther.lyrics import Lyrics

lyrics_machine = Lyrics()

song_list = lyrics_machine.artist_songs("billy bragg")

for count, song in enumerate(song_list):
    print(
        "{}. {}".format(
            count+1,
            song
        )
    )
```

Resulting in the following output.:

```
1. The Milkman of Human Kindness
2. To Have and to Have Not
3. Richard
.
.
.
396. To Have And Have Not
397. Walk Away Renee
398. Youngest Son
```

This code allows the programmer to search for the exact name of an artist.:

```
from codefurther.lyrics import Lyrics

lyrics_machine = Lyrics()

artist_details = lyrics_machine.artist_search("Billy Bragg")

print(artist_details)
```

If an exact match of the artist is not found, then the nearest match is returned.

1.3.3 Lyrics API

Return the lyrics given an artist and song.

class `lyrics.`**Lyrics**(*base_url='http://cflyricsserver.herokuapp.com/lyricsapi/'*)
Provides the programmer with properties that return lyrics from the Wikia site.

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

error_format
`str`

The format string to be used when creating error messages.

bad_response
`str`

The text to be used in the raised error if the server response is unexpected.

artist_exists(*artist*)
Determines whether an artist exists in Lyrics Wikia USING THE SPELLING and puntuation provided.

Proceed with a little caution as I'm not completely sure that these results are accurate.

Returns True if the artist specified is found exactly. It may be that the artist is known by another, similar name.

Parameters **artist** – (`str`) The name of the artist being searched for.

Returns (`bool`): If the artist was found exactly as named in the search results, then `True` is returned, otherwise False is returned.

Return type result

artist_search(*artist*)
Returns the first result of a search for the given artist on Lyrics Wikia.

Proceed with a little caution as I'm not completely sure that these results are accurate.

Returns a string containing the search result. The actual string returned depends on what the search functionality at Lyrics Wikia returns.

Parameters **artist** – (`str`) The name of the artist being searched for.

Returns (`str`): The result of the search. If the string contains a colon `:`, then it typically means that an artist and song has been returned, separated by the colon. If a string is returned without a colon, then it likely means that only an artist match was found, but the artise returned should be checked to see if it is the same as the artist that was searched for.

Return type result

artist_songs(*artist*)
Returns a generator that yields song titles for the given artist.

If the *all_details* flag is set to *True*, then a `dict` is returned that contains. Returns an empty generator if no songs were found for the specified artist.

Parameters **artist** – (`str`) The name of the artist for the song being looked up.

Returns (`list`): A `list` of `str` representing each song of the artist.

Return type song_list

Raises

- `ValueError` – If artist is `None` or `""` (empty).
- `ValueError` – If the response from the server is not in the correct format.

song_lyrics(*artist*, *title*)

Return a list of string lyrics for the given artist and song title.

Parameters

- **artist** – (`str`) The name of the artist for the song being looked up.
- **title** – (`str`) The name of the song being looked up.

Returns (`list`) of (`str`) one for each lyric line in the song. Blank lines can be returned to space verses from the chorus, etc.

1.4 directions - Google maps routes

The **Directions** module is part of the **codefurther** Python package, and is designed to be used in UK schools to provide students with access to data that describes journeys from one point to another.

Directions is part of a wider initiative that I'm referring to as **CodeFurther**. 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 **CodeFurther**.

The data that **Directions** accesses is provided by Google as part of its Google Maps API.

1.4.1 Features

Directions provides:

- Simplified access to directions from *a* to *b* using Google Maps
- Place names can be vague, Google will do it's best to decipher them
- Walking, cycling, driving and transit routes

1.4.2 Usage

Importing the module

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

```
from codefurther.directions import GetDirections
directions = GetDirections("southampton", "winchester")
```

GetDirections parameters

The `GetDirections` object is initialised with a starting location and an end location. It can also be initialised with an optional `mode` parameter that describes the mode of transport to use.:

```
from codefurther.directions import GetDirections

driving_directions = GetDirections("southampton","winchester", "driving")
walking_directions = GetDirections("southampton","winchester", "walking)
bike_directions = GetDirections("southampton","winchester", "bicycling")
transit_directions = GetDirections("southampton","winchester", "transit")
```

GetDirections properties

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

- `GetDirections.found`
- `GetDirections.heading`
- `GetDirections.footer`
- `GetDirections.steps`

Example program

The example code below shows how you can use these properties to get directions from one place to another.:

```
from codefurther.directions import GetDirections

directions = GetDirections("Southampton, UK", "Winchester, UK")

if directions.found:
    print( directions.heading )
    for step in directions.steps:
        print( step )
    print( directions.footer )
```

When run, this code results in the following extract being printed to the display.:

```
These are the steps for the (walking) journey from Southampton, Southampton, UK to Winchester, Winche
1. Head east (8 m / 1 min)
2. Turn left toward Brunswick Pl/A3024 (7 m / 1 min)
3. Turn right toward Brunswick Pl/A3024 (0.2 km / 3 mins)
.
.
.
39. Turn right onto Colebrook St (85 m / 1 min)
Map data ©2014 Google
```

It is possible to run the same code without first checking if the result was found. In that case, **CodeFurther** will simply replace the header text with a message stating that the route could not be found, no direction steps will be returned, and the footer will be blank too.:

```
from codefurther.directions import GetDirections

directions = GetDirections("1231123", "345345134")
```

```
print(directions.heading)
for step in directions.steps:
    print(step)
print(directions.footer)
```

When run, this code would produce the following output (returned by the `GetDirections.heading` property).

```
We couldn't find (walking) directions from: 1231123, to 345345134.
```

The idea here is that when used in the classroom, the students will not be put off experimenting by having to remember to check for the `GetDirections.found` property.

1.4.3 Directions API

Return the directions from a given start point to a given end point.

class `directions.`**`GetDirections`**(*starting_point*, *end_point*, *mode='walking'*)

A wrapper for the gmaps Direction class to make it simpler to deal with in the classroom

The `GetDirections` class is inititialised with a starting point, an end_point and and optional transport mode.

```
>>> directions = GetDirections("southampton, UK","winchester, UK", mode="walking")
```

This returns an object that can then be interrogated for information about the route.

```
>>> if directions.found:
>>>     print("Directions found!")
>>> else:
>>>     print("Couldn't find a route.")
```

If a valid route was found, simplified and prettified text can be accessed through:

```
>>> directions.heading
>>> "These are the steps for the (walking) journey from Southampton, Southampton, UK to Winchest
```

and:

```
>>> directions.footer
>>> "Map data ©2014 Google"
```

and the steps of the route can be accessed through the `GetDirections.steps` property, which returns a `list` of `str`.

```
>>> directions.steps
>>> [
>>>     "1. Head east (8 m / 1 min)",
>>>     "2. Turn left toward Brunswick Pl/A3024 (7 m / 1 min)",
>>>     "3. Turn right toward Brunswick Pl/A3024 (0.2 km / 3 mins)",
>>>     "4. Turn right onto Brunswick Pl/A3024 (13 m / 1 min)",
>>>     .
>>>     .
>>>     .
>>>     "39. Turn right onto Colebrook St (85 m / 1 min)"
>>> ]
```

The raw `gmaps.Directions` object can be accessed using:

```
>>> directions.raw
```

Once the object has been created, subsequent calls to the `GetDirections.journey()` method will create new routes without the need to create a brand new `GetDirections()` object.

```
>>> new_directions = directions.new_journey("winchester, uk", "southampton, uk", "driving")
```

footer
Returns the text footer for this route.

Returns the text footer for this route which is usually a copyright notice - if valid. Or if not valid then a null string is returned instead.

Returns **_footer** – A text copyright notice if it is valid, or a null string if the route is not valid.

Return type `str`

found
Reveals if the route specified was found.

Returns a boolean True if the route specified was found by Google, or False if the route could not be found.

Returns **_found** – True if the route was valid or False if the route was invalid.

Return type `bool`

heading
Returns the text heading for this route.

Returns the text heading for this route - if valid. Or if not valid then an appropriate message is returned instead.

Returns **_heading** – A text description of the route if it is valid, or an appropriate message if the route is not valid.

Return type `str`

new_journey(*starting_point*, *end_point*, *mode=None*)
Create a new journey, specifying start and end points and the mode of travel.

This method pretty much mirrors the `GetDirections.__init__()` method.

If the journey appears valid to Google Maps, then this method sets appropriate values for the `GetDirections.heading` method, the `GetDirections.footer` and the `GetDirections.steps` methods.

Parameters

- **starting_point** (`str`) – The text string that describes the starting point for the route
- **end_point** (`str`) – The text string that describes the end point for the route
- **mode** (`str`, optional) – Text string either "walking", "driving", "bicycling" or "transit", defaults to "walking".

starting_point
`str`

The text string that describes the starting point for the route

end_point
`str`

The text string that describes the end point for the route

mode
str

Text string either "walking", "driving", "bicycling" or "transit". Note that transit doesn't seem to be widely supported outside of the US.

default_mode
str

The mode that was specified the first time the object instance was created.

Returns **self** – Returns an instance of the GetDirections object to allow for object chaining.

Return type GetDirections

raw
Provides access to the raw `gmaps.directions.Directions` class object.

Returns **_directions** – The raw Directions object.

Return type `gmaps.Directions`

steps
Returns a list of strings that describe the steps for this route.

Returns a list of strings or an empty list if the route is not valid.

Yields (str) : A generator of list of strings that describe the steps for this route, or an empty list if the route is not valid.

1.5 CodeFurther utils

The utils module contains utility functions and classes used by the other modules in the suite.

utils.**isolate_path_filename**(*uri*)
Accept a url and return the isolated filename component

Accept a uri in the following format - http://site/folder/filename.ext and return the filename component.

Parameters **uri** (str) – The uri from which the filename should be returned

Returns **file_component** – The isolated filename

Return type str

1.6 CodeFurther errors

The errors module containing the exceptions that PythonTop40 uses

exception errors.**CodeFurtherError**
Base class for all exceptions

exception errors.**CodeFurtherConversionError**
This is raised when a conversion is specified, but causes an error

exception errors.**CodeFurtherConnectionError**
This is raised when a connection cannot be established to the remote server

exception errors.**CodeFurtherReadTimeoutError**
This is raised when an ongoing action takes longer than expected

exception errors.**CodeFurtherHTTPError**(*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.**CodeFurtherWeatherError**
This is raised if a problem occurs when a weather forecast is being retrieved

1.7 Change Log for CodeFurther

1.7.1 v0.1.0.dev7 13th January 2015

- Still trying to optimize the requirements and (install_requires)
- Removed Munch requirement
- Removed nap requirement
- Added MarkupSafe requirement to install_requires

1.7.2 v0.1.0.dev6 13th January 2015

- Added Pull request from @msabramo SHA: cbcbafea74e1998e3740033e0271ab3cb11adc20 to remove additional restructured text chars that break the PyPI module
- Added dev requirement for restview so that I can test for the above
- Added setup requirement for python-gmaps (install_requires)
- Added setup requirement for requests-cache (install_requires)

1.7.3 v0.1.0.dev5 12th January 2015

- Tidied files in the MANIFEST.in
- Included NULL.rst in the MANIFEST.in
- Included requirements files in the MANIFEST.in - not sure if they should be there yet as I'm learning about packaging Python modules
- Changes the MANIFEST.in to include codefurther/package_info.json
- Update CHANGES.RST
- Renamed CHANGES.rst to have lowercase extension
- Removed orphaned top40demo.py file from project route

1.7.4 v0.1.0.dev4 10th January 2015

- Minor fix to requirements files

1.7.5 v0.1.0.dev3 10th January 2015

- Minor fix to requirements files

1.7.6 v0.1.0.dev2 10th January 2015

- Merged PythonTop40 v0.1.6 (not 0.1.5 as the branch suggests)
- Added support for requests cache for top40
- Moved requests cache temp file to temp directory
- Made the test FileSpoofer class generic across multiple tests
- Added tests to check for malformed JSON in lyrics
- Removed some redundant tests
- Added code to test directions
- Added artist_exists() method for the lyrics module
- Changed the way lyrics handles HTTP errors
- Created helpers module for functions/classes across different modules
- Added PY2 / PY3 compatible code to several modules

ToDo:

- Add weather forecast support
- Add Premiership football leagues
- Add IMDB / Movie DB querying
- Lots of other ideas too
- etc.

CHAPTER 2

CodeFurther

The **CodeFurther** library is designed to be used in UK schools to provide students with access to data that hopefully has some relevance for them. The hope is that by gaining access to meaningful data, they will be inspired to **CodeFurther**.

CodeFurther is under active development and is licensed under the Apache2 license, so feel free to contribute and report errors and suggestions.

Note: The **CodeFurther** package 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 **CodeFurther**. This documentation will therefore most likely be aimed at teachers and education professionals, who may not have a deep knowledge of Python.

Warning: **CodeFurther** is currently designed to work with Python version 3. I have recently carried out some work to make it run on Python 2 too, but this does need to be more thoroughly tested that 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 Modules in the Package

CodeFurther contains a number of modules that provide access to *interesting* data. Those modules are shown below:

Table 2.1: CodeFurther Modules

Module	Description
`top40`	Provides access to the UK Top 40 charts for singles and albums.
`lyrics`	Allows lyrics for a given artist and song title to be accessed within Python.
`directions`	Allows Google Maps route directions to be accessed from within Python.

CHAPTER 3

Features

CodeFurther provides:

- a list of the current Top 40 UK singles using the *singles <top40.Top40.singles>* property of the *~top40.Top40* class.
- a list of the current Top 40 UK albums using the *albums <top40.Top40.singles>* property of the *~top40.Top40* class.
- the ability to retrieve the lyrics for a given artist
- the ability to find all of the songs for a given artist
- the ability to search for a specific artist

CHAPTER 4

Installation

CodeFurther can be found on the Python Package Index PyPi here. It can be installed using `pip`, like so.

```
pip install codefurther
```

CHAPTER 5

Documentation

The documentation for **CodeFurther** can be found on the ReadTheDocs site.

5.1 Tests

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

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

5.2 Changes

See *Changes <changes>*.

5.3 Indices and tables

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

T

U