pokebase Documentation

Release 2.0.0

Greg Hilmes

Aug 01, 2018

Contents:

The simple but powerful Python interface to the PokéAPI database.

CHAPTER 1

Getting Started

1.1 Installation

```
pip install pokebase
```

Alternatively, the source can be found on GitHub

1.2 Importing

For most use cases,

```
import pokebase
```

will be sufficient. Or, to save a few characters:

```
import pokebase as pb
```

This will include the core wrapper classes, as well as the loaders functions.

Note: If you plan to change the cache location, avoid importing the cache constants directly. You should only import them with the whole cache module. If you do not do this, calling `set_cache()` will not change your local copy of the variable.

Bad!

```
>>> from pokebase.cache import API_CACHE
```

Good!

```
>>> from pokebase import cache
>>> cache.API_CACHE
```

CHAPTER 2

Examples

Here is a selection of possible ways to use `pokebase` to help you access PokéAPI. Obviously they can't *all* be listed, but if you think of another good example you would like to add yourself, submit a pull request with your new example on GitHub. New examples are always welcome!

2.1 Simple Usage

The best way to get familiar with pokebase/PokéAPI is to open up a session in the Python interactive interpreter!

```
>>> import pokebase as pb
>>> bulba = pb.pokemon(1)
>>> for type_slot in bulba.types:
...     print('{}: {}'.format(type_slot.slot, type_slot.type.name.title()))
...
1: Grass
2: Poison
>>> bulba.base_experience
64
>>> pound = pb.move('pound')
>>> pound.accuracy
100
>>> pound.type.name.title()
'Normal'
```

Check out the PokéAPI docs for even more use-cases, information, and data examples.

2.2 Getting All Pokémon Names from a Generation

In this example, we want to print out the name of every Pokémon that was introduced in a given generation of Pokémon games.

```
import pokebase as pb

# View Pokemon from this generation number.
GENERATION = 2

# Get API data associated with that particular generation.
gen_resource = pb.generation(GENERATION)

# Iterate through the list of Pokemon introduced in that generation.
for pokemon in gen_resource.pokemon_species:
    print(pokemon.name.title())
```

2.3 Finding Moves of a Certain Type

A quick 'n' dirty method to do this is to get a list of every move, check its type, and then print it if the type matches our searched type. And this is exactly what I did when writing this example. But there's a better way (we'll get to that in a second). Here's the bad example:

```
import pokebase as pb

# Print all moves of the type named here.
TYPE = 'normal'

# Get a list of EVERY move from the API.
all_moves = pb.APIResourceList('move')

# Bad method. Don't actually do this.
for move_data in all_moves:
    # Get API data for this move.
    move = pb.move(move_data['name'])

    # Print its name, if its type matches.
    if move.type.name == TYPE:
        print(move.name)

```

But sometimes the API has done work for you already. Looking in the docs for types, we see that each type resource has a `moves` method. Here's the better code.

```
import pokebase as pb

# Print all moves of the type named here.
TYPE = 'normal'

# Good method.
# Get API data associated with the type we want.
type_moves = pb.type_(TYPE).moves

# Iterate & print.
for move in type_moves:
    print(move.name)
```

The first example is poor is because it calls the API signicantly more times than this second example does (once for *every* move, as opposed to only calling the API for each move in the type's list of moves). The second method

guarantees that whenever we call the API for a move, we know it has the type that we're looking for. The more API calls you have, the slower your script runs.

2.4 Making a Type Chart

For this example, we want to write a function to find the type multiplier when any one Pokémon type attacks any other. This could be useful if you wanted to make a move damage calculator or a game to test your knowledge of the Pokémon type chart.

```
import pokebase as pb

TYPES = ['normal', 'fighting', 'flying', 'poison', 'ground', 'rock', 'bug', 'ghost',
↪'steel', 'fire', 'water', 'grass', 'electric', 'psychic', 'ice', 'dragon', 'dark',
↪'fairy']

def type_multiplyer(attack, defense):
    # Get API data for the attcking type.
    atk_type = pb.type_(attack)

    # Check which damage_relation list the defense is in. Matches by name
    if defense in [t.name for t in atk_type.damage_relations.no_damage_to]:
        return 0.0
    elif defense in [t.name for t in atk_type.damage_relations.half_damage_to]:
        return 0.5
    elif defense in [t.name for t in atk_type.damage_relations.double_damage_to]:
        return 2.0
    else:
        return 1.0
```

CHAPTER 3

API Reference

CHAPTER 4

Indices and tables

- genindex
- modindex
- search