# confusable$_h$omoglyphsDocumentation

***Release 2.0.1+2.gba211a6.dirty***

**Victor Felder**

January 30, 2017

Contents

Contents:

CHAPTER 1

# confusable_homoglyphs [doc]

*a homoglyph is one of two or more graphemes, characters, or glyphs with shapes that appear identical or very similar* wikipedia:Homoglyph

Unicode homoglyphs can be a nuisance on the web. Your most popular client, AlaskaJazz, might be upset to be impersonated by a trickster who deliberately chose the username AlaskaJazz.

- `AlaskaJazz` is single script: only Latin characters.
- `AlaskaJazz` is mixed-script: the first character is a greek letter.

You might also want to avoid people being tricked into entering their password on `www.microsft.com` or `www.faebook.com` instead of `www.microsoft.com` or `www.facebook.com`. Here is a utility to play with these **confusable homoglyphs**.

Not all mixed-script strings have to be ruled out though, you could only exclude mixed-script strings containing characters that might be confused with a character from some unicode blocks of your choosing.

- `Allo` and $\rho\tau\tau$ are fine: single script.
- `Allo`$\Gamma$ is fine when our preferred script alias is 'latin': mixed script, but $\Gamma$ is not confusable.
- `Allo`$\rho$ is dangerous: mixed script and $\rho$ could be confused with `p`.

This library is compatible Python 2 and Python 3.

## 1.1 API documentation

## 1.2 Is the data up to date?

Yep.

The unicode blocks aliases and names for each character are extracted from this file provided by the unicode consortium.

The matrix of which character can be confused with which other characters is built using this file provided by the unicode consortium.

This data is stored in two JSON files: `categories.json` and `confusables.json`. If you delete them, they will both be recreated by downloading and parsing the two abovementioned files and stored as JSON files again.

CHAPTER 2

# Installation

At the command line:

```
$ easy_install confusable_homoglyphs
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv confusable_homoglyphs
$ pip install confusable_homoglyphs
```

CHAPTER 3

# Usage

To use confusable_homoglyphs in a project:

```
pip install confusable_homoglyphs
import confusable_homoglyphs
```

CHAPTER 4

# API Documentation

## 4.1 confusable_homoglyphs package

### 4.1.1 Submodules

### 4.1.2 confusable_homoglyphs.categories module

confusable_homoglyphs.categories.**alias**(*chr*)
Retrieves the script block alias for a unicode character.

```
>>> categories.alias('A')
'LATIN'
>>> categories.alias('τ')
'GREEK'
>>> categories.alias('-')
'COMMON'
```

**Parameters** **chr** (*str*) – A unicode character

**Returns** The script block alias.

**Return type** str

confusable_homoglyphs.categories.**aliases_categories**(*chr*)
Retrieves the script block alias and unicode category for a unicode character.

```
>>> categories.aliases_categories('A')
('LATIN', 'L')
>>> categories.aliases_categories('τ')
('GREEK', 'L')
>>> categories.aliases_categories('-')
('COMMON', 'Pd')
```

**Parameters** **chr** (*str*) – A unicode character

**Returns** The script block alias and unicode category for a unicode character.

**Return type** (str, str)

confusable_homoglyphs.categories.**category**(*chr*)
Retrieves the unicode category for a unicode character.

```
>>> categories.category('A')
'L'
>>> categories.category('τ')
'L'
>>> categories.category('-')
'Pd'
```

**Parameters** **chr** (*str*) – A unicode character

**Returns** The unicode category for a unicode character.

**Return type** str

confusable_homoglyphs.categories.**generate**()
Generates the categories JSON data file from the unicode specification.

**Returns** True for success, raises otherwise.

**Return type** bool

confusable_homoglyphs.categories.**unique_aliases**(*string*)
Retrieves all unique script block aliases used in a unicode string.

```
>>> categories.unique_aliases('ABC')
{'LATIN'}
>>> categories.unique_aliases('ραΑτ-')
{'GREEK', 'LATIN', 'COMMON'}
```

**Parameters** **string** (*str*) – A unicode character

**Returns** A set of the script block aliases used in a unicode string.

**Return type** (str, str)

### 4.1.3 confusable_homoglyphs.confusables module

**exception** confusable_homoglyphs.confusables.**Found**
Bases: exceptions.Exception

confusable_homoglyphs.confusables.**generate**()
Generates the confusables JSON data file from the unicode specification.

**Returns** True for success, raises otherwise.

**Return type** bool

confusable_homoglyphs.confusables.**is_confusable**(*string*, *greedy=False*, *preferred_aliases=[]*)
Checks if `string` contains characters which might be confusable with characters from `preferred_aliases`.

If `greedy=False`, it will only return the first confusable character found without looking at the rest of the string, `greedy=True` returns all of them.

`preferred_aliases=[]` can take an array of unicode block aliases to be considered as your 'base' unicode blocks:

- considering `paρa`,
  - with `preferred_aliases=['latin']`, the 3rd character `ρ` would be returned because this greek letter can be confused with latin `p`.

– with `preferred_aliases=['greek']`, the 1st character `p` would be returned because this latin letter can be confused with greek $\rho$.

– with `preferred_aliases=[]` and `greedy=True`, you'll discover the 29 characters that can be confused with `p`, the 23 characters that look like `a`, and the one that looks like $\rho$ (which is, of course, *p* aka *LATIN SMALL LETTER P*).

```
>>> confusables.is_confusable('paρa', preferred_aliases=['latin'])[0]['character']
'ρ'
>>> confusables.is_confusable('paρa', preferred_aliases=['greek'])[0]['character']
'p'
>>> confusables.is_confusable('Abç', preferred_aliases=['latin'])
False
>>> confusables.is_confusable('AlloΓ', preferred_aliases=['latin'])
False
>>> confusables.is_confusable('ρττ', preferred_aliases=['greek'])
False
>>> confusables.is_confusable('ρτ.τ', preferred_aliases=['greek', 'common'])
False
>>> confusables.is_confusable('ρττp')
[{'homoglyphs': [{'c': 'p', 'n': 'LATIN SMALL LETTER P'}], 'alias': 'GREEK', 'character': 'ρ'}]
```

**Parameters**

- **string** (*str*) – A unicode string
- **greedy** (*bool*) – Don't stop on finding one confusable character - find all of them.
- **preferred_aliases** (*list(str)*) – Script blocks aliases which we don't want `string`'s characters to be confused with.

**Returns** False if not confusable, all confusable characters and with what they are confusable otherwise.

**Return type** bool or list

`confusable_homoglyphs.confusables.`**is_dangerous**(*string*, *preferred_aliases=[]*)

Checks if `string` can be dangerous, i.e. is it not only mixed-scripts but also contains characters from other scripts than the ones in `preferred_aliases` that might be confusable with characters from scripts in `preferred_aliases`

For `preferred_aliases` examples, see `is_confusable` docstring.

```
>>> bool(confusables.is_dangerous('Allo'))
False
>>> bool(confusables.is_dangerous('AlloΓ', preferred_aliases=['latin']))
False
>>> bool(confusables.is_dangerous('Alloρ'))
True
>>> bool(confusables.is_dangerous('AlaskaJazz'))
False
>>> bool(confusables.is_dangerous('AlaskaJazz'))
True
```

**Parameters**

- **string** (*str*) – A unicode string
- **preferred_aliases** (*list(str)*) – Script blocks aliases which we don't want `string`'s characters to be confused with.

**Returns** Is it dangerous.

**Return type** bool

confusable_homoglyphs.confusables.**is_mixed_script**(*string*, *allowed_aliases=['COMMON']*)

Checks if `string` contains mixed-scripts content, excluding script blocks aliases in `allowed_aliases`.

E.g. `B. C` is not considered mixed-scripts by default: it contains characters from **Latin** and **Common**, but **Common** is excluded by default.

```
>>> confusables.is_mixed_script('Abç')
False
>>> confusables.is_mixed_script('ρτ.τ')
False
>>> confusables.is_mixed_script('ρτ.τ', allowed_aliases=[])
True
>>> confusables.is_mixed_script('Alloτ')
True
```

**Parameters**

- **string** (*str*) – A unicode string
- **allowed_aliases** (*list (str)*) – Script blocks aliases not to consider.

**Returns** Whether `string` is considered mixed-scripts or not.

**Return type** bool

### 4.1.4 confusable_homoglyphs.utils module

confusable_homoglyphs.utils.**delete**(*filename*)

Deletes a JSON data file if it exists.

confusable_homoglyphs.utils.**get**(*url*)

confusable_homoglyphs.utils.**load**(*filename*)

Loads a JSON data file.

**Returns** A dict.

**Return type** dict

confusable_homoglyphs.utils.**u**(*x*)

### 4.1.5 Module contents

CHAPTER 5

# Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

## 5.1 Types of Contributions

### 5.1.1 Report Bugs

Report bugs at https://github.com/vhf/confusable_homoglyphs/issues.

If you are reporting a bug, please include:

- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

### 5.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" is open to whoever wants to implement it.

### 5.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "feature" is open to whoever wants to implement it.

### 5.1.4 Write Documentation

confusable_homoglyphs could always use more documentation, whether as part of the official confusable_homoglyphs docs, in docstrings, or even on the web in blog posts, articles, and such.

### 5.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/vhf/confusable_homoglyphs/issues.

If you are proposing a feature:

- Explain in detail how it would work.

- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

## 5.2 Get Started!

Ready to contribute? Here's how to set up *confusable_homoglyphs* for local development.

1. Fork the *confusable_homoglyphs* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_username_here/confusable_homoglyphs.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv confusable_homoglyphs
$ cd confusable_homoglyphs/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 confusable_homoglyphs tests
$ python setup.py test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

## 5.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.7, 3.3, 3.4, and 3.5. Check https://travis-ci.org/vhf/confusable_homoglyphs/pull_requests and make sure that the tests pass for all supported Python versions.

CHAPTER 6

# Credits

## 6.1 Maintainer

- Victor Felder <victorfelder@gmail.com>

## 6.2 Contributors

None yet. Why not be the first? See: CONTRIBUTING.rst

CHAPTER 7

# History

CHAPTER 8

# 1.0.0 (2016)

Initial release.

## A



## C



## D



## F



## G



## I



## L



## U