**Правительство Российской Федерации**
**Федеральное государственное автономное образовательное учреждение высшего образования**

**Национальный исследовательский университет**
**«Высшая школа экономики»**

Факультет гуманитарных наук
Образовательная программа
«Фундаментальная и компьютерная лингвистика»

Воронов Михаил Кириллович

# Программная библиотека для лингвистической типологии на языке Python

*Python Library for Linguistic Typology*

Выпускная квалификационная работа студента 4 курса бакалавриата

| Академический руководитель образовательной программы | Научный руководитель |
| --- | --- |
| канд. филологических наук, доц. | канд. филологических наук, доц. |
| Ю.А.Ландер | Б.Орехов |

4 июня 2019 г.

Москва 2019

# Contents

# 1 Introduction

There are multiple linguistic tools for Python. Most of them concentrate on natural language processing (e.g. NLTK (Bird, Steven, Edward Loper and Ewan Klein 2009)). There are also some libraries that are there to assist linguistic research.

A good example of such package is LingPy (List, Greenhill, and Forkel 2017). It provides multiple calculation and visualisation algorithms for historical linguistics.

Another example of this kind of libraries is LingCorpora (Koshevoy et al. 2018). It allows to perform queries in multiple online text corpora. It is in active development at the moment and already supports more than 25 corpora.

However, I did not find any Python tools that allow to work with online linguistic databases.

Most of such databases are stored in Cross-Linguistic Linked Data format (Haspelmath and Forkel 2013). This specification also provides framework (Forkel et al. 2019) that allows creating CLLD apps. However, it does not provide user-friendly API for the stored data and cannot access databases from remote repositories.

There is a tool for Glottolog (Hammarström, Forkel, and Haspelmath 2019) that provides an API and console application for Glottolog (Forkel 2019). However, this tool requires a local copy of Glottolog data that takes more than 700 megabytes of storage.

Also, there seems to be no Python tools for linguistic interactive mapping and researchers have to use libraries such as Folium (Filipe et al. 2019b) which is a general tool for interactive mapping and is not designed for linguistic maps specifically.

So, the first gap that my package attempts to cover is lack of Python tools that provide an interface for online linguistic databases. The second gap is lack of Python tools designed for linguistic interactive mapping.

There is a package for the R programming language called 'lingtypology' (Moroz 2017). It provedes an API for linguistic databases, a tool to work with Glottolog data and a tool to create interactive linguistic maps. My package was inspired by this R library and I consider it to be its counterpart for Python. Therefore my package is also called 'lingtypology'.

One of the main purposes of my package is to provide a tool for easy reproducibility. Usually researchers manually find the data from multiple linguistic research. With my package it is possible to reproduce such research very quickly.

In the following chapters, I will describe technical aspects of 'lingtypology', provide documentation and several small studies to demonstrate its usability.

# 2 Project Description

In this section I will provide description for `lingtypology`. To avoid repetition I will not include documentation into this section, it is present in Chapter 3 'Usage'.

## 2.1 General

`lingtypology` is written in the Python programming language version 3.7 (Python Software Foundation 2019). It also supports versions 3.5 and 3.6. It is planned to keep maintaining all the versions of Python that are supported except for 2.7 branch. It is not supported due to its coming end of life in early 2020.

This project uses `git` distributed version control system. The source code of the project is stored in the remote repository (Voronov 2019).

## 2.2 Dependencies

`lingtypology` package requires a number of additional libraries.

- Folium and Branca (Filipe et al. 2019b). Folium is a Python wrapper for `leaflet` library for JavaScript (Agafonkin 2017). Branca is the additional package for Folium that allows editing HTML code of the maps while Folium works with JavaScript only.
- Pandas (Augspurger et al. 2019). Pandas introduces dataframes in Python.
- pyglottolog (Forkel 2019). pyglottolog application is used to extract necessary data from Glottolog.
- matplotlib (Caswell et al. 2019). Matplotlib is a tool for creating plots.
- jinja2. A template engine.
- colour. A library for colors.

## 2.3 Package

`lingtypology` package consists of different modules and data files.

### 2.3.1 Modules

- `__init.py__` contains imports and version.
- `maps.py` is the module for linguistic interactive mapping.
- `glottolog.py` contains a number of useful functions for Glottolog.
- `db_apis.py` contains API for multiple online databases.

### 2.3.2 Data Files

- `legend.html` HTML-template for map legends and title.
- `language_elevation_mapping.json` data on elevation for each language from Glottolog.
- `autotyp_lang_mapping.json` mapping from language IDs from Autotyp to languages from Glottolog. Taken from the R counterpart under the rule of GNU GPL license (Moroz 2017).
- CSV file that starts with `glottolog-languoids` contains some of Glottolog data. It is generated with `pyglottolog` application with `glottolog --repos=glottolog languoids` command.
- JSON file that starts with `glottolog-languoids` contains metadata for the CSV file above.

## 2.4 Interactive Maps

In this subsection I will describe `lingtypology.maps` module.

### 2.4.1 General

`lingtypology.maps` contains the `LingMap` object. This object has attributes and methods that allow to render interactive maps. This object stores the data that a user wants to be rendered and when `create_map` method is called all the data is processed and passed into `folium.map` object. Then it can be rendered as HTML.

For example, method `add_overlapping_features` applies different colors from `colors` atrribute to each feature, finds out the proper size of markers based on the amount of features for each language, creates `folium.CircleMarker` objects, adds popups and tooltips to the markers if necessary and then adds the markers to the map.

Also, `lingtypology.maps` contain several supplementary functions.

Full description of the functions and the `LingMap` class can be found in Chapter 3.

### 2.4.2 Elevation Data

`lingtypology` provides elevation data for each language in Glottolog (as of version 3.4). The source if this data is SRTM dataset (Jarvis A., H.I. Reuter, A. Nelson, E. Guevara 2008). It was processed using locally run Open Elevation API server (Lourenço and Developer66 2019).

### 2.4.3 Legend

Folium does not provide API to create a legend. Nevertheless, Lingtypology has a legend. It is created automatically based on the data that user passes to lingtypology.

The templated for the legend is based on the HTML code from here (Talbert 2018).

### 2.4.4 Strokes

The default appearance of standard `folium.CircleMarker` objects does not fit the needs of the package due to the wrong behaviour of strokes. Compare pictures 1 and 2.

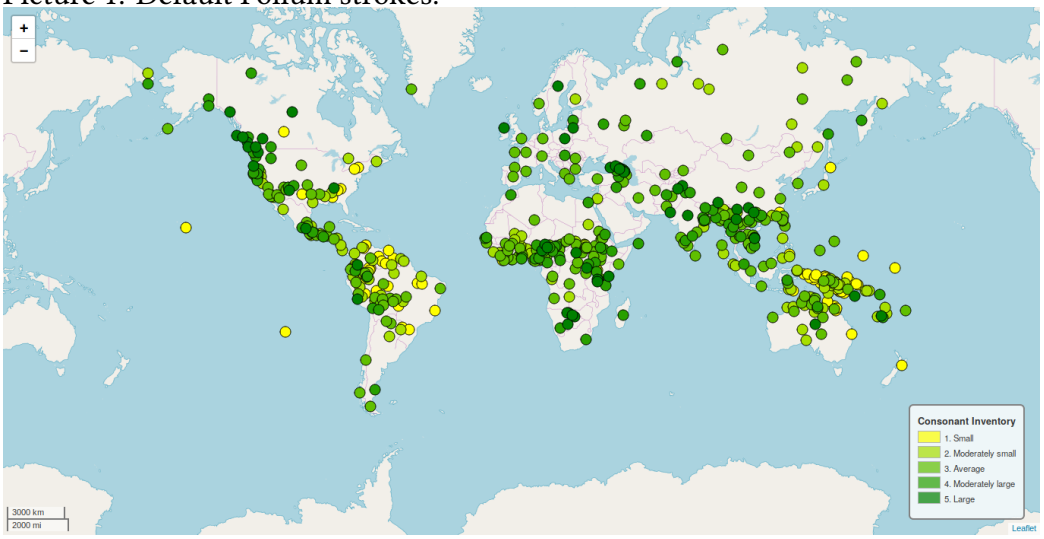
In the first picture strokes appear after each individual marker (the default behaviour of `folium`). In the second picture strokes appear after groups of markers. This bahaviour is impossible to achive using `folium.CircleMarker` objects by default. Therefore, instead of rendering one marker with stroke, `lingtypology` renders two markers: a marker without stroke and a black marker that 115% larger. Markers are not added to the map right away. They are rendered in the proper order:

1. Draw all black background markers.
2. Draw other markers.

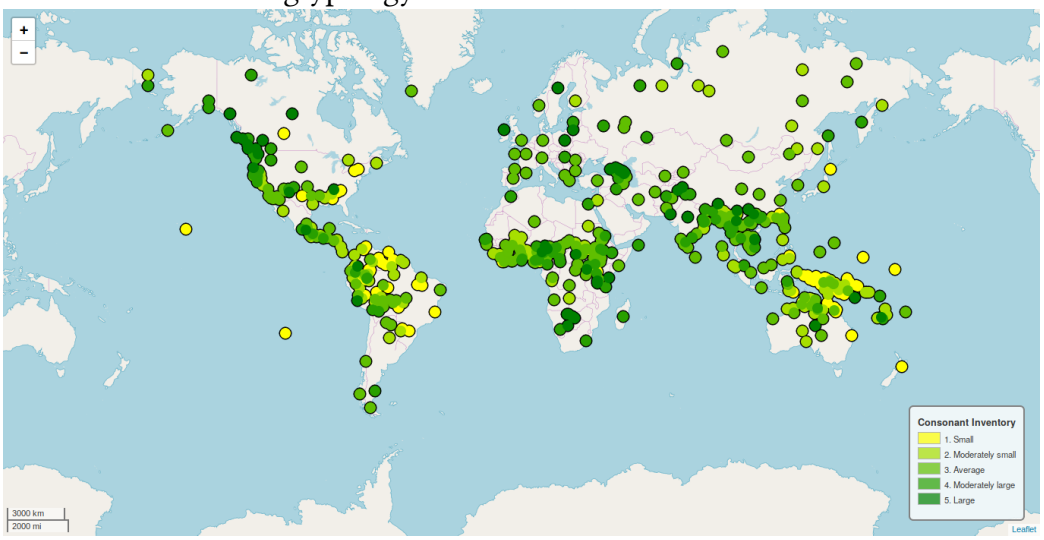
The way markers are rendered in the second picture is the default behaviour of `lingtypology`. However, the option to use the standard behaviour of `foilum` is present as well.

Both pictures were generated using `lingtypology`. Listings are in the attachment ('Listing 2' and 'Listing 1').

Picture 1. Default Folium strokes.




Picture 2. Default Lingtypology strokes.




### 2.4.5 Minicharts

I had a feature request to add functionality to draw minicharts instead of markers in the map.

Due to the fact that neither `folium` nor `folium.plugins` does not provide such functionality and I did not find a Leaflet plugin with such functionality, minicharts are rendered as SVG with Matplotlib (Caswell et al. 2019).

Matplotilib does provide functionality to render plots as SVG images and save

them as plain text files. Nevertheless, it does not provide documentation for methods that allow to get SVG as Python type `str`. Therefore, the plots rendered as SVG have to be caught with `io.StringIO` object from standard Python library.

Object `folium.Marker` allows passing HTML elements that define their appearance. However, it is not possible to pass the SVG as is: if there are newline symbols, the HTML file of the map will be broken. Therefore, all such symbols have to be removed from the SVG string.

Due to lack of unified API for different charts in Matplotlib, the number of available minicharts is limited. At the moment only pie-charts and bar-charts are supported.

During this research I received a request from one of the Folium developers to add an example of minicharts usage to the Folium gallery. My example is available in the ‘examples‘ directory of the Folium repository (Filipe et al. 2019b).

#### 2.4.6 PNG

Foium does not provide official support for rendering maps as PNG. Nevertheless, I found undocumented API that allows it (`folium.Map._to_png` method).

It was implemented into Lingtypology. However, this method requires additional application (Geckodriver) which is not included into main repositories of popular operating systems (Windows, macOS, Debian GNU/Linux, OpenSUSE etc.) and requires additional efforts to install. Due to this fact, this functionality of Lingtypology is marked as experimental until another way to implement is found. Also, this functionality requires additional Python dependency: selenium. It is used to render HTML as PNG.

### 2.5 Glottolog

Lingtypology uses Glottolog data at its core. Therefore, accessing it online each time would significantly slow down the package. Glottolog data necessary for the package is included in the package.

However, Glottolog data is updated continually. I update included Glottolog data with each new release of Lingtypology. If user requires newer version of the data before new release, I provide the instruction how to update it manually.

`lingtypology.glottolog` provides multiple functions to work with Glottolog data. Usage information may be found in Chapter 3.

## 2.6 Databases API

Lingtypology provides API for the following lingustic databases:

- WALS (Dryer and Haspelmath 2013).
- Autotyp (Bickel et al. 2017).
- AfBo (Seifart 2013).
- SAILS (Muysken et al. 2016).
- PHOIBLE (Moran and McCloy 2019)

### 2.6.1 General

Databases usually can be retrieved in CSV format. They are read into `pandas.DataFrame`, processed and returned to user in easy to read and use format.

`lingtypology.db_apis` contains classes for each database. The module attemts to provide a unified API for all datasets, so there are methods that work the same way for all the databases.

The moment when the database is downloaded depends on the database. In some cases the data is received right after the initialization of the class, in other cases the data is downloaded when `get_df` method is called.

If the data in the database is divided into different pages, tables etc., Lingtypology is able to process and merge several of them.

### 2.6.2 WALS

WALS: 'The World Atlas of Language Structures (WALS) is a large database of structural (phonological, grammatical, lexical) properties of languages gathered from descriptive materials (such as reference grammars) by a team of 55 authors.' (Dryer and Haspelmath 2013). The data from wals is retrieved from multiple web-pages that contain data for each chapter when `get_df` method is called.

### 2.6.3 Autotyp

Autotyp is database that contains of multiple modules. Each module represents a grammatical feature (e.g. Agreeement), it contains information on this feature for various languages (Bickel et al. 2017). The data is downloaded when `get_df` method is called.

### 2.6.4 AfBo

AfBo: A world-wide survey of affix borrowing (Seifart 2013). AfBo contains information about borrewed affixes in different languages. It provides data in ZIP archive with CSV files. The data is downloaded with initialization of the class.

### 2.6.5 SAILS

'The South American Indigenous Language Structures (SAILS) is a large database of grammatical properties of languages gathered from descriptive materials (such as reference grammars)' (Muysken et al. 2016). Like in the case of AfBo, SAILS data is available in ZIP archive. The data is downloaded with initialization of the class.

### 2.6.6 PHOIBLE

'PHOIBLE is a repository of cross-linguistic phonological inventory data, which have been extracted from source documents and tertiary databases and compiled into a single searchable convenience sample.' (Moran and McCloy 2019). Unlike other databases supported by Lingtypology, PHOIBLE is not a unified dataset. It contains data of the following datasets: UPSID, SPA, AA, PH, GM, RA, SAPHON.

# 3 Usage

In this part I will provide full guide for 'lingtypology'. It is divided into four parts: Installation, Interactive Maps, Glottolog functions and the API for databases.

## 3.1 Installation

The package is uploaded to PyPI software repository, therefore it can be installed with the `pip` utility with the following command:

```
pip3 install lingtypology --user
```

## 3.2 Interactive Maps

Interactive maps can be created with `lingtypology.maps` module.

### 3.2.1 LingMap Class

*class* lingtypology.**LingMap** *(languages=[], glottocode=False)*
Bases: *object*
**Parameters**:

- **languages**: *list* or *pandas.Series* of strings, default *[]*.

  A list of languages. The language names should correspond to their names from Glottolog unless you use *add_custom_coordinates* method.

  Instead of language names you could use Glottocodes (language ID in Glottolog). In this case you need to set *glottocode* parameter to true.

- **glottocode**: *bool*, default *False*.

  Whether to treat *languages* as Glottocodes.

**Attributes**:

- **tiles**: *str*, default *'OpenStreetMap'*

  Tiles for the map. You can use one of these tiles (list of tiles is borrowed from the Folium Documentation (Filipe et al. 2019a)):

  - "OpenStreetMap"
  - "Mapbox Bright" (Limited levels of zoom for free tiles)
  - "Mapbox Control Room" (Limited levels of zoom for free tiles)
  - "Stamen" (Terrain, Toner, and Watercolor)
  - "Cloudmade" (Must pass API key)

  - "Mapbox" (Must pass API key)
  - "CartoDB" (positron and dark_matter)
  - or pass the custum URL.

- **start_location**: *(float, float)* or *str*, default *(0, 0)*

  Coordinates of the start location for the map *(latitude, longitude)* or a text shortcut. List of available shortcuts: *"Central Europe", "Caucasus", "Australia & Oceania", "Papua New Guinea", "Africa", "Asia", "North America", "Central America", "South America"*.

- **start_zoom**: *int*, default *2*

  Initial zoom level. Bypassed if you are using a shortcut *start_location*.

- **control_scale**: *bool*, default *True*

  Whether to add control scale.

- **prefer_canvas**: *bool*, default *False*

  Use canvas instead of SVG. If set to True, the map may be more responsive in case you have a lot of markers.

- **base_map**: *folium.Map*, default *None*

  In case you want to draw something on particular *folium.Map*.

- **title**: *str*, default *None*

  You can add a title to the map.

- **legend**: *bool*, default *True*

  Whether to add legend for features (*add_features* method).

- **stroke_legend**: *bool*, default *True*.

  Whether to add legend for stroke features (*add_stroke_features* method)

- **legend_title**: *str*, default *'Legend'*

  Legend title.

- **stroke_legend_title**: *str*, default *'Legend'*

  Stroke legend title.

- **legend_position**: *str*, default *'bottomright'*

  Legend position. Available values: 'right', 'left', 'top', 'bottom', 'bottomright', 'bottomleft', 'topright', 'topleft'.

- **stroke_legend_position**: *str*, default *'bottomleft'*

  Stroke legend position. Available values: 'right', 'left', 'top', 'bottom', 'bottomright', 'bottomleft', 'topright', 'topleft'.

- **colors**: *list* of html codes for colors (*str*).

  Colors that represent features. You can either use the 20 default colors or set yours.

- **stroke_colors**: *list* of html codes for colors (*str*)

  Colors that represent additional (stroke) features.

- **shapes**: *list* of characters (*str*)

  If you use shapes instead of colors, you can either use the default shapes or set yours. Shapes are Unicode symbols.

- **stroked**: *bool*, default *True*

  Whether to add stroke to markers.

- **unstroked**: *bool*, default *True*

  If set to True, circle marker will merge if you zoom out without stroke between them. It multiplies the number of markers by 2. For better performance set it to False. More information and examples in Chapter 2, Paragraph 4.4.

- **languages_in_popups**: *bool*, default *True*

  Whether to show links to Glottolog website in popups.

- **control**: *bool*, default *False*

  Whether to add LayerControls and group by features.

- **stroke_control**: *bool*, default *False*

  Whether to add LayerControls and group by stroke features.

- **control_position**: *str*, default *'topright'*

  Position of LayerControls. May be 'topleft', 'topright', 'bottomleft' or 'bottomright'.

- **colormap_colors**: *tuple*, default *('white', 'green')*

  Colors for the colormap.

**Methods**:

- **add_custom_coordinates** *(custom_coordinates)*

  Set custom coordinates. By default cooordinates for the languages are taken from the Glottolog database. If you have coordinates and want to use them, use this function.

  **Parameter** custom_coordinates: *list* of custom_coordinates (*tuples*)

  Length of the list should equal to length of languages.

- **add_features***(features, radius=7, opacity=1, numeric=False, control=False, use_shapes=False)*

  Add features to the map.

  **Parameters**:

  - **features**: *list*

    List of features. Amount of features should be equal to the amount of languages. By default, if you add features, a legend will appear. To shut it down set *legend* attribute to *False*. To change the title of the legend use *legend*$_t$*itle* attribute. To change legend position use use *legend_position* attribute.

  - **radius**: *int*, default *7*

    Marker radius.

  - **numeric**: *bool*, default *False*

    Whether to assign different color to each feature (*False*), or to assign a color from colormap (*True*). You can set it to True only in case your features are numeric and stroke features are not given. To change the default colors of the color scale use *colormap_colors* attribute.

  - **control**: *bool*, default *False*

    Whether to add LayerControls to the map. It allows interactive turning on/off given features.

  - **use_shapes**: *bool*, default *False*

    Whether to use shapes instead of colors. This option allows to represent features as shapes. Shapes are Unicode charaters like ⊠ or ⊠. You can replace or add to default symbols by changing *shapes* attribute. If colors are not a viable option for you, you can set this option to True.

- **add_stroke_features***(features, radius=12, opacity=1, numeric=False, control=False)*:

  Add additional set of features that look like strokes around markers.

- **features**: *list*

  List of additional features. Amount of features should be equal to the amount of languages. By default, if you add stroke features, a legend will appear. To shut it down set *stroke_legend* attribute to *False*. To change the title of the legend use *stroke_legend_title* attribute. To change legend position use use *stroke_legend_position* attribute.

- **radius**: *int*, default *12*

  Marker radius. Note that this radius is absolute as well.

- **control**: *bool*, default *False*

  Whether to add LayerControls to the map. It allows interactive turning on/off given stroke features.

- **add_overlapping_features** *(features, radius=7, radius_increment=4, mapping=None)*:

  Add overlapping features. For example, if you want to draw on map whether language 'is ergative', 'is slavic', 'is spoken in Russia'. It will draw several markers of different size for each feature.

  **Parameters**:

  - **features**: *list* of lists

    List of features. Amount of features should be equal to the amount of languages.

  - **radius**: *int*, default 7

    Radius of the smallest circle.

  - **radius_increment**: *int*, default *4*

    Step by which the size of the marker for each feature will be incremented.

  - **mapping**: *dict*, default *None*

    Mapping for the legend.

- **add_minicharts** *(*minicharts, typ='pie', size=0.6, names=None, textprops=None, labels=False, colors=[], startangle=90)*:

  Create minicharts using Maplotlib.

  **Parameters**:

  - ***minicharts**: list-like objects

    Data for minicharts. Two list-like objects.

- **typ**: *str*, default *pie*
  Type of the minicharts. Either *pie* or *bar*.
- **size**: *float*
  Size of the minicharts.
- **texprops**: *dict*, default *None*
  Textprops for Matplotlib.
- **labels**: *bool*, default *False*
  Whether to display labels.
- **colors**: *list*, default *[]*
  Minicharts colors.
- **startange**: *int*, default *90*
  Start angle of pie-charts (pie-charts only).

- **add_heatmap**: *(heatmap=[])*

  Add heatmap.

  **Parameter** heatmap: list-like object with tuples

  Coordinates for the heatmap. To create a heatmap-only map, do not pass any languages.

- **add_popups** *(popups, parse_html=False)*

  Add popups to markers.

  **Parameters**:

  - **popups**: *list* of strings List of popups. Length of the list should equal to length of languages.
  - **parse_html**: *bool*, default *False* By default (*False*) you can add HTML elements. If you need to add full HTML pages to popups, you need to set the option to *True*.

- **add_tooltips** *(tooltips)*:

  Add tooltips to markers.

  **Parameter** tooltips: *list* of strings

  List of tooltips. Length of the list should equal to length of languages.

- **add_minimap** *(position='bottomleft', width=150, height=150, collapsed_width=25, collapsed_height=25, zoom_animation=True)*

  Add minimap.

  **Parameters**:

  - **position**: *str*, default *'bottomleft'*
  - **width** and **height**: *int*, default *150*
  - **collapsed_width** and **collapsed_height**: *int*, default *25*
  - **zoom_animation**: *bool*, default *True*
    You can disable zoom animation for better performance.

- **create_map** *()*

  Create the map.

  **Returns** folium.Map

- **render***()*

  **Returns** the HTML code as *str*

- **save** *(path)*

  Save the map as HTML.

  **Parameter** path, *str*

  Path to the file.

- **save_static** *(path=None)*

  Save the map as PNG. Experimental function. Requires additional Python package Selenium and additional application Geckodriver.

  If path is not given **returns** the PNG as *bytes*.

### 3.2.2 Functions

*function* lingtypology.**merge** *(*maps)*
Accepts *LingMap* objects and creates map of them.
**Parameter** *maps: *LingMap* objects.
**Returns** foilum.Map

*function* lingtypology.**get_elevations** *(languages)*

Get data on elevation for languages. More information in Chapter 2, Paragraph 4.2.

**Parameter** lagnauges: *list* of strings
**Returns** list

*function* lingtypology.**gradient** *(iterations, color1='white', color2='green')*
Creates color gradient of given length.
**Returns** list of HEX-colors

### 3.3 Glottolog

# 4 Bibliography

## References


Agafonkin, Vladimir (2017). *Leaflet API reference*. URL: https://leafletjs.com/reference-1.5.0.html.

Augspurger, Tom et al. (2019). *pandas: powerful Python data analysis toolkit*. URL: http://pandas.pydata.org/pandas-docs/stable.

Bickel, Balthasar et al. (2017). *The AUTOTYP typological databases. Version 0.1.0*. URL: https://github.com/autotyp/autotyp-data/tree/0.1.0.

Bird, Steven, Edward Loper and Ewan Klein (2009). *Natural Language Processing with Python*.

Caswell, Thomas A et al. (May 2019). *matplotlib/matplotlib v3.1.0*. DOI: 10.5281/zenodo.2893252. URL: https://doi.org/10.5281/zenodo.2893252.

Dryer, Matthew S. and Martin Haspelmath, eds. (2013). *WALS Online*. Leipzig: Max Planck Institute for Evolutionary Anthropology. URL: https://wals.info/.

Filipe et al. (2019a). *Folium*. URL: https://python-visualization.github.io/folium/index.html.

— (May 2019b). *python-visualization/folium: v0.9.1*. DOI: 10.5281/zenodo.3229045. URL: https://doi.org/10.5281/zenodo.3229045.

Forkel, Robert (Apr. 2019). *clld/pyglottolog: Glottolog API*. DOI: 10.5281/zenodo.2620250. URL: https://doi.org/10.5281/zenodo.2620250.

Forkel, Robert et al. (Apr. 2019). *clld/clld: clld - a toolkit for cross-linguistic databases*. DOI: 10.5281/zenodo.2635592. URL: https://doi.org/10.5281/zenodo.2635592.

Hammarström, Harald, Robert Forkel, and Martin Haspelmath (Apr. 2019). *clld/glottolog: Glottolog database 3.4*. DOI: 10.5281/zenodo.2620814. URL: https://doi.org/10.5281/zenodo.2620814.

Haspelmath, Martin and Robert Forkel (2013). *CLLD – Cross-Linguistic Linked Data*. URL: https://clld.org/.

Jarvis A., H.I. Reuter, A. Nelson, E. Guevara (2008). *Hole-filled seamless SRTM data V4*. URL: http://srtm.csi.cgiar.org.

Koshevoy, Alexey et al. (Feb. 2018). *lingcorpora/lingcorpora.py: initial release*. DOI: 10.5281/zenodo.1172457. URL: https://doi.org/10.5281/zenodo.1172457.

List, Johann-Mattis, Simon Greenhill, and Robert Forkel (2017). *LingPy. A Python library for quantitative tasks in historical linguistics*. Jena. DOI: https://

doi.org/10.5281/zenodo.1065403. URL: http://lingpy.org.

Lourenço, João Ricardo and Developer66 (2019). *Open-Elevation - Remake*. URL: https://github.com/Developer66/open-elevation.

Moran, Steven and Daniel McCloy, eds. (2019). *PHOIBLE 2.0*. Jena: Max Planck Institute for the Science of Human History. URL: https://phoible.org/.

Moroz, George (2017). *lingtypology: easy mapping for Linguistic Typology*. DOI: 10.5281/zenodo.1289471. URL: https://CRAN.R-project.org/package=lingtypology.

Muysken, Pieter et al. (2016). *South American Indigenous Language Structures (SAILS) Online*. URL: http://sails.clld.org.

Python Software Foundation (2019). *The Python Language Reference*. URL: https://docs.python.org/3.7/reference/.

Seifart, Frank, ed. (2013). *AfBo: A world-wide survey of affix borrowing*. Leipzig: Max Planck Institute for Evolutionary Anthropology. URL: https://afbo.info/.

Talbert, Colin (2018). *How does one add a legend (categorical) to a folium map*. URL: https://nbviewer.jupyter.org/gist/talbertc-usgs/18f8901fc98f109f2b71156cf3ac81cd.

Voronov, Michael (May 2019). *lingtypology: a Python tool for linguistic interactive mapping*. DOI: 10.5281/zenodo.2669068. URL: https://doi.org/10.5281/zenodo.2669068.

## 5 Attachment

**Listing 1. Wals 1A map with default Lingtypology strokes.**

```
import os
os.chdir('images')
import lingtypology

wals_page = lingtypology.db_apis.Wals('1a', '2a').get_df()
m = lingtypology.LingMap(wals_page.language)
m.add_custom_coordinates(wals_page.coordinates)
m.add_features(wals_page._1A)
m.legend_title = 'Consonant Inventory'
m.colors = lingtypology.gradient(5, 'yellow', 'green')
m.save_static('LingtypologyStrokeAppearance.png')
```

**Listing 2. Wals 1A map with default Folium strokes.**

```
import os
os.chdir('images')
import lingtypology

wals_page = lingtypology.db_apis.Wals('1a', '2a').get_df()
m = lingtypology.LingMap(wals_page.language)
m.add_custom_coordinates(wals_page.coordinates)
m.add_features(wals_page._1A)
m.legend_title = 'Consonant Inventory'
m.colors = lingtypology.gradient(5, 'yellow', 'green')
m.unstroked = False
m.save_static('FoliumStrokeAppearance.png')
```