PyPop API Reference

Developer documentation

Release 1.4.1

Alexander K. Lancaster

Mar 23, 2026

Contents

1 API changes 1

2 Package introduction 2

3 Submodules 3
- PyPop.citation . . . 3
- PyPop.command_line_interface . . . 3
- PyPop.datatypes . . . 5
- PyPop.filters . . . 8
- PyPop.haplo . . . 13
- PyPop.hardyweinberg . . . 16
- PyPop.homozygosity . . . 19
- PyPop.parsers . . . 22
- PyPop.popaggregate . . . 25
- PyPop.popanalysis . . . 27
- PyPop.popmeta . . . 29
- PyPop.pypop . . . 29
- PyPop.randombinning . . . 29
- PyPop.utils . . . 30
- PyPop.xslt . . . 39

4 Deprecated Submodules 41
- PyPop.arlequin . . . 41

5 Attributes 43

6 Exceptions 43

7 Functions 43

8 Package Contents 43

9 GNU Free Documentation License 44

Python Module Index 47

Index 49

***Documenting API for release* 1.4.1 *of PyPop*.**

Document revision: 1.4.1.post2+gfae62b262

This API reference guide for PyPop is automatically generated from the 1.4.1 source code via sphinx-autoapi[1].

Copyright © 2025 PyPop contributors

License terms Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections no Front-Cover Texts and no Back-Cover Texts. A copy of the license is included in the License chapter. (*GNU Free Documentation License*)

References to the *User Guide* can be found in the *PyPop User Guide*: HTML[2]| PDF[3].

1 API changes

In PyPop 1.4.0, modules have been renamed to follow the lower-case convention of PEP8[4]. In addition to lowercasing, some have further renaming to clarify their purpose and follow standard conventions. Backwards-compatibile bindings have been created that allow end-user Python scripts using the PyPop API to continue to work with the old module names. However such use will raise a *`PyPopModuleRenameDeprecationWarning`* (a custom `DeprecationWarning`[5]). In the following minor release, 1.5.0, the warnings will become more visible `UserWarning`[6]. These bindings will be completely removed in the next major release.

Note:

Command-line users of `pypop` will not be affected by these changes, which are completely internal, scripts will continue to work with no changes needed in any configuration files.

Below are the list of all API changes, including removals and any other ongoing API deprecations, and notifications of upcoming removals.

Changed in version 1.4.0

Changed in version 1.4.0: The following modules were renamed or refactored:

- `PyPop.CommandLineInterface` → *`PyPop.command_line_interface`* Lowercased for PEP8 compliance; underscores separate readable words.
- `PyPop.DataTypes` → *`PyPop.datatypes`* Lowercased for PEP8 compliance and consistency with plural naming for data structures.
- `PyPop.Filter` → *`PyPop.filters`* Lowercased and clarified plural form since module defines multiple filter functions.
- `PyPop.Haplo` → *`PyPop.haplo`* Lowercased for PEP8 compliance.
- `PyPop.HardyWeinberg` → *`PyPop.hardyweinberg`* Lowercased for PEP8 compliance.
- `PyPop.Homozygosity` → *`PyPop.homozygosity`* Lowercased for PEP8 compliance.
- `PyPop.Main` → *`PyPop.popanalysis`* Lowercased and renamed for clarity; represents per-population analysis rather than script entry point.
- `PyPop.Meta` → *`PyPop.popaggregate`* Lowercased and renamed for clarity; aggregates results across populations, not 'metadata'.
- `PyPop.ParseFile` → *`PyPop.parsers`* Lowercased for and renamed for clarity: module parses multiple file types, not just one file.
- `PyPop.RandomBinning` → *`PyPop.randombinning`* Lowercased for PEP8 compliance.
- `PyPop.Utils` → *`PyPop.utils`* Lowercased for PEP8 compliance.

Removed in version 1.4.0

Removed in version 1.4.0: The following modules or classes were removed:

- `PyPop.GUIApp` Obsolete, never fully implemented a full `wxPython` UI. Replaced by built-in Tkinter file-picker
- `PyPop.Utils.Index` Obsolete, replaced with `collections.OrderedDict`[7] with it's own `Index` class
- `PyPop.Utils.OrderedDict` Obsolete, replaced with `collections.OrderedDict`[8]

Deprecated since version 1.0.0

Deprecated since version 1.0.0: The following modules were marked deprecated:

- `PyPop.Arlequin` → *`PyPop.arlequin`* **Scheduled for removal in 1.5.0.** Lowercased for PEP8 compliance.

Deprecated since version 0.6.0

Deprecated since version 0.6.0: The following modules were marked deprecated:

- *`PyPop.datatypes.AlleleCounts`* **Scheduled for removal in 1.4.2.** The *`Genotypes`* class now holds allele count data as pseudo-genotype matrix.

2 Package introduction

PyPop is a framework for performing population genetics analyses.

PyPop was originally designed as an end-to-end pipeline that reads configuration files and datasets and produces standardized outputs. While the primary workflow is file-based, most internal functionality is exposed as Python modules and classes.

Important

Updates to PyPop's API to better expose and streamline "library" access to PyPop's functionality in end-user programs is still a work-in-progress. Although this API is intended to serve end-users and developers of PyPop, parts of it are not yet optimized for end-users.

Driving PyPop programmatically can be done via the *`popanalysis`* and *`popaggregate`* modules. In the example below, we run an simple analysis on a single input `.pop` file and generate output TSV files. There are two main steps:

1. Create the `ConfigParser`[9] instance (see configuration file section in the *PyPop User Guide* for the description of the configuration options), supply this to the *`Main`* class, along with an input `.pop` file, to perform the analysis.

2. Next get the name of output XML file from the generated `Main` instance, and pass it to the *`Meta`* to generate TSV output files.

```
>>> from PyPop.popanalysis import Main
>>> from configparser import ConfigParser
>>>
>>> config = ConfigParser()
>>> config.read_dict({
...     "ParseGenotypeFile": {"validSampleFields": "*a_1\n*a_2"},
...     "HardyWeinberg": {"lumpBelow": "5"}})
>>>
>>> pop_contents = '''a_1\ta_2
... 01:01\t02:01
... 02:10\t03:01:02'''
>>> with open("my.pop", "w") as f:
...     _ = f.write(pop_contents)
...
>>> application = Main(
...     config=config,
...     fileName="my.pop",
...     version="fake",
... )
LOG: no XSL file, skipping text output
LOG: Data file has no header data block
>>> outXML = application.getXmlOutPath()
>>> from PyPop.popaggregate import Meta
>>> _ = Meta (TSV_output=True, xml_files=[outXML])
./1-locus-hardyweinberg.tsv
./1-locus-summary.tsv
./1-locus-allele.tsv
./1-locus-genotype.tsv
```

See also

The PyPop API examples in the *PyPop User Guide* for a more detailed breakdown of use of the API.

3 Submodules

PyPop.citation

Module for generating citation formats.

Attributes

`citation_output_formats`	Valid citation output formats supported by `cffconvert`

Functions

`convert_citation_formats`(build_lib, citation_path)	Generate all supported citation formats.

Module Contents

`citation_output_formats = ['apalike', 'bibtex', 'endnote', 'ris', 'codemeta', 'cff', 'schema.org', 'zenodo']`

Valid citation output formats supported by `cffconvert`

`convert_citation_formats`(*build_lib*, *citation_path*)

Generate all supported citation formats.

Parameters

- **`build_lib`** (*`str`*) – path to build directory when creating package
- **`citation_path`** (*`str`*) – path to standard `CITATION.cff` file

PyPop.command_line_interface

Command-line interface for PyPop scripts.

Classes

CitationAction	A custom `argparse` `Citation` action to read the appropriate citation file format.

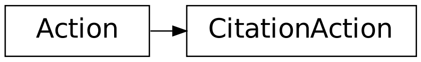
Functions

get_parent_cli([version, copyright_message])	Command-line options common to all scripts.
get_pypop_cli([version, copyright_message])	Command-line options for `pypop` script.
get_popmeta_cli([version, copyright_message])	Command-line options for `popmeta` script.

Module Contents

class CitationAction(*option_strings*, *dest*, *nargs=None*, *const=None*, *default=None*, *type=None*, *choices=None*, *required=False*, *help=None*, *metavar=None*)

Bases: `argparse.Action`[10]



A custom `argparse` `Citation` action to read the appropriate citation file format.

__call__(*parser*, *_*, *values*, *_option_string=None*)

get_parent_cli(*version=''*, *copyright_message=''*)

Command-line options common to all scripts.

Parameters

- **version** (*str*) – Software version.
- **copyright_message** (*str*) – Override the copyright message.

Returns

A tuple of:

- parent_parser (argparse.ArgumentParser): The base parser.
- ihwg_args (tuple): Options for the IHWG module.
- phylip_args (tuple): Options for the Phylip module.
- common_args (tuple): Common options.
- prefix_tsv_args (tuple): TSV prefix options.

Return type

tuple

get_pypop_cli(*version=''*, *copyright_message=''*)

Command-line options for `pypop` script.

Parameters

- **version** (*str*) – software version
- **copyright_message** (*str*) – override the copyright message

Returns

parser for `pypop`

Return type

argparse.ArgumentParser[11]

get_popmeta_cli(*version=''*, *copyright_message=''*)

Command-line options for `popmeta` script.

Parameters

- **version** (*str*) – software version
- **copyright_message** (*str*) – override the copyright message

Returns
parser for `popmeta`

Return type
argparse.ArgumentParser[12]

PyPop.datatypes

Data structures storing genotype and allele count data.

Classes

Genotypes	Stores genotypes and caches basic genotype statistics.
AlleleCounts	Deprecated class to store information in allele count form.

Functions

checkIfSequenceData(matrix)	Heuristic check to determine whether we are analysing sequence.
getMetaLocus(locus, isSequenceData)	Get the overall locus that this sequence belongs to.
getLocusPairs(matrix, sequenceData)	Get locus pairs for a given matrix.
getLumpedDataLevels(genotypeData, locus, lumpLevels)	Get lumped data for a specific locus.

Module Contents

`class Genotypes`(*matrix=None*, *untypedAllele='****'*, *unsequencedSite=None*, *allowSemiTyped=0*)

Stores genotypes and caches basic genotype statistics.

Parameters

- **matrix** (`StringMatrix`) – The `StringMatrix` to be converted into a `Genotype` instance
- **untypedAllele** (*str*) – The placeholder for an untyped allele site
- **unsequencedSite** (*bool*) – The identifier used for an unsequenced site (only used for sequence data)
- **allowSemiTyped** (*int*) – Whether or not to allow individuals that are typed at only one allele

getLocusList()

Get the list of loci.

Note

The returned list filters out all loci that consist of individuals that are all untyped. The order of returned list is now fixed for the lifetime of the object.

Returns
The list of loci.

Return type
list

getAlleleCount()

Allele count statistics for all loci.

Returns
a map of tuples where the key is the locus name. Each tuple is a triple, consisting of a map keyed by alleles containing counts, the total count at that locus and the number of untyped individuals.

Return type
dict

getAlleleCountAt(*locus*, *lumpValue=0*)

Get allele count for given locus.

Parameters

- **locus** (*str*) – locus

- **lumpValue** (*int*) – the specified amount of lumping (Default: 0)

Returns
a tuple consisting of a map keyed by alleles containing counts, the total count at that locus, and number of untyped individuals.

Return type
tuple

serializeSubclassMetadataTo(*stream*)

Serialize subclass-specific metadata.

Specifically, total number of individuals and loci and population name.

Parameters
stream (TextOutputStream) – the stream used for output.

serializeAlleleCountDataAt(*stream*, *locus*)

Serialize locus count data for a specific locus.

Specifically, total number of individuals and loci and population name.

Parameters

- **stream** (TextOutputStream) – the stream used for output
- **locus** (*str*) – locus

serializeAlleleCountDataTo(*stream*)

Serialize allele count data for a specific locus.

Parameters
stream (TextOutputStream) – the stream used for output

Returns
always returns 1

Return type
int

getLocusDataAt(*locus*, *lumpValue=0*)

Get the genotyped data for specified locus.

Note

The returned list has filtered out all individuals that are untyped at either chromosome. Data is sorted so that `allele1 < allele2`, alphabetically

Parameters

- **locus** (*str*) – locus to use
- **lumpValue** (*int*) – the specified amount of lumping (Default: `0`).

Returns
a list genotypes consisting of 2-tuples which contain each of the alleles for that individual in the list.

Return type
list

getLocusData()

Get the genotyped data for all loci.

Returns
keyed by locus name of lists of 2-tuples as defined by *getLocusDataAt()*

Return type
dict

getIndividualsData()

Get data for all individuals.

Returns
`StringMatrix` for all individuals

Return type
StringMatrix

`class AlleleCounts`(*alleleTable=None*, *locusName=None*)

Deprecated class to store information in allele count form.

Deprecated since version 0.6.0

Deprecated since version 0.6.0: this class is now obsolete, the *`Genotypes`* class now holds allele count data as pseudo-genotype matrix.

`serializeSubclassMetadataTo`(*stream*)

Serialize subclass-specific metadata.

Specifically, total number of alleles and loci.

`serializeAlleleCountDataAt`(*stream*, *locus*)

`getAlleleCount`()

`getLocusName`()

`checkIfSequenceData`(*matrix*)

Heuristic check to determine whether we are analysing sequence.

Note

The regex matches loci of the form `A_32` or `A_-32`

Parameters

`matrix` (`StringMatrix`) – matrix to check

Returns

if sequence, return `1`, otherwise `0`

Return type

int

`getMetaLocus`(*locus*, *isSequenceData*)

Get the overall locus that this sequence belongs to.

Parameters

- **`locus`** (*`str`*) – Locus of interest.
- **`isSequenceData`** (*`bool`*) – whether this locus is sequence data

Returns

The locus name, or `None` if not sequence data.

Return type

str

`getLocusPairs`(*matrix*, *sequenceData*)

Get locus pairs for a given matrix.

Parameters

- **`matrix`** (`StringMatrix`) – matrix
- **`sequenceData`** (*`bool`*) – is this sequence data?

Returns

Returns a list of all pairs of loci from a given `StringMatrix`.

Return type

list

`getLumpedDataLevels`(*genotypeData*, *locus*, *lumpLevels*)

Get lumped data for a specific locus.

Parameters

- **`genotypeData`** (`Genotypes`) – genotype data to query
- **`locus`** (*`str`*) – the locus
- **`lumpLevels`** (*`list`*) – a list of integers representing lumping levels

Returns

a dictionary of tuples:

- locusData: keyed by locus
- alleleCount:

Return type
dict

PyPop.filters

Filters for pre-filtering of data files before analysis.

This module includes filters that modify or otherwise transform the input data before being passed to PyPop analysis.

Exceptions

SubclassError	Customized exception if a subclass doesn't implement required methods.

Classes

Filter	Abstract base class for all Filters.
PassThroughFilter	A filter that doesn't change input data.
AnthonyNolanFilter	Filters data via Anthony Nolan's allele call data.
BinningFilter	Filters original data into "bins".
AlleleCountAnthonyNolanFilter	Filters data with an allelecount less than a threshold.

Module Contents

exception SubclassError

Bases: Exception[13]

Customized exception if a subclass doesn't implement required methods.

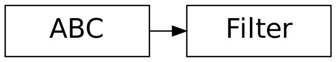
Initialize self. See help(type(self)) for accurate signature.

__str__()

Returns a warning to subclass.

class Filter

Bases: abc.ABC[14]



Abstract base class for all Filters.

abstractmethod doFiltering(*matrix=None*)

abstractmethod startFirstPass(*locus*)

abstractmethod checkAlleleName(*alleleName*)

abstractmethod addAllele(*alleleName*)

abstractmethod endFirstPass()

abstractmethod startFiltering()

abstractmethod filterAllele(*alleleName*)

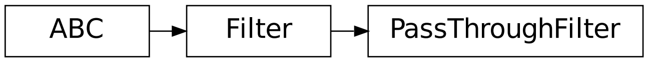
abstractmethod endFiltering()

abstractmethod writeToLog(*logstring=None*)

abstractmethod cleanup()

class PassThroughFilter

Bases: *Filter*



A filter that doesn't change input data.

doFiltering(*matrix=None*)

startFirstPass(*locus*)

checkAlleleName(*alleleName*)

addAllele(*alleleName*)

endFirstPass()

startFiltering()

filterAllele(*alleleName*)

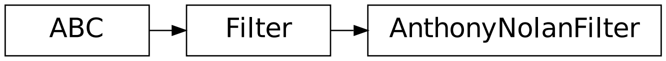
endFiltering()

writeToLog(*logstring=None*)

cleanup()

class AnthonyNolanFilter(*directoryName=None*, *remoteMSF=None*, *alleleFileFormat='msf'*, *preserveAmbiguousFlag=0*, *preserveUnknownFlag=0*, *preserveLowresFlag=0*, *alleleDesignator='*'*, *logFile=None*, *untypedAllele='****'*, *unsequencedSite='#'*, *sequenceFileSuffix='_prot'*, *filename=None*, *numDigits=4*, *verboseFlag=1*, *sequenceFilterMethod='strict'*)

Bases: *Filter*



Filters data via Anthony Nolan's allele call data.

Allele call data files can be of either `txt` or `msf` formats.

- `txt` files available at http://www.anthonynolan.com
- `msf` files available at ftp://ftp.ebi.ac.uk/pub/databases/imgt/mhc/hla/

Base class parameters.

Parameters

- **directoryName** (*str*) – directory that AnthonyNolan allele data is located
- **remoteMSF** (*str*) – Specifies the version (tag) of the remote `msf` directory in the IMGT-HLA GitHub repo[15]. If present, the remote MSF files for the specified version will be downloaded on-demand, and cached for later reuse
- **alleleFileFormat** (*str, optional*) – file format, can be `txt` or `msf` (default). Use of `msf` files is required in order to translate allele codes into polymorphic sequence data.
- **preserveAmbiguousFlag** (*int, optional*) – If set to `0` (default) then ambiguitity is removed (e.g. `010101/0102/010301` will truncate this to `0101`). To preserve the ambiguity, set the option to 1 (for this example, it will result in a filtered allele "name" of `0101/0102/0103`)
- **preserveUnknownFlag** (*int, optional*) – If set to `0` (default) replace unknown alleles with the `untypedAllele` designator. To keep unrecognized allele names set to 1.
- **preserveLowresFlag** (*int, optional*) – This option is similar to `preserveUnknownFlag`, but only applies to lowres alleles. If set to 1, PyPop will keep allele names that are shorter than the default allele name length, usually 4 digits long. But if `preserveUnknownFlag` is set, this option has no effect, because all unknown alleles are preserved.
- **alleleDesignator** (*str, optional*) – the designator used to indicate a locus name (default *),
- **logFile** (*str, optional*) – log file
- **untypedAllele** (*str, optional*) – defaults to ****

- **unsequencedSite** (*str, optional*) – defaults to #
- **sequenceFileSuffix** (*str, optional*) – Suffix for file names used for finding sequences each allele. (e.g.,, if the file for locus `A` is `A_prot.msf`, then keep the default be `_prot`. For nucleotide sequence files, this would be set `_nuc`.
- **filename** (*str, optional*) – Currently not used
- **numDigits** (*int, optional*) – Number of digits used for HLA data (default 4)
- **verboseFlag** (*int, optional*) – Verbose output (default is on, i.e. 1)
- **sequenceFilterMethod** (*str, optional*) – matching alleles to sequence, defaults to `strict`, can also be `greedy`

doFiltering(*matrix=None*)

Do filtering on the provided matrix.

Parameters
matrix (`StringMatrix`) – matrix to be filteredng

Returns
returns processed matrix for further downstream processing

Return type
StringMatrix

startFirstPass(*locus*)

Start the first pass of filtering.

Parameters
locus (*str*) – locus to start filtering

See also

Must be paired with a subsequent *endFirstPass()*

checkAlleleName(*alleleName*)

Checks allele name against the database.

Parameters
alleleName (*str*) – allele name

Returns
returns the original `allele` truncated to appropriate number of digits, if it can't be found using any of the heuristics, return it as an `untypedAllele` (normally `****`).

Return type
str

addAllele(*alleleName*)

Add allele to be filtered.

Parameters
alleleName (*str*) – process allele to be filtered

endFirstPass()

End first pass of filtering.

See also

Must be paired with a previous *startFirstPass()*

startFiltering()

Start the main filtering.

See also

must be paired with a subsequent *endFiltering()*

filterAllele(*alleleName*)

Filter a specified allele.

Parameters
alleleName (*str*) – allele to filter

Returns
return the translated allele

Return type
dict

endFiltering()

End filtering.

See also

Must be paired with a previous *startFiltering()*

writeToLog(*logstring='\n'*)

Write a string to log.

Parameters
logstring (*str*) – defaults to line feed

cleanup()

Do any cleanups.

makeSeqDictionaries(*matrix=None*, *locus=None*)

Make a sequence dictionary for a given locus.

Parameters

- **matrix** (StringMatrix) – matrix to use.
- **locus** (*str*) – locus to use.

Returns

polyseq (dict): Keyed on `locus*allele` of all allele sequences, containing **ONLY** the polymorphic positions.

polyseqpos (dict): Keyed on `locus` of the positions of the polymorphic residues which you find in `polyseq`.

Return type
tuple

Raises
RuntimeError[16]– If the alignment length could not be found in the MSF header.

translateMatrix(*matrix=None*)

Translate the whole matrix (all loci).

Parameters
matrix (StringMatrix) – matrix to translate

Returns
new instance with sequence data in columns

Return type
StringMatrix

class **BinningFilter**(*customBinningDict=None*, *logFile=None*, *untypedAllele='****'*, *filename=None*, *binningDigits=4*)

Filters original data into "bins".

This can be done through either digits (for HLA alleles) or custom rules defined a file for each locus.

Parameters

- **customBinningDict** (*dict, optional*) – a custom binning dict, this is keyed by locus, but each key consists of a series of lines, each line containing ruleset of which alleles belong in a given bin
- **logFile** (*str, optional*) – output logfilek, must be set
- **untypedAllele** (*str, optional*) – defaults to `****`
- **filename** (*str, optional*) – filename (**unused**), defaults to `None`

- **binningDigits** (*int, optional*) – defaults to 4

doDigitBinning(*matrix=None*)

Do the **digit** binning on specified matrix.

> **Note**
>
> Digit binning is done only if `binningDigits` is set.

Parameters
matrix (`StringMatrix`) – matrix to modify

Returns
the modified matrix

Return type
StringMatrix

doCustomBinning(*matrix=None*)

Do the **custom** binning on specified matrix.

> **Note**
>
> Custom binning is done only if `customBinningDict` is set.

Parameters
matrix (`StringMatrix`) – matrix to modify

Returns
the modified matrix

Return type
StringMatrix

lookupCustomBinning(*testAllele*, *locus*)

Apply custom binning rules to a allele and locus pair.

Parameters

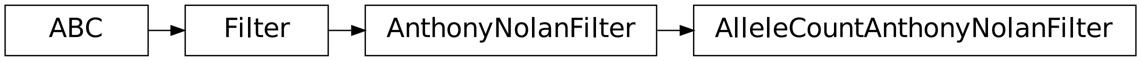
- **testAllele** (*str*) – allele to check
- **locus** (*str*) – locus to check

Returns
binned (or not) allele

Return type
str

class AlleleCountAnthonyNolanFilter(*lumpThreshold=None*, ***kw*)

Bases: *AnthonyNolanFilter*



Filters data with an allelecount less than a threshold.

Parameters
lumpThreshold (*int*) – set threshold

Base class parameters.

Parameters

- **directoryName** (*str*) – directory that AnthonyNolan allele data is located
- **remoteMSF** (*str*) – Specifies the version (tag) of the remote `msf` directory in the IMGT-HLA GitHub repo[17]. If present, the remote MSF files for the specified version will be downloaded on-demand, and cached for later reuse
- **alleleFileFormat** (*str, optional*) – file format, can be `txt` or `msf` (default). Use of `msf` files is required in order to translate allele codes into polymorphic sequence data.

- **preserveAmbiguousFlag** (*int, optional*) – If set to `0` (default) then ambiguitity is removed (e.g. `010101/0102/010301` will truncate this to `0101`). To preserve the ambiguity, set the option to `1` (for this example, it will result in a filtered allele "name" of `0101/0102/0103`)
- **preserveUnknownFlag** (*int, optional*) – If set to `0` (default) replace unknown alleles with the `untypedAllele` designator. To keep unrecognized allele names set to 1.
- **preserveLowresFlag** (*int, optional*) – This option is similar to `preserveUnknownFlag`, but only applies to lowres alleles. If set to 1, PyPop will keep allele names that are shorter than the default allele name length, usually 4 digits long. But if `preserveUnknownFlag` is set, this option has no effect, because all unknown alleles are preserved.
- **alleleDesignator** (*str, optional*) – the designator used to indicate a locus name (default `*`),
- **logFile** (*str, optional*) – log file
- **untypedAllele** (*str, optional*) – defaults to `****`
- **unsequencedSite** (*str, optional*) – defaults to `#`
- **sequenceFileSuffix** (*str, optional*) – Suffix for file names used for finding sequences each allele. (e.g.,, if the file for locus `A` is `A_prot.msf`, then keep the default be `_prot`. For nucleotide sequence files, this would be set `_nuc`.
- **filename** (*str, optional*) – Currently not used
- **numDigits** (*int, optional*) – Number of digits used for HLA data (default `4`)
- **verboseFlag** (*int, optional*) – Verbose output (default is on, i.e. `1`)
- **sequenceFilterMethod** (*str, optional*) – matching alleles to sequence, defaults to `strict`, can also be `greedy`

endFirstPass()

End first pass and then lump alleles.

First process regular *AnthonyNolanFilter* then modify all alleles with a `count` < `lumpThreshold` to `lump`.

PyPop.haplo

Module for estimating haplotypes and linkage disequilibrium measures.

Currently there are two implementations: *Emhaplofreq* and *Haplostats*.

Classes

Haplo	Estimating haplotypes given genotype data.
Emhaplofreq	Haplotype and linkage disequilibrium (LD) estimation via emhaplofreq.
Haplostats	Haplotype and LD estimation implemented via `haplo.stats`.
HaploArlequin	Performs haplotype estimation via Arlequin.

Module Contents

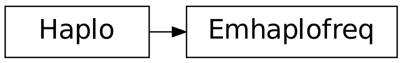
class **Haplo**

Estimating haplotypes given genotype data.

This is abstract stub class (currently has no methods).

class **Emhaplofreq**(*locusData*, *untypedAllele='****'*, *stream=None*, *testMode=False*)

Bases: *Haplo*



Haplotype and linkage disequilibrium (LD) estimation via emhaplofreq.

This is essentially a wrapper to a Python extension built on top of the `emhaplofreq` command-line program. Will refuse to estimate haplotypes longer than that defined by `emhaplofreq`.

Parameters

- **locusData** (StringMatrix) – a StringMatrix
- **untypedAllele** (*str*) – defaults to `****`
- **stream** (TextOutputStream) – output file
- **testMode** (*bool*) – default is `False`

`serializeStart()`

Serialize start of XML output to the currently defined XML stream.

See also

must be paired with a subsequent *`Emhaplofreq.serializeEnd()`*

`serializeEnd()`

Serialize end of XML output to the currently defined XML stream.

See also

must be paired with a previous *`Emhaplofreq.serializeStart()`*

`estHaplotypes`(*locusKeys=None*, *numInitCond=None*)

Estimate haplotypes for listed loci in `locusKeys`.

Parameters

- **`locusKeys`** (*`str`*) – format is a string consisting of
  - comma (`,`) separated haplotypes blocks for which to estimate haplotypes
  - within each "block", each locus is separated by colons (`:`)
- **`numInitCond`** (*`int`*) – number of initial conditions to use

Example

`*DQA1:*DPB1,*DRB1:*DQB1`, means to estimate haplotypes for `DQA1` and `DPB1` loci followed by estimation of haplotypes for `DRB1` and `DQB1` loci.

`estLinkageDisequilibrium`(*locusKeys=None*, *permutationPrintFlag=0*, *numInitCond=None*, *numPermutations=None*, *numPermuInitCond=None*)

Estimate linkage disequilibrium (LD) for listed loci.

Parameters

- **`locusKeys`** (*`str`*) – see *`estHaplotypes()`*
- **`permutationPrintFlag`** (*`int`*) – print all permutations (default `0`)
- **`numInitCond`** (*`int`*) – number of initial conditions (default `None`)
- **`numPermutations`** (*`int`*) – number of permutations (default `None`)
- **`numPermuInitCond`** (*`int`*) – number of initial conditions for each permutation (default `None`)

Example

See *`estHaplotypes()`* for an example that estimates LD

`allPairwise`(*permutationPrintFlag=0*, *numInitCond=None*, *numPermutations=None*, *numPermuInitCond=None*, *haploSuppressFlag=None*, *haplosToShow=None*, *mode=None*)

Estimate pairwise statistics for a given set of loci.

Depending on the flags passed, this can be used to estimate both LD (linkage disequilibrium) and HF (haplotype frequencies), an optional permutation test on LD can be run.

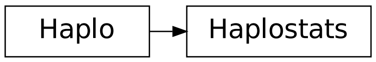
Parameters

- **`permutationPrintFlag`** (*`int`*) – sets whether the result from permutation output run will be included in the output XML. Default: `0` (disabled).
- **`numInitCond`** (*`int`*) – sets number of initial conditions before performing the permutation test. Default: `None`.
- **`numPermutations`** (*`int`*) – sets number of permutations that will be performed. Default: `None`.
- **`numPermuInitCond`** (*`int`*) – sets number of initial conditions tried per-permutation. Default: `None`.
- **`haploSuppressFlag`** (*`int`*) – sets whether haplotype information is generated in the output. Default: `None`
- **`haplosToShow`** (*`list`*) – list of haplotypes to show in output

- **`mode`** (*`str`*) – mode for haplotype output

class `Haplostats`(*locusData*, *untypedAllele='****'*, *stream=None*, *testMode=False*)

Bases: *`Haplo`*



Haplotype and LD estimation implemented via `haplo.stats`.

This is a wrapper to a portion of the `haplo.stats` R package.

Parameters

- **`locusData`** (`StringMatrix`) – a StringMatrix
- **`untypedAllele`** (*`str`*) – defaults to `****`
- **`stream`** (`TextOutputStream`) – output file
- **`testMode`** (*`bool`*) – default is `False`

`serializeStart`()

Serialize start of XML output to currently defined XML stream.

See also

must be paired with a subsequent *`Haplostats.serializeEnd()`*

`serializeEnd`()

Serialize end of XML output to currently defined XML stream.

See also

must be paired with a previous *`Haplostats.serializeStart()`*

`estHaplotypes`(*locusKeys=None*, *weight=None*, *control=None*, *numInitCond=10*, *testMode=False*)

Estimate haplotypes for listed loci in `locusKeys`.

If `locusKeys` is `None`, assume entire matrix. LD is also estimated if there are `locusKeys` consisting of only two loci.

Warning

FIXME: this does *not* yet remove missing data before haplotype estimations

Parameters

- **`locusKeys`** (*`str`*) – see *`Emhaplofreq.estHaplotypes()`* for format
- **`weight`** (*`list`*) – set weights (default `None`, which sets all weights equal)
- **`control`** (*`dict`*) – a dictionary of control parameters
- **`numInitCond`** (*`int`*) – number of initial conditions (default `None`)
- **`testMode`** (*`bool`*) – run in test mode default is `False`

Returns

multiple statistics

Return type

tuple

`allPairwise`(*weight=None*, *control=None*, *numInitCond=10*)

Estimate pairwise statistics for all pairs of loci.

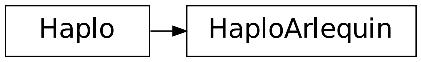
Parameters

- **`weight`** (*`list`*) – see *`Haplostats.estHaplotypes()`*

- **control** (*dict*) – see *Haplostats.estHaplotypes()*
- **numInitCond** (*int*) – see *Haplostats.estHaplotypes()*

class HaploArlequin(*arpFilename*, *idCol*, *prefixCols*, *suffixCols*, *windowSize*, *mapOrder=None*, *untypedAllele='0'*, *arlequinPrefix='arl_run'*)

Bases: *Haplo*



Performs haplotype estimation via Arlequin.

Deprecated since version 1.0.0

Deprecated since version 1.0.0.

Outputs Arlequin format data files and runtime info, also runs and parses the resulting Arlequin data so it can be made available programmatically to rest of Python framework.

Delegates all calls Arlequin to an internally instantiated ArlequinBatch Python object called 'batch'.

Parameters

- **arpFilename** (*str*) – Arlequin filename (must have `.arp` file extension)
- **idCol** (*str*) – column in input file that contains the individual `id`.
- **prefixCols** (*int*) – number of columns to ignore before allele data starts
- **suffixCols** (*int*) – number of columns to ignore after allele data stops
- **windowSize** (*int*) – size of sliding window
- **mapOrder** (*list*) – list order of columns if different to column order in file (defaults to order in file)
- **untypedAllele** (*str*) – (defaults to `0`)
- **arlequinPrefix** (*str*) – prefix for all Arlequin run-time files (defaults to `arl_run`).

outputArlequin(*data*)

Outputs the specified `.arp` sample file.

Parameters
data (*list*) – list of strings containing the `.arp` sample file

runArlequin()

Run the Arlequin haplotyping program.

Generates the expected `.txt` set-up files for Arlequin, then forks a copy of `arlecore.exe`, which must be on `PATH` to actually generate the haplotype estimates from the generated `.arp` file.

genHaplotypes()

Parses Arlequin output to retrieve estimated haplotypes.

Returns

a list of the sliding `windows` which consists of tuples. Each tuple consists of:

- freqs (dict): dictionary entry (the haplotype-frequency) key-value pairs.
- popName (str): population name (original `.arp` file prefix)
- sampleCount (int): sample count (number of samples for that window)
- lociList (list): ordered list of loci considered

Return type
list

PyPop.hardyweinberg

Computing Hardy-Weinberg statistics on genotype data.

Attributes

use_scipy	If `True` use `scipy` to compute pvalue, rather than internal `pval`

Classes

HardyWeinberg	Calculate Hardy-Weinberg statistics for a single locus.
HardyWeinbergGuoThompson	Use Guo & Thompson (1992) algorithm for calculating statistics.
HardyWeinbergEnumeration	HW testing with Maldonado Torres' exact enumeration test.
HardyWeinbergGuoThompsonArlequin	Arlequin implementation of the Guo & Thompson algorithm.

Functions

pval(chisq, dof)	Calculate p-value.

Module Contents

use_scipy = False

If `True` use `scipy` to compute pvalue, rather than internal `pval`

class HardyWeinberg(*locusData=None*, *alleleCount=None*, *lumpBelow=5*, *flagChenTest=0*)

Calculate Hardy-Weinberg statistics for a single locus.

Given the observed genotypes for a locus, calculate the expected genotype counts based on Hardy Weinberg proportions for individual genotype values, and test for fit.

Parameters

- **locusData** (*list*) – list of tuples of genotype `(allele1, allele2)`
- **alleleCount** (*tuple*) – a tuple consisting of a dictionary of counts, total count and number of untyped individuals as returned by `PyPop.DataTypes.Genotypes.getLocusDataAt()`
- **lumpBelow** (*int, optional*) – lump alleles with frequency less than this threshold as if they were in same class (Default: 5)
- **flagChenTest** (*int, optional*) – if enabled (1) do Chen's chi-square-based "corrected" p-value (Default: `0`, disabled)

serializeTo(*stream*, *allelelump=0*)

Serialize output to specified XML stream.

Parameters

- **stream** (*XMLOutputStream*) – write to specified XML stream (generally a file)
- **allelelump** (*int*) – record the allele lumping value

serializeXMLTableTo(*stream*)

Serialize the genotype table.

Parameters

stream (*XMLOutputStream*) – XML stream

class HardyWeinbergGuoThompson(*locusData=None*, *alleleCount=None*, *runMCMCTest=0*, *runPlainMCTest=0*, *dememorizationSteps=2000*, *samplingNum=1000*, *samplingSize=1000*, *maxMatrixSize=250*, *monteCarloSteps=1000000*, *testing=False*, ***kw*)

Bases: *HardyWeinberg*

HardyWeinberg
HardyWeinbergGuoThompson

Use Guo & Thompson (1992) algorithm for calculating statistics.

This Python class wraps the functionality of the Guo & Thompson program `gthwe`. In addition to the arguments for the base class, this class accepts the following additional keywords:

Parameters

- **locusData** (*list*) – list of tuples of genotype `(allele1, allele2)`
- **alleleCount** (*tuple*) – a tuple consisting of a dictionary of counts, total count and number of untyped individuals as returned by `PyPop.DataTypes.Genotypes.getLocusDataAt()`

- **runMCMCTest** (*int*) – If enabled (1) run the Monte Carlo-Markov chain (MCMC) version of the test (what is normally referred to as “Guo & Thompson”), default disabled (0)
- **runPlainMCTest** (*int*) – If enabled (1) run a plain Monte Carlo/randomization without the Markov-chain version of the test (this is also described in the original Guo & Thompson *Biometrics* paper, but was not in their original program)
- **dememorizationSteps** (*int*) – number of “dememorization” initial steps for random number generator (default 2000).
- **samplingNum** (*int*) – the number of chunks for random number generator (default 1000).
- **samplingSize** (*int*) – size of each chunk (default 1000).
- **maxMatrixSize** (*int*) – maximum size of *flattened’ lower-triangular matrix of observed alleles (default ``250`*).
- **monteCarloSteps** (*int*) – number of steps for the plain Monte Carlo randomization test (without Markov-chain)
- **testing** (*bool*) – testing mode, default False

generateFlattenedMatrix()

Generated a flattened version of the genotype matrix.

dumpTable(*locusName*, *stream*, *allelelump=0*)

Output table to stream.

Parameters

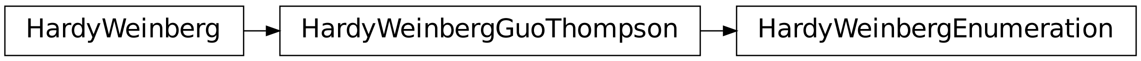
- **locusName** (*str*) – locus to output table
- **stream** (XMLOutputStream) – name of XML stream
- **allelelump** (*int*) – record allele lumping level (default 0)

Returns
if an empty tag

Return type
None

class **HardyWeinbergEnumeration**(*locusData=None*, *alleleCount=None*, *doOverall=0*, ***kw*)

Bases: *HardyWeinbergGuoThompson*



HW testing with Maldonado Torres’ exact enumeration test.

Warning

This requires the Enumeration C code to be compiled as a module using SWIG. By default this is currently disabled.

Parameters

- **locusData** (*list*) – list of tuples of genotype (allele1, allele2)
- **alleleCount** (*tuple*) – a tuple consisting of a dictionary of counts, total count and number of untyped individuals as returned by PyPop.DataTypes.Genotypes.getLocusDataAt()
- **doOverall** (*int*) – if set to true (1), then do overall *p*-value test default is false (0)

serializeTo(*stream*, *allelelump=0*)

Serialize enumeration test output to stream.

Parameters

- **stream** (XMLOutputStream) – XML stream to use
- **allelelump** (*int*) – record allele lumping level (default 0)

class **HardyWeinbergGuoThompsonArlequin**(*matrix=None*, *locusName=None*, *arlequinExec='arlecore.exe'*, *markovChainStepsHW=100000*, *markovChainDememorisationStepsHW=1000*, *untypedAllele='****'*)

Arlequin implementation of the Guo & Thompson algorithm.

Deprecated since version 1.0.0

Deprecated since version 1.0.0.

This class extracts the Hardy-Weinberg (HW) statistics using the Arlequin implementation of the HW exact test, by the following:

1. creates a subdirectory `arlequinRuns` in which all the Arlequin specific files are generated;
2. then the specified arlequin executable is run, generating the Arlequin output HTML files (`*.htm`);
3. the Arlequin output is then parsed for the relevant statistics;
4. lastly, the `arlequinRuns` directory is removed.

Since the directory name `arlequinRuns` is currently hardcoded, this has the consequence that this class cannot be invoked concurrently.

Parameters

- **`matrix`** (`StringMatrix`) – matrix to extract locus from
- **`locusName`** (*`str`*) – locus to use
- **`arlequinExec`** (*`str`*) – name of Arlequin executable
- **`markovChainStepsHW`** (*`int`*) – number of steps to use in Markov chain (default: `100000`).
- **`markovChainDememorisationStepsHW`** (*`int`*) – "Burn-in" time for Markov chain (default: `1000`).
- **`untypedAllele`** (*`str`*) – untyped allele identifier

`serializeTo`(*stream*)

Serialize output to stream.

Parameters
`stream` (`XMLOutputStream`) – stream to serialize to

`pval`(*chisq*, *dof*)

Calculate p-value.

Parameters

- **`chisq`** (*`float`*) – Chi-square value
- **`dof`** (*`int`*) – degrees of freedom

Returns
p-value

Return type
float

PyPop.homozygosity

Computing homozygosity statistics on genotype or allele counts.

Classes

`Homozygosity`	Calculate homozygosity statistics.
`HomozygosityEWSlatkinExact`	Compute homozygosity using the Ewens-Watterson-Slatkin "exact test".
`HomozygosityEWSlatkinExactPairwise`	Compute pairwise homozygosity using the Ewens-Watterson-Slatkin.

Functions

`getObservedHomozygosityFromAlleleData`(alleleData)	Get homozygosity from allele data.

Module Contents

`class` **`Homozygosity`**(*alleleData*, *rootPath='.'*)

Calculate homozygosity statistics.

Given allele count data for a given locus, calculates the observed homozygosity and returns the approximate expected homozygosity statistics taken from previous simulation runs.

Parameters

- **alleleData** (*list*) – list of allele counts
- **rootPath** (*str*) – path to the root of the directory where the pre-calculated expected homozygosity statistics can be found.

getObservedHomozygosity()

Calculate and return observed homozygosity.

Available even if expected stats cannot be calculated.

Returns
observed homozygosity

Return type
float

canGenerateExpectedStats()

Can expected homozygosity stats be calculated?

Returns 1 if expected homozygosity statistics can be calculated. Should be called before attempting to get any expected homozygosity statistics.

Returns
1 if can be calculated, otherwise `0`

Return type
int

getPValueRange()

Gets lower and upper bounds for p-value.

Only meaningful if *canGenerateExpectedStats()* returns true.

Returns
(`lower`, `upper`) bounds.

Return type
tuple

getCount()

Number of runs used to calculate statistics.

Only meaningful if *canGenerateExpectedStats()* returns 1.

Returns
number of runs

Return type
int

getExpectedHomozygosity()

Gets mean of expected homozygosity.

This is the estimate of the *expected* homozygosity. Only meaningful if *canGenerateExpectedStats()* returns true.

Returns
mean of expected homozygosity

Return type
float

getVarExpectedHomozygosity()

Gets variance of expected homozygosity.

This is the estimate of the variance *expected* homozygosity. Only meaningful if *canGenerateExpectedStats()* returns true.

Returns
variance of expected homozygosity

Return type
float

getNormDevHomozygosity()

Gets normalized deviate of homozygosity.

Only meaningful if *canGenerateExpectedStats()* returns true.

Returns
normalized deviate of homozygosity

Return type
float

`serializeHomozygosityTo`(*stream*)

Serialize homozygosity to a stream.

Parameters
`stream` (`XMLOutputStream`) – stream to save to

class `HomozygosityEWSlatkinExact`(*alleleData=None*, *numReplicates=10000*)

Bases: *`Homozygosity`*

Compute homozygosity using the Ewens-Watterson-Slatkin “exact test”.

Parameters

- `alleleData` (*`list`*) – list of allele counts
- `numReplicates` (*`int`*) – number or replicates for simulation.

`doCalcs`(*alleleData*)

Run the computations.

Parameters
`alleleData` (*`list`*) – list of allele counts

`getHomozygosity`()

Get the homozygosity statistics.

Returns

tuple consisting of:

- theta
- prob_ewens
- prob_homozygosity
- mean_homozygosity
- obsv_homozygosity
- var_homozygosity

Return type
tuple

`serializeHomozygosityTo`(*stream*)

Serialize homozygosity to a stream.

Parameters
`stream` (`XMLOutputStream`) – stream to save to

`returnBulkHomozygosityStats`(*alleleCountDict=None*, *binningMethod=None*)

Get bulk homozygosity statistics for multiple allele counts.

This function is designed to work with the `PyPop.RandomBinning` submodule.

Parameters

- `alleleCountDict` (*`dict`*) – dictionary of lists of allele counts
- `binningMethod` (*`str`*) – record the binning method used

Returns
dictionary of statistics

Return type
dict

class HomozygosityEWSlatkinExactPairwise(*matrix=None*, *numReplicates=10000*, *untypedAllele='****'*)

Compute pairwise homozygosity using the Ewens-Watterson-Slatkin.

Parameters

- **matrix** (`StringMatrix`) – matrix with multiple loci columns for pairwise comparison
- **numReplicates** (*int, optional*) – number or replicates for simulation.
- **untypedAllele** (*str, optional*) – untyped allele

serializeTo(*stream*)

Serialize to a stream.

Parameters
stream (`XMLOutputStream`) – stream to save to

getObservedHomozygosityFromAlleleData(*alleleData*)

Get homozygosity from allele data.

Parameters
alleleData (*list*) – list of allele counts

Returns
observed homozygosity

Return type
float

PyPop.parsers

Parsing input population data files.

Includes *ParseGenotypeFile* for parsing individuals genotyped at multiple loci and *ParseAlleleCountFile* for parsing literature data which only includes allele counts.

Both file formats are assumed to have a population header information with, consisting of a line of column headers (population metadata) followed by a line with the actual data, followed by the column headers for the samples (sample metadata) followed by the sample data itself (either individuals in the genotyped case, or alleles in the allele count case).

Classes

ParseFile	Common functionality for reading the two file formats.
ParseGenotypeFile	Class to parse standard datafile in genotype form.
ParseAlleleCountFile	Class to parse datafile in allele count form.

Module Contents

class ParseFile(*filename*, *validPopFields=None*, *validSampleFields=None*, *separator='\t'*, *fieldPairDesignator='_1:_2'*, *alleleDesignator='*'*, *popNameDesignator='+'*)

Common functionality for reading the two file formats.

Base class.

Parameters

- **filename** (*str*) – filename for the file to be parsed.
- **validPopFields** (*str*) – valid headers (one per line) for overall population data (no default)
- **validSampleFields** (*str*) – valid headers (one per line) for lines of sample data. (no default)
- **separator** (*str, optional*) – separator for adjacent fields (default: a tab stop, '\t').
- **fieldPairDesignator** (*str, optional*) – consists of additions to the allele *stem' for fields grouped in pairs (allele fields) [e.g. for ``HLA-A'*, and `HLA-A(2)`, then we use `:(2)`, for `DQA1_1` and `DQA1_2`, then use `_1:_2`, the latter case distinguishes both fields from the stem] (default: `:(2)`)
- **alleleDesignator** (*str, optional*) – first character of the key which determines whether this column contains allele data. Defaults to *
- **popNameDesignator** (*str, optional*) – first character of the key which determines whether this column contains the population name. Defaults to +

`getPopData()`

Returns a dictionary of population data.

Returns
keyed by types specified in population metadata file

Return type
dict

`getSampleMap()`

Returns dictionary of sample data.

Returns

each entry contains either a 2-tuple of column
position or a single column position keyed by field originally specified in sample metadata file

Return type
dict

`getFileData()`

Returns the file data.

Returns

a 2-tuple "wrapper":

- str: raw sample lines, *without* header metadata.
- str: the field separator.

Return type
tuple

`genSampleOutput`(*fieldList*)

Prints the data specified in ordered field list.

Deprecated since version 0.7.0

Deprecated since version 0.7.0.

`serializeMetadataTo`(*stream*)

Write metadata to stream.

Parameters
`stream` (*`XMLStreamOutput`*) – output stream

`class ParseGenotypeFile`(*filename*, *untypedAllele='****'*, ***kw*)

Bases: `ParseFile`

ParseFile
ParseGenotypeFile

Class to parse standard datafile in genotype form.

Processes files that consist specifically of data with individual genotyped for one or more loci.

Parameters

- **`filename`** (*`str`*) – filename for the file to be parsed.
- **`untypedAllele`** (*`str, optional`*) – The designator for an untyped locus. Defaults to `****`.

Base class.

Parameters

- **`filename`** (*`str`*) – filename for the file to be parsed.
- **`validPopFields`** (*`str`*) – valid headers (one per line) for overall population data (no default)
- **`validSampleFields`** (*`str`*) – valid headers (one per line) for lines of sample data. (no default)
- **`separator`** (*`str, optional`*) – separator for adjacent fields (default: a tab stop, '\t').
- **`fieldPairDesignator`** (*`str, optional`*) – consists of additions to the allele *stem' for fields grouped in pairs (allele fields) [e.g. for ``HLA-A'*, and `HLA-A(2)`, then we use `:(2)`, for `DQA1_1` and `DQA1_2`, then use `_1:_2`, the latter case distinguishes both fields from the stem] (default: `:(2)`)

- **alleleDesignator** (*str, optional*) – first character of the key which determines whether this column contains allele data. Defaults to *
- **popNameDesignator** (*str, optional*) – first character of the key which determines whether this column contains the population name. Defaults to +

`genValidKey`(*field*, *fieldList*)

Check and validate key.

- 'field': string with field name.
- 'fieldList': a dictionary of valid fields.

Check to see whether 'field' is a valid key, and generate the appropriate 'key'. Returns a 2-tuple consisting of 'isValidKey' boolean and the 'key'.

Note: this is explicitly done in the subclass of the abstract 'ParseFile' class (i.e. since this subclass should have `knowledge' about the nature of fields, but the abstract class should not have)

`getMatrix`()

Returns the genotype data.

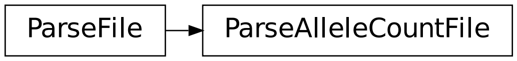
Returns the genotype data in a 'StringMatrix' NumPy array.

`serializeSubclassMetadataTo`(*stream*)

Serialize subclass-specific metadata.

class `ParseAlleleCountFile`(*filename*, ***kw*)

Bases: `ParseFile`



Class to parse datafile in allele count form.

Input files consist of allele counts across a whole population. Currently only handles one locus per population. Example:

```
<metadata-line1>
<metadata-line2>
DQA1 count
0102 20
0103 33
...
```

Base class.

Parameters

- **filename** (*str*) – filename for the file to be parsed.
- **validPopFields** (*str*) – valid headers (one per line) for overall population data (no default)
- **validSampleFields** (*str*) – valid headers (one per line) for lines of sample data. (no default)
- **separator** (*str, optional*) – separator for adjacent fields (default: a tab stop, '\t').
- **fieldPairDesignator** (*str, optional*) – consists of additions to the allele *stem' for fields grouped in pairs (allele fields) [e.g. for ``HLA-A'*, and `HLA-A(2)`, then we use `:(2)`, for `DQA1_1` and `DQA1_2`, then use `_1:_2`, the latter case distinguishes both fields from the stem] (default: `:(2)`)
- **alleleDesignator** (*str, optional*) – first character of the key which determines whether this column contains allele data. Defaults to *
- **popNameDesignator** (*str, optional*) – first character of the key which determines whether this column contains the population name. Defaults to +

`genValidKey`(*field*, *fieldList*)

Checks validity of a field.

Parameters

- **field** (*str*) – field to check
- **fieldList** (*str*) – list that `field` is checked against

Returns

2-tuple of:

- boolean: whether key is valid

- str: key

Return type
tuple

> **Note**
>
> The first element in the `fieldList` is a locus name, which may contain many loci (delimited by colons `:`). If `field` in the input file match *any* of these keys , this method will return the field and a valid match.

Example

If the first element of `fieldList` is `DQA1:DRA:DQB1`, then calling this function with `field` set to `DRA`, this would return `(True, DRA)`

`serializeSubclassMetadataTo``(`*stream*`)`

Serialize subclass specific metadata.

Parameters
`stream` (`XMLOutputStream`) – output stream

`getAlleleTable``()`

Get the current allele table.

Returns
keyed by allele name with value count

Return type
dict

`getLocusName``()`

Get the locus name.

Returns
locus name

Return type
str

`getMatrix``()`

Get the full genotype data.

Returns
containing all the genotype data

Return type
StringMatrix

PyPop.popaggregate

Module for collecting multiple population outputs.

Classes

`Meta`	Aggregates output from multiple population runs.

Functions

`translate_string_to_stdout`(xslFilename, inString[, ...])	Transform XML string using XSLT and save to stdout.
`translate_string_to_file`(xslFilename, inString, outFile)	Transform XML string using XSLT and save to file.
`translate_file_to_stdout`(xslFilename, inFile[, ...])	Transform XML file using XSLT and save to stdout.
`translate_file_to_file`(xslFilename, inFile, outFile[, ...])	Transform XML file using XSLT and save to a file.

Module Contents

class `Meta`(*popmetabinpath=None*, *datapath=None*, *metaXSLTDirectory=None*, *dump_meta=False*, *TSV_output=True*, *prefixTSV=None*, *PHYLIP_output=False*, *ihwg_output=False*, *batchsize=0*, *outputDir=None*, *xml_files=None*)

Aggregates output from multiple population runs.

Transform a specified list of `.xml` output files to `.tsv` tab-separated values (TSV) form.

Parameters

- **popmetabinpath** (*str*) – the directory for where meta sources are kept
- **datapath** (*str*) – data where XSLT and other meta sources may be kept
- **metaXSLTDirectory** (*str*) – fallback XSLT directory
- **dump_meta** (*bool*) – create the `meta.xml` file (default to `False,`)
- **TSV_output** (*bool*) – output `.tsv` tables by default (enabled by default). (such tables can be used by R)
- **prefixTSV** (*str*) – prefix to use for all `.tsv` files
- **PHYLIP_output** (*bool*) – create PHYLIP output (disabled by default)
- **ihwg_output** (*bool*) – by default, don't enable the 13th IHWG format headers
- **batchsize** (*int*) – size of batches to process separately (default `batchsize=0`, a separate batch for each file)
- **outputDir** (*str*) – output directory to write XML files to
- **xml_files** (*list*) – list of generate XML files

`translate_string_to_stdout`(*xslFilename*, *inString*, *outputDir=None*, *params=None*)

Transform XML string using XSLT and save to stdout.

Parameters

- **xslFilename** (*str*) – name of XSLT file
- **inString** (*str*) – XML string
- **outputDir** (*str, optional*) – name of output directory
- **params** (*list, optional*) – list of XSLT parameters

`translate_string_to_file`(*xslFilename*, *inString*, *outFile*, *outputDir=None*, *params=None*)

Transform XML string using XSLT and save to file.

Parameters

- **xslFilename** (*str*) – name of XSLT file
- **inString** (*str*) – XML string
- **outFile** (*str*) – name of output file
- **outputDir** (*str*) – name of output directory
- **params** (*list*) – list of XSLT parameters

`translate_file_to_stdout`(*xslFilename*, *inFile*, *inputDir=None*, *params=None*)

Transform XML file using XSLT and save to stdout.

Parameters

- **xslFilename** (*str*) – name of XSLT file
- **inFile** (*str*) – name of input XML file
- **inputDir** (*str, optional*) – name of input directory
- **params** (*list, optional*) – list of XSLT parameters

Returns

consisting of a bool (transformation successful) and str (output)

Return type

tuple

translate_file_to_file(*xslFilename*, *inFile*, *outFile*, *inputDir=None*, *outputDir=None*, *params=None*)

Transform XML file using XSLT and save to a file.

Parameters

- **xslFilename** (*str*) – name of XSLT file
- **inFile** (*str*) – name of input XML file
- **outFile** (*str*) – name of output file
- **inputDir** (*str, optional*) – name of input directory
- **outputDir** (*str, optional*) – name of output directory
- **params** (*list, optional*) – list of XSLT parameters

Returns

transformation successful

Return type

bool

PyPop.popanalysis

Primary access to PyPop's population genetics statistics modules.

This module handles processing `configparser.ConfigParser`[18]instance. The *Main* class coordinates running the analysis packages specified in this `configparser.ConfigParser`[19]instance which can be:

- created from a filename passed from command-line argument oar;
- from values populated by the GUI (for example, selected from an `.ini` file,
- created programmatically as part of an external Python program

Here is an example of calling *Main* programmatically, explicitly specifying the `untypedAllele` and `alleleDesignator` in the `.pop` file:

```
>>> from PyPop.popanalysis import Main
>>> from configparser import ConfigParser
>>>
>>> config = ConfigParser()
>>> config.read_dict({
...     "ParseGenotypeFile": {"untypedAllele": "****",
...                           "alleleDesignator": "*",
...                           "validSampleFields": "*a_1\n*a_2"}})
>>>
>>> pop_contents = '''a_1\ta_2
... ****\t****
... 01:01\t02:01
... 02:10\t03:01:02'''
>>> with open("my.pop", "w") as f:
...     _ = f.write(pop_contents)
...
>>> application = Main(
...     config=config,
...     fileName="my.pop",
...     version="fake",
... )
LOG: no XSL file, skipping text output
LOG: Data file has no header data block
```

Classes

Main	Main interface to the PyPop modules.

Functions

getConfigInstance([configFilename, altpath])	Create and return ConfigParser instance.
get_sequence_directory(directory_str)	Get the directory for the `PyPop.Filter.AnthonyNolanFilter`.

Module Contents

`class Main`(*config=None*, *xslFilename=None*, *xslFilenameDefault=None*, *fileName=None*, *datapath=None*, *thread=None*, *outputDir=None*, *version=None*, *testMode=False*)

Main interface to the PyPop modules.

Runs the analyses specified in the configuration object provided to the `config` parameter, and an input `fileName`, and generates an output XML file. The XML output file name, appends `-out.xml` on to the stem of the provided `fileName`. For example, if `fileName="MyPopulation.pop"` is provided as a parameter, the output XML file will be `MyPopulation-out.xml`.

Changed in version 1.4.0

Changed in version 1.4.0: If an `xslFilename` or `xslFilenameDefault` is provided, also generate a plain text output. Otherwise no text output is generated. Previous to this version, if neither were provided, the program would exit with an error.

Parameters

- **config** (*configparser.ConfigParser*[20]) – configure object
- **xslFilename** (*str, optional*) – XSLT file to use
- **xslFilenameDefault** (*str, optional*) – fallback file name
- **fileName** (*str*) – input .pop file
- **datapath** (*str, optional*) – root of data path
- **thread** (*str, optional*) – specified thread
- **outputDir** (*str, optional*) – use a different output directory than default
- **version** (*str, optional*) – current Python version for output
- **testMode** (*bool, optional*) – enable testing mode

`getXmlOutPath`()

Get name of XML file.

Returns
return XML file name

Return type
XMLOutputStream

`getTxtOutPath`()

Get name of `.txt` output file.

Returns
return txt file name

Return type
TextOutputStream

`getConfigInstance`(*configFilename=None*, *altpath=None*)

Create and return ConfigParser instance.

Parameters

- **configFilename** (*str*) – a specified `.ini` filename
- **altpath** (*str*) – an alternative path to search if no `.ini` filename provided in configFilename

Returns
configuration object

Return type
configparser.ConfigParser[21]

`get_sequence_directory`(*directory_str*)

Get the directory for the `PyPop.Filter.AnthonyNolanFilter`.

Parameters
directory_str (*str*) – directory to search

Returns
path to sequence files

Return type
str

PyPop.popmeta

Command-line interface for `popmeta`.

Functions

main([argv])	Entry point for `popmeta` script.

Module Contents

main(*argv=sys.argv*)

Entry point for `popmeta` script.

Parameters

argv (*list*) – list of command-line options (default is `sys.argv`)

PyPop.pypop

Command-line interface for `pypop`.

Functions

main([argv])	Entry point for `pypop` script.
main_interactive([argv])	Entry point for interactive mode script `pypop-interactive`.

Module Contents

main(*argv=sys.argv*)

Entry point for `pypop` script.

Parameters

argv (*list*) – list of command-line options (default is `sys.argv`)

main_interactive(*argv=sys.argv*)

Entry point for interactive mode script `pypop-interactive`.

Parameters

argv (*list*) – list of command-line options (default is `sys.argv`)

PyPop.randombinning

Generating randomized sets allele counts for statistical analyses.

Classes

RandomBinsForHomozygosity	Generate randomized sets of bins for homozygosity analysis.

Module Contents

class RandomBinsForHomozygosity(*logFile=None*, *untypedAllele='****'*, *filename=None*, *numReplicates=10000*, *binningReplicates=100*, *locus=None*, *xmlfile=None*, *randomResultsFileName=None*)

Generate randomized sets of bins for homozygosity analysis.

Parameters

- **logFile** (*str*) – output log file
- **untypedAllele** (*str, optional*) – untyped allele (default `****`)
- **filename** (*str*) – input filename
- **numReplicates** (*int, optional*) – replicates (default `10000`)
- **binningReplicates** (*int, optional*) – replicates for binning (default `100`)
- **locus** (*str*) – locus name

- **xmlfile** (XMLOutputStream, *optional*) – output stream
- **randomResultsFileName** (*str*) – output file for the randomized results

randomMethod(*alleleCountsBefore=None*, *alleleCountsAfter=None*)

Do binning replicates with random-based method.

Parameters

- **alleleCountsBefore** (*list*) – allele counts before binning
- **alleleCountsAfter** (*list*) – allele counts after binning

sequenceMethod(*alleleCountsBefore=None*, *alleleCountsAfter=None*, *polyseq=None*, *polyseqpos=None*)

Do binning replicates with sequence-based method.

Parameters

- **alleleCountsBefore** (*list*) – allele counts before binning
- **alleleCountsAfter** (*list*) – allele counts after binning
- **polyseq** (*dict*) – Keyed on `locus*allele` of all allele sequences, containing **ONLY** the polymorphic positions.
- **polyseqpos** (*dict*) – Keyed on `locus` of the positions of the polymorphic residues which you find in `polyseq`.

PyPop.utils

Module for common utility classes and functions.

Contains convenience classes for output of text and XML files.

Attributes

GENOTYPE_SEPARATOR	Separator between genotypes
GENOTYPE_TERMINATOR	Terminator of genotypes

Classes

TextOutputStream	Output stream for writing text files.
XMLOutputStream	Output stream for writing XML files.
StringMatrix	Matrix of strings and other metadata from input file to PyPop.
Group	Group list or sequence into non-overlapping chunks.

Functions

critical_exit(message, *args)	Log a CRITICAL message and exit with status 1.
getStreamType(stream)	Get the type of stream.
glob_with_pathlib(pattern)	Use globbing with `pathlib`.
natural_sort_key(s[, _nsre])	Generate a key for natural (human-friendly) sorting.
unique_elements(li)	Gets the unique elements in a list.
appendTo2dList(aList[, appendStr])	Append a string to each element in a list.
convertLineEndings(file, mode)	Convert line endings based on platform.
fixForPlatform(filename[, txt_ext])	Fix for some Windws/MS-DOS platforms.
copyfileCustomPlatform(src, dest[, txt_ext])	Copy file to file with fixes.
copyCustomPlatform(file, dist_dir[, txt_ext])	Copy file to directory with fixes.
checkXSLFile(xslFilename[, path, subdir, abort, msg])	Check XSL filename and return full path.
getUserFilenameInput(prompt, filename)	Get user filename input.
splitIntoNGroups(alist[, n])	Divides a list up into n parcels (plus whatever is left over).

Module Contents

GENOTYPE_SEPARATOR = '~'

Separator between genotypes

Example

In a haplotype `01:01~13:01~04:02`

`GENOTYPE_TERMINATOR = '~'`

Terminator of genotypes

Example

``02:01:01:01~`

`class` **`TextOutputStream`**(*file*)

Output stream for writing text files.

Parameters
file (*file*) – file handle

`write`(*str*)

Write to stream.

Parameters
str (*str*) – string to write

`writeln`(*str='\n'*)

Write a newline to stream.

Parameters
str (*str, optional*) – defaults to newline

`close`()

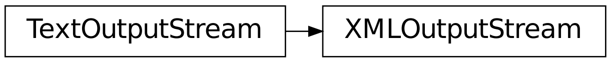
Close stream.

`flush`()

Flush to disk.

`class` **`XMLOutputStream`**(*file*)

Bases: *TextOutputStream*



Output stream for writing XML files.

`opentag`(*tagname*, ***kw*)

Write an open XML tag to stream.

Tag attributes passed as optional named keyword arguments.

Example

`opentag('tagname', role=something, id=else)`

produces the result:

`<tagname role="something" id="else">`

Attribute and values are optional:

`opentag('tagname')`

Produces:

`<tagname>`

See also

Must be be followed by a *closetag()*.

Parameters
tagname (*str*) – name of XML tag

emptytag(*tagname*, ***kw*)

Write an empty XML tag to stream.

This follows the same syntax as *opentag()* but without XML content (but can contain attributes).

Example

`` `emptytag('tagname', attr='val') ``

produces:

`<tagname attr="val"/>`

Parameters
tagname (*str*) – name of XML tag

closetag(*tagname*)

Write a closing XML tag to stream.

Example

`closetag('tagname')`

Generate a tag in the form:

`</tagname>`

See also

Must be be preceded by a *opentag()*.

Parameters
tagname (*str*) – name of XML tag

tagContents(*tagname*, *content*, ***kw*)

Write XML tags around contents to a stream.

Example

`tagContents('tagname', 'foo bar')`

produces:

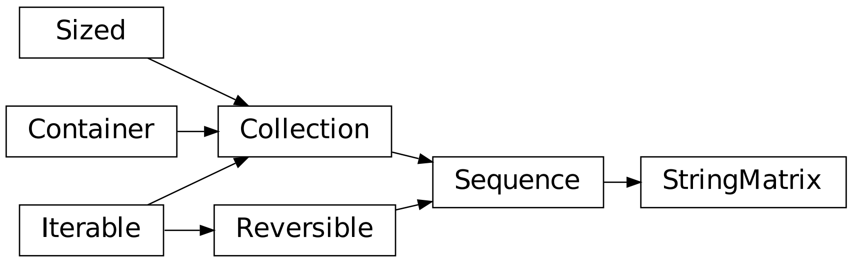
``<tagname>foo bar</tagname>` ``

Parameters

- **tagname** (*str*) – name of XML tag
- **content** (*str*) – must only be a string. &, < and > are converted into valid XML equivalents.

class StringMatrix(*rowCount=None*, *colList=None*, *extraList=None*, *colSep='\t'*, *headerLines=None*)

Bases: `collections.abc.Sequence`[22]



Matrix of strings and other metadata from input file to PyPop.

`StringMatrix` is a subclass of `collections.abc.Sequence`[23] and represents genotype or locus-based data in a row-oriented matrix structure with NumPy-style indexing and sequence semantics. Rows correspond to individuals, and columns correspond to loci.

The object supports indexing, assignment, copying, and printing using standard Python and NumPy idioms.

Parameters

- **`rowCount`** (*`int`*) – number of rows in matrix
- **`colList`** (*`list`*) – list of locus keys in a specified order
- **`extraList`** (*`list`*) – other non-matrix metadata
- **`colSep`** (*`str`*) – column separator
- **`headerLines`** (*`list`*) – list of lines in the header of original file

Note

- `len(matrix)` returns the number of rows.
- Indexing retrieves data by locus or locus combinations.
- Assignment updates genotype or metadata values in place.
- Slicing over rows (e.g., `matrix[i:j]`) is not currently supported.
- Deep copying produces a fully independent matrix.

Examples

Create a matrix of two individuals with two loci and assign genotype data:

```
>>> matrix = StringMatrix(2, ["A", "B"])
>>> matrix [0, "A"] = ("A0_1", "A0_2")
>>> matrix [1, "A"] = ("A1_1", "A1_2")
>>> matrix [0, "B"] = ("B0_1", "B0_2")
>>> matrix [1, "B"] = ("B1_1", "B1_2")
```

Length of matrix is defined as the number of individuals in the matrix:

```
>>> len(matrix)
2
```

Retrieve data for a single locus:

```
>>> matrix["A"]
[['A0_1', 'A0_2'], ['A1_1', 'A1_2']]
```

String representation:

```
>>> print (matrix)
StringMatrix([['A0_1', 'A0_2', 'B0_1', 'B0_2'],
       ['A1_1', 'A1_2', 'B1_1', 'B1_2']], dtype=object)
```

Copying the matrix:

```
>>> import copy
>>> m2 = copy.deepcopy(matrix)
>>> m2 is matrix
False
```

`__repr__`()

Override default representation.

Returns
new string representation

Return type
str

`__len__`()

Get number of rows (individuals) in the matrix.

This allows `StringMatrix` instances to be used with *len()*, iteration, and other Python sequence protocols.

Returns
number of rows in the matrix

Return type
int

`__deepcopy__`(*memo*)

Create a deepcopy for `copy.deepcopy`.

This simply calls `self.copy()` to allow `copy.deepcopy(matrixInstance)` to work out of the box.

Parameters
`memo` (*`dict`*) – opaque object

Returns
copy of the matrix

Return type
StringMatrix

`__getslice__`(*i*, *j*)

Get slice (overrides built-in).

> **Warning**
>
> Currently not supported for *`StringMatrix`*

`__getitem__`(*key*)

Get the item at given key (overrides built-in numpy).

Parameters
`key` (*`str`*) – locus key

Returns
a list (a single column vector if only one position specified), or list of lists: (a set of column vectors if several positions specified) of tuples for `key`

Return type
list

Raises
`KeyError`[24]– if key is not found, or of wrong type

`__setitem__`(*index*, *value*)

Set the value at an index (override built in).

Parameters

- **`index`** (*`tuple`*) – index into matrix
- **`value`** (*`tuple` | `str`*) – can set using a tuple of strings, or a single string (for metadata)

Raises

- **`IndexError`**[25]– if `index` is not a tuple
- **`ValueError`**[26]– if `value` is not a tuple or string
- **`KeyError`**[27]– if the `index` can't be found

`dump`(*locus=None*, *stream=sys.stdout*)

Write file to a stream in original format.

Parameters

- **`locus`** (*`str, optional`*) – write just specified locus, if omitted, default to all loci
- **`stream`** (`TextOutputStream` | `XMLOutputStream` | *`stdout`*) – output stream

`copy`()

Make a (deep) copy.

Returns
a deep copy of the current object

Return type
StringMatrix

`getNewStringMatrix`(*key*)

Create new StringMatrix containing specified loci.

Note

The format of the keys is identical to *__getitem__()* except that it returns a full `StringMatrix` instance which includes all metadata

Parameters
: **key** (*str*) – a string representing the loci, using the `locus1:locus2` format

Returns
: full instance

Return type
: *StringMatrix*

Raises
: `KeyError`[28]– if locus can not be found.

`getUniqueAlleles`(*key*)

Get naturally sorted list of unique alleles.

Parameters
: **key** (*str*) – loci to get

Returns
: list of unique integers sorted by allele name using natural sort

Return type
: list

`convertToInts`()

Convert the matrix to integers.

Note

This function is used by the *PyPop.haplo.Haplostats* class. Note that integers start at 1 for compatibility with haplo-stats module

Returns
: matrix where the original allele names are now represented by integers

Return type
: *StringMatrix*

`countPairs`()

Count all possible pairs of haplotypes for each matrix row.

Warning

This does *not* do any involved handling of missing data as per `geno.count.pairs` from R `haplo.stats` module.

Returns
: each element is the number of pairs in row order

Return type
: list

`flattenCols`()

Flatten columns into a single list.

Important

Currently assumes entries are integers.

Returns
: all alleles, the two genotype columns concatenated for each locus

Return type
list

filterOut(*key*, *blankDesignator*)

Get matrix rows filtered by a designator.

Parameters

- **key** (*str*) – locus to filter
- **blankDesignator** (*str*) – string to exclude

Returns
the rows of the matrix that *do not* contain `blankDesignator` at any rows

Return type
list

getSuperType(*key*)

Get a matrix grouped by specified key.

Example

Return a new matrix with the column vector with the alleles for each genotype concatenated like so:

```
>>> matrix = StringMatrix(2, ["A", "B"])
>>> matrix[0, "A"] = ("A01", "A02")
>>> matrix[1, "A"] = ("A11", "A12")
>>> matrix[0, "B"] = ("B01", "B02")
>>> matrix[1, "B"] = ("B11", "B12")
>>> print(matrix)
StringMatrix([['A01', 'A02', 'B01', 'B02'],
       ['A11', 'A12', 'B11', 'B12']], dtype=object)
>>> matrix.getSuperType("A:B")
StringMatrix([['A01:B01', 'A02:B02'],
       ['A11:B11', 'A12:B12']], dtype=object)
```

Parameters
key (*str*) – loci to group

Returns
a new matrix with the columns concatenated

Return type
StringMatrix

class Group(*li*, *size*)

Group list or sequence into non-overlapping chunks.

Example

```
>>> for pair in Group('aabbccddee', 2):
...    print(pair)
...
aa
bb
cc
dd
ee
```

```
>>> a = Group('aabbccddee', 2)
>>> a[0]
'aa'
>>> a[3]
'dd'
```

Parameters

- **li** (*str*|*list*) – string or list
- **size** (*int*) – size of grouping

`__getitem__`(*group*)

Get the item by position.

Parameters
`group` (*`int`*) – get the item by position

Returns
the value at that position

Return type
str|list

Raises
`IndexError`[29] – if `group` is out of bounds

`critical_exit`(*message*, **args*)

Log a CRITICAL message and exit with status 1.

Added in version 1.4.0

Added in version 1.4.0.

Parameters
`message` (*`str`*) – Logging format string.

`getStreamType`(*stream*)

Get the type of stream.

Parameters
`stream` (`TextOutputStream`|`XMLOutputStream`) – stream to check

Returns
either `xml` or `text`.

Return type
string

`glob_with_pathlib`(*pattern*)

Use globbing with `pathlib`.

Parameters
`pattern` (*`str`*) – globbing pattern

Returns
of pathlib globs

Return type
list

`natural_sort_key`(*s*, *_nsre=re.compile('([0-9]+)')*)

Generate a key for natural (human-friendly) sorting.

This function splits a string into text and number components so that numbers are compared by value instead of lexicographically. It is intended for use as the `key` function in `list.sort()` or `sorted()`.

Example

```
>>> items = ["item2", "item10", "item1"]
>>> sorted(items, key=natural_sort_key)
['item1', 'item2', 'item10']
```

Parameters

- **`s`** (*`str`*) – The string to split into text and number components.
- **`_nsre`** (*`Pattern`*) – Precompiled regular expression used internally to split the string into digit and non-digit chunks. This is not intended to be overridden in normal use.

Returns
A list of strings and integers to be used as a sort key.

Return type
list

`unique_elements`(*li*)

Gets the unique elements in a list.

Parameters
`li` (*list*) – a list

Returns
unique elements

Return type
list

`appendTo2dList`(*aList*, *appendStr=':'*)

Append a string to each element in a list.

Parameters

- `aList` (*list*) – list to append to
- `appendStr` (*str*) – string to append

Returns
a list with string appended to each element

Return type
list

`convertLineEndings`(*file*, *mode*)

Convert line endings based on platform.

Parameters

- `file` (*str*) – file name to convert
- `mode` (*int*) – Conversion mode, one of
  - 1 Unix to Mac
  - 2 Unix to DOS

`fixForPlatform`(*filename*, *txt_ext=0*)

Fix for some Windws/MS-DOS platforms.

Parameters

- `filename` (*str*) – path to file
- `txt_ext` (*int, optional*) – if enabled (1) add a `.txt` extension

`copyfileCustomPlatform`(*src*, *dest*, *txt_ext=0*)

Copy file to file with fixes.

Parameters

- `src` (*str*) – source file
- `dest` (*str*) – source file
- `txt_ext` (*int, optional*) – if enabled (1) add a `.txt` extension

`copyCustomPlatform`(*file*, *dist_dir*, *txt_ext=0*)

Copy file to directory with fixes.

Parameters

- `file` (*str*) – source file
- `dist_dir` (*str*) – source directory
- `txt_ext` (*int, optional*) – if enabled (1) add a `.txt` extension

`checkXSLFile`(*xslFilename*, *path=''*, *subdir=''*, *abort=False*, *msg=''*)

Check XSL filename and return full path.

Parameters

- `xslFilename` (*str*) – name of the XSL file
- `path` (*str*) – root path to check
- `subdir` (*str*) – subdirectory under `path` to check

- **abort** (*bool*) – if enabled (`True`) file isn't found, exit with an error. Default is `False`
- **msg** (*str*) – output message on abort

Returns
checked and validaated path

Return type
str

`getUserFilenameInput`(*prompt*, *filename*)

Get user filename input.

Read user input for a filename, check its existence, continue requesting input until a valid filename is entered.

Parameters

- **prompt** (*str*) – description of file
- **filename** (*str*) – default filename

Returns
name of file eventually selected

Return type
str

`splitIntoNGroups`(*alist*, *n=1*)

Divides a list up into n parcels (plus whatever is left over).

Example

```
>>> a = ['A', 'B', 'C', 'D', 'E']
>>> splitIntoNGroups(a, 2)
[['A', 'B'], ['C', 'D'], ['E']]
```

Parameters

- **alist** (*list*) – list to divide up
- **n** (*int*) – parcel size

Returns
list of lists

Return type
list

PyPop.xslt

Python XSLT extensions for handling things outside the scope of XSLT 1.0.

Attributes

ns	Function namespace for custom PyPop XSLT extension functions.

Functions

num_zeros(decimal)	Count zeroes.
exponent_len(num)	Calculate space taken for exponent.
format_number_fixed_width(_context, *args)	Format number to fixed width.

Package Contents

ns

Function namespace for custom PyPop XSLT extension functions.

This namespace allows registering Python functions that can be called directly from XSLT stylesheets.

prefix

The namespace prefix used in XSLT stylesheets. Here it is set to `"es"`, so extension functions are invoked as `es:format_number_fixed_width(...)`. See example in *format_number_fixed_width()*

Type

str

num_zeros(*decimal*)

Count zeroes.

Parameters

decimal (*float*) – number to check

Returns

number of zeroes in floating point number, or `inf` if number is zero

Return type

int

exponent_len(*num*)

Calculate space taken for exponent.

Example

```
>>> exponent_len(1e-03)
2
>>> exponent_len(1e-10)
3
```

Parameters

num (*float*) – input number

Returns

length of exponent

Return type

int

format_number_fixed_width(*_context*, **args*)

Format number to fixed width.

Example

```
>>> ns["format_number_fixed_width"] = format_number_fixed_width
>>> root = etree.XML("<a><b>0.0000043</b></a>")
>>> doc = etree.ElementTree(root)
>>> xslt = etree.XSLT(etree.XML('''
... <stylesheet version="1.0" xmlns="http://www.w3.org/1999/XSL/Transform" xmlns:es="http://pypop.org/lxml/functions">
...  <output method="text" encoding="ASCII"/>
...  <template match="/">
...   <text>Yep [</text>
...   <value-of select="es:format_number_fixed_width(string(/a/b), 5)"/>
...   <text>]</text>
...  </template>
... </stylesheet>
... '''))
>>>
>>> print(xslt(doc))
Yep [4.3e-6]
```

Note

arguments from XSLT file: `num` and `places` are encoded in `*args`.

Parameters

_context (*obj*) – not used

Returns

formatted number to fixed width

Return type

str

4 Deprecated Submodules

PyPop.arlequin

Provides Arlequin functionality in Python.

Deprecated since version 1.0.0

Deprecated since version 1.0.0: Only works for an obsolete version of Arlequin.

Attributes

usage_message

Classes

ArlequinWrapper	Wraps the functionality of the Arlequin[30]program.
ArlequinExactHWTest	Wraps the Arlequin Hardy-Weinberg exact functionality.
ArlequinBatch	A wrapper for running Arlequin from the command-line.

Module Contents

`class` **`ArlequinWrapper`**(*matrix=None*, *arlequinPrefix='arl_run'*, *arlequinExec='arlecore.exe'*, *untypedAllele='****'*, *arpFilename='output.arp'*, *arsFilename='arl_run.ars'*)

Wraps the functionality of the Arlequin[31]program.

Parameters

- **`matrix`** (`StringMatrix`) – matrix
- **`arlequinPrefix`** (*`str, optional`*) – directory prefix (default `arl_run`)
- **`arlequinExec`** (*`str, optional`*) – executable program (default `arlecore.exe`)
- **`untypedAllele`** (*`str, optional`*) – untyped allele designator (default `****`)
- **`arpFilename`** (*`str, optional`*) – default output file name (default `output.arp`)
- **`arsFilename`** (*`str, optional`*) – default run file name (default `arl_run.ars`)

`outputArpFile`(*group*)

Output the `.arp` file.

Parameters
`group` (*`list`*) – list of loci to pass to Arlequin

`outputArsFile`(*arsFilename*, *arsContents*)

Outputs the run-time Arlequin program file.

Parameters

- **`arsFilename`** (*`str`*) – name of file
- **`arsContents`** (*`str`*) – contents of file

`outputRunFiles`()

Generates the expected '.txt' set-up files for Arlequin.

`runArlequin`()

Run the Arlequin haplotyping program.

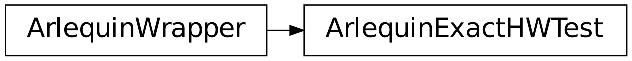
Forks a copy of `arlecore.exe`, which must be on `PATH` to actually generate the desired statistics estimates from the generated `.arp` file.

`cleanup`()

Remove the working Arlequin subdirectory.

class **ArlequinExactHWTest**(*matrix=None*, *lociList=None*, *markovChainStepsHW=100000*, *markovChainDememorisationStepsHW=1000*, ***kw*)

Bases: `ArlequinWrapper`



Wraps the Arlequin Hardy-Weinberg exact functionality.

Run Hardy-Weinberg exact test on list specified in `lociList`.

hwExactTest

standard config options for Arlequin

Type

str

Parameters

- **matrix** (`StringMatrix`) – StringMatrix for testing
- **lociList** (*list*) – list of loci
- **markovChainStepsHW** (*int, optional*) – Number of steps to use in Markov chain (default: `100000`)
- **markovChainDememorisationStepsHW** (*int, optional*) – "Burn-in" time for Markov chain (default: `1000`).

getHWExactTest()

Returns a dictionary of loci.

Returns

Each dictionary element contains a tuple of the results from the Arlequin implementation of the Hardy-Weinberg exact test, namely:

- number of genotypes,
- observed heterozygosity,
- expected heterozygosity,
- the p-value,
- the standard deviation,
- number of steps,

If locus is monomorphic, the HW exact test can't be run, and the contents of the dictionary element simply contains the string `monomorphic`, rather than the tuple of values.

Return type

dict

class **ArlequinBatch**(*arpFilename*, *arsFilename*, *idCol*, *prefixCols*, *suffixCols*, *windowSize*, *mapOrder=None*, *untypedAllele='0'*, *arlequinPrefix='arl_run'*)

A wrapper for running Arlequin from the command-line.

Given a delimited text file of multi-locus genotype data: provides methods to output Arlequin format data files and runtime info and execution of Arlequin itself. Used to provide a "batch" (i.e. command line) mode for generating appropriate Arlequin input files and for forking Arlequin itself.

Parameters

- **arpFilename** (*str*) – Arlequin filename (must have `.arp` file extension)
- **arsFilename** (*str*) – Arlequin settings filename (must have `.ars` file extension)
- **idCol** (*str*) – column in input file that contains the individual id.
- **prefixCols** (*int*) – number of columns to ignore before allele data starts
- **suffixCols** (*int*) – number of columns to ignore after allele data stops
- **windowSize** (*int*) – size of sliding window
- **mapOrder** (*list, optional*) – list order of columns if different to column order in file (defaults to order in file)
- **untypedAllele** (*str, optional*) – (defaults to `0`)
- **arlequinPrefix** (*str, optional*) – prefix for all Arlequin run-time files (defaults to `arl_run`).

outputArlequin(*data*)

Outputs the specified .arp sample file.

Parameters
data (*list*) – list of lines of data.

outputRunFiles()

Generates the expected set-up files for Arlequin.

Includes `.txt` and `.ars` file names.

runArlequin()

Run the Arlequin haplotyping program.

Forks a copy of `arlecore.exe`, which must be on `PATH` to actually generate the desired statistics estimates from the generated `.arp` file.

usage_message = Multiline-String

```
"""Usage: Arlequin.py [OPTION] INPUTFILE ARPFILE ARSFILE
Process a tab-delimited INPUTFILE of alleles to produce an data files
(including ARPFILE), using parameters from ARSFILE for the Arlequin population
genetics program.

 -i, --idcol=NUM       column number of identifier (first column is zero)
 -l, --ignorelines=NUM number of header lines to ignore in in file
 -c, --cols=POS1,POS2  number of leading columns (POS1) before start and
                        number of trailing columns before the end (POS2) of
                        allele data (including IDCOL)
 -k, --sort=POS1,..    specify order of loci if different from column order
                        in file (must not repeat a locus)
 -w, --windowsize=NUM  number of loci involved in window size
                        (note that this is half the number of allele columns)
 -u, --untyped=STR     the string that represents `untyped' alleles
                        (defaults to '****')
 -x, --execute         execute the Arlequin program
 -h, --help            this message
 -d, --debug           switch on debugging

  INPUTFILE   input text file
  ARPFILE     output Arlequin '.arp' project file
  ARSFILE     input Arlequin '.ars' settings file"""
```

5 Attributes

logger	Package-wide logger used throughout a PyPop run.
__version__	PyPop version. If installed, this is the package version, otherwise it returns repository version.
copyright_message	copyright information used in `--help` screens and elsewhere
platform_info	platform information used in `--help` screens and elsewhere

6 Exceptions

PyPopModuleRenameDeprecationWarning	Deprecation warning for PyPop module renames.

7 Functions

setup_logger([level, filename, doctest_mode])	Configure the 'pypop' logger with stdout/file handler, optional debug verbosity, and doctest mode.

8 Package Contents

exception PyPopModuleRenameDeprecationWarning

Bases: `DeprecationWarning`[32]

PyPopModuleRenameDeprecationWarning

Deprecation warning for PyPop module renames.

Added in version 1.4.0

Added in version 1.4.0.

Initialize self. See help(type(self)) for accurate signature.

`logger`

Package-wide logger used throughout a PyPop run.

Added in version 1.4.0

Added in version 1.4.0.

`__version__`

PyPop version. If installed, this is the package version, otherwise it returns repository version.

`copyright_message = Multiline-String`

```
"""Copyright (C) 2003-2006 Regents of the University of California.
Copyright (C) 2007-2025 PyPop team.
This is free software.  There is NO warranty; not even for
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE."""
```

copyright information used in `--help` screens and elsewhere

`platform_info`

platform information used in `--help` screens and elsewhere

`setup_logger`(*level=logging.INFO*, *filename=None*, *doctest_mode=True*)

Configure the 'pypop' logger with stdout/file handler, optional debug verbosity, and doctest mode.

Added in version 1.4.0

Added in version 1.4.0.

Parameters

- **level** (*str, optional*) – `INFO` (default), `DEBUG` (more detailed), `WARNING`, `CRITICAL`
- **filename** (*str, optional*) – Optional file to log to. If `None`, logs to `stdout`.
- **doctest_mode** (*bool, optional*) – If True, forcibly rebinds the logger to sys.stdout and disables propagation so doctests see output.

9 GNU Free Documentation License

Copyright (C) 2000,2001,2002 Free Software Foundation, Inc. 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A “Secondary Section” is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document’s overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The “Invariant Sections” are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The “Cover Texts” are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A “Transparent” copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not “Transparent” is called “Opaque”.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The “Title Page” means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, “Title Page” means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

A section “Entitled XYZ” means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as “Acknowledgements”, “Dedications”, “Endorsements”, or “History”.) To “Preserve the Title” of such a section when you modify the Document means that it remains a section “Entitled XYZ” according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document’s license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document’s license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled “History”, Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled “History” in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the “History” section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled “Acknowledgements” or “Dedications”, Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled “Endorsements”. Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled “Endorsements” or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties–for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4. above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright (c) YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with…Texts." line with this:

> with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

"""

Python Module Index

p

PyPop, 2
PyPop.arlequin, 41
PyPop.citation, 3
PyPop.command_line_interface, 3
PyPop.datatypes, 5
PyPop.filters, 8
PyPop.haplo, 13
PyPop.hardyweinberg, 16
PyPop.homozygosity, 19
PyPop.parsers, 22
PyPop.popaggregate, 25
PyPop.popanalysis, 27
PyPop.popmeta, 29
PyPop.pypop, 29
PyPop.randombinning, 29
PyPop.utils, 30
PyPop.xslt, 39

Notes

1. https://github.com/readthedocs/sphinx-autoapi
2. http://pypop.org/docs
3. http://pypop.org/pypop-guide-1.4.1.pdf
4. https://peps.python.org/pep-0008/#package-and-module-names
5. https://docs.python.org/3/library/exceptions.html#DeprecationWarning
6. https://docs.python.org/3/library/exceptions.html#UserWarning
7. https://docs.python.org/3/library/collections.html#collections.OrderedDict
8. https://docs.python.org/3/library/collections.html#collections.OrderedDict
9. https://docs.python.org/3/library/configparser.html#configparser.ConfigParser
10. https://docs.python.org/3/library/argparse.html#argparse.Action
11. https://docs.python.org/3/library/argparse.html#argparse.ArgumentParser
12. https://docs.python.org/3/library/argparse.html#argparse.ArgumentParser
13. https://docs.python.org/3/library/exceptions.html#Exception
14. https://docs.python.org/3/library/abc.html#abc.ABC
15. https://github.com/ANHIG/IMGTHLA/
16. https://docs.python.org/3/library/exceptions.html#RuntimeError
17. https://github.com/ANHIG/IMGTHLA/
18. https://docs.python.org/3/library/configparser.html#configparser.ConfigParser
19. https://docs.python.org/3/library/configparser.html#configparser.ConfigParser
20. https://docs.python.org/3/library/configparser.html#configparser.ConfigParser
21. https://docs.python.org/3/library/configparser.html#configparser.ConfigParser
22. https://docs.python.org/3/library/collections.abc.html#collections.abc.Sequence
23. https://docs.python.org/3/library/collections.abc.html#collections.abc.Sequence
24. https://docs.python.org/3/library/exceptions.html#KeyError
25. https://docs.python.org/3/library/exceptions.html#IndexError
26. https://docs.python.org/3/library/exceptions.html#ValueError
27. https://docs.python.org/3/library/exceptions.html#KeyError
28. https://docs.python.org/3/library/exceptions.html#KeyError
29. https://docs.python.org/3/library/exceptions.html#IndexError
30. http://lgb.unige.ch/arlequin/
31. http://lgb.unige.ch/arlequin/
32. https://docs.python.org/3/library/exceptions.html#DeprecationWarning

Index

Symbols

__call__() (*CitationAction method*), 4
__deepcopy__() (*StringMatrix method*), 33
__getitem__() (*Group method*), 36
__getitem__() (*StringMatrix method*), 34
__getslice__() (*StringMatrix method*), 34
__len__() (*StringMatrix method*), 33
__repr__() (*StringMatrix method*), 33
__setitem__() (*StringMatrix method*), 34
__str__() (*SubclassError method*), 8
__version__ (*in module PyPop*), 44

A

addAllele() (*AnthonyNolanFilter method*), 10
addAllele() (*Filter method*), 8
addAllele() (*PassThroughFilter method*), 9
AlleleCountAnthonyNolanFilter (*class in PyPop.filters*), 12
AlleleCounts (*class in PyPop.datatypes*), 6
allPairwise() (*Emhaplofreq method*), 14
allPairwise() (*Haplostats method*), 15
AnthonyNolanFilter (*class in PyPop.filters*), 9
appendTo2dList() (*in module PyPop.utils*), 38
ArlequinBatch (*class in PyPop.arlequin*), 42
ArlequinExactHWTest (*class in PyPop.arlequin*), 41
ArlequinWrapper (*class in PyPop.arlequin*), 41

B

BinningFilter (*class in PyPop.filters*), 11

C

canGenerateExpectedStats() (*Homozygosity method*), 20
checkAlleleName() (*AnthonyNolanFilter method*), 10
checkAlleleName() (*Filter method*), 8
checkAlleleName() (*PassThroughFilter method*), 9
checkIfSequenceData() (*in module PyPop.datatypes*), 7
checkXSLFile() (*in module PyPop.utils*), 38
citation_output_formats (*in module PyPop.citation*), 3
CitationAction (*class in PyPop.command_line_interface*), 4
cleanup() (*AnthonyNolanFilter method*), 11
cleanup() (*ArlequinWrapper method*), 41
cleanup() (*Filter method*), 9
cleanup() (*PassThroughFilter method*), 9
close() (*TextOutputStream method*), 31
closetag() (*XMLOutputStream method*), 32
convert_citation_formats() (*in module PyPop.citation*), 3
convertLineEndings() (*in module PyPop.utils*), 38
convertToInts() (*StringMatrix method*), 35
copy() (*StringMatrix method*), 34
copyCustomPlatform() (*in module PyPop.utils*), 38
copyfileCustomPlatform() (*in module PyPop.utils*), 38
copyright_message (*in module PyPop*), 44
countPairs() (*StringMatrix method*), 35
critical_exit() (*in module PyPop.utils*), 37

D

doCalcs() (*HomozygosityEWSlatkinExact method*), 21
doCustomBinning() (*BinningFilter method*), 12
doDigitBinning() (*BinningFilter method*), 12
doFiltering() (*AnthonyNolanFilter method*), 10
doFiltering() (*Filter method*), 8
doFiltering() (*PassThroughFilter method*), 9
dump() (*StringMatrix method*), 34
dumpTable() (*HardyWeinbergGuoThompson method*), 18

E

Emhaplofreq (*class in PyPop.haplo*), 13
emptytag() (*XMLOutputStream method*), 31
endFiltering() (*AnthonyNolanFilter method*), 11
endFiltering() (*Filter method*), 8
endFiltering() (*PassThroughFilter method*), 9
endFirstPass() (*AlleleCountAnthonyNolanFilter method*), 13
endFirstPass() (*AnthonyNolanFilter method*), 10
endFirstPass() (*Filter method*), 8
endFirstPass() (*PassThroughFilter method*), 9
estHaplotypes() (*Emhaplofreq method*), 14
estHaplotypes() (*Haplostats method*), 15
estLinkageDisequilibrium() (*Emhaplofreq method*), 14
exponent_len() (*in module PyPop.xslt*), 40

F

Filter (*class in PyPop.filters*), 8
filterAllele() (*AnthonyNolanFilter method*), 10
filterAllele() (*Filter method*), 8
filterAllele() (*PassThroughFilter method*), 9
filterOut() (*StringMatrix method*), 36
fixForPlatform() (*in module PyPop.utils*), 38
flattenCols() (*StringMatrix method*), 35
flush() (*TextOutputStream method*), 31
format_number_fixed_width() (*in module PyPop.xslt*), 40

G

generateFlattenedMatrix() (*HardyWeinbergGuoThompson method*), 18
genHaplotypes() (*HaploArlequin method*), 16
GENOTYPE_SEPARATOR (*in module PyPop.utils*), 30
GENOTYPE_TERMINATOR (*in module PyPop.utils*), 31
Genotypes (*class in PyPop.datatypes*), 5
genSampleOutput() (*ParseFile method*), 23
genValidKey() (*ParseAlleleCountFile method*), 24
genValidKey() (*ParseGenotypeFile method*), 24
get_parent_cli() (*in module PyPop.command_line_interface*), 4
get_popmeta_cli() (*in module PyPop.command_line_interface*), 4
get_pypop_cli() (*in module PyPop.command_line_interface*), 4
get_sequence_directory() (*in module PyPop.popanalysis*), 28
getAlleleCount() (*AlleleCounts method*), 7
getAlleleCount() (*Genotypes method*), 5
getAlleleCountAt() (*Genotypes method*), 5
getAlleleTable() (*ParseAlleleCountFile method*), 25
getConfigInstance() (*in module PyPop.popanalysis*), 28
getCount() (*Homozygosity method*), 20
getExpectedHomozygosity() (*Homozygosity method*), 20
getFileData() (*ParseFile method*), 23
getHomozygosity() (*HomozygosityEWSlatkinExact method*), 21
getHWExactTest() (*ArlequinExactHWTest method*), 42
getIndividualsData() (*Genotypes method*), 6
getLocusData() (*Genotypes method*), 6
getLocusDataAt() (*Genotypes method*), 6
getLocusList() (*Genotypes method*), 5
getLocusName() (*AlleleCounts method*), 7
getLocusName() (*ParseAlleleCountFile method*), 25
getLocusPairs() (*in module PyPop.datatypes*), 7
getLumpedDataLevels() (*in module PyPop.datatypes*), 7
getMatrix() (*ParseAlleleCountFile method*), 25
getMatrix() (*ParseGenotypeFile method*), 24
getMetaLocus() (*in module PyPop.datatypes*), 7
getNewStringMatrix() (*StringMatrix method*), 34
getNormDevHomozygosity() (*Homozygosity method*), 20
getObservedHomozygosity() (*Homozygosity method*), 20
getObservedHomozygosityFromAlleleData() (*in module PyPop.homozygosity*), 22

getPopData() (*ParseFile method*), 22
getPValueRange() (*Homozygosity method*), 20
getSampleMap() (*ParseFile method*), 23
getStreamType() (*in module PyPop.utils*), 37
getSuperType() (*StringMatrix method*), 36
getTxtOutPath() (*Main method*), 28
getUniqueAlleles() (*StringMatrix method*), 35
getUserFilenameInput() (*in module PyPop.utils*), 39
getVarExpectedHomozygosity() (*Homozygosity method*), 20
getXmlOutPath() (*Main method*), 28
glob_with_pathlib() (*in module PyPop.utils*), 37
Group (*class in PyPop.utils*), 36

H

Haplo (*class in PyPop.haplo*), 13
HaploArlequin (*class in PyPop.haplo*), 16
Haplostats (*class in PyPop.haplo*), 15
HardyWeinberg (*class in PyPop.hardyweinberg*), 17
HardyWeinbergEnumeration (*class in PyPop.hardyweinberg*), 18
HardyWeinbergGuoThompson (*class in PyPop.hardyweinberg*), 17
HardyWeinbergGuoThompsonArlequin (*class in PyPop.hardyweinberg*), 18
Homozygosity (*class in PyPop.homozygosity*), 19
HomozygosityEWSlatkinExact (*class in PyPop.homozygosity*), 21
HomozygosityEWSlatkinExactPairwise (*class in PyPop.homozygosity*), 21
hwExactTest (*ArlequinExactHWTest attribute*), 42

L

logger (*in module PyPop*), 44
lookupCustomBinning() (*BinningFilter method*), 12

M

Main (*class in PyPop.popanalysis*), 27
main() (*in module PyPop.popmeta*), 29
main() (*in module PyPop.pypop*), 29
main_interactive() (*in module PyPop.pypop*), 29
makeSeqDictionaries() (*AnthonyNolanFilter method*), 11
Meta (*class in PyPop.popaggregate*), 25
module
    PyPop, 2
    PyPop.arlequin, 41
    PyPop.citation, 3
    PyPop.command_line_interface, 3
    PyPop.datatypes, 5
    PyPop.filters, 8
    PyPop.haplo, 13
    PyPop.hardyweinberg, 16
    PyPop.homozygosity, 19
    PyPop.parsers, 22
    PyPop.popaggregate, 25
    PyPop.popanalysis, 27
    PyPop.popmeta, 29
    PyPop.pypop, 29
    PyPop.randombinning, 29
    PyPop.utils, 30
    PyPop.xslt, 39

N

natural_sort_key() (*in module PyPop.utils*), 37
ns (*in module PyPop.xslt*), 39
num_zeros() (*in module PyPop.xslt*), 40

O

opentag() (*XMLOutputStream method*), 31
outputArlequin() (*ArlequinBatch method*), 42
outputArlequin() (*HaploArlequin method*), 16
outputArpFile() (*ArlequinWrapper method*), 41
outputArsFile() (*ArlequinWrapper method*), 41
outputRunFiles() (*ArlequinBatch method*), 43
outputRunFiles() (*ArlequinWrapper method*), 41

P

ParseAlleleCountFile (*class in PyPop.parsers*), 24
ParseFile (*class in PyPop.parsers*), 22
ParseGenotypeFile (*class in PyPop.parsers*), 23
PassThroughFilter (*class in PyPop.filters*), 9
platform_info (*in module PyPop*), 44
prefix (*in module PyPop.xslt*), 39
pval() (*in module PyPop.hardyweinberg*), 19
PyPop
    module, 2
PyPop.arlequin
    module, 41
PyPop.citation
    module, 3
PyPop.command_line_interface
    module, 3
PyPop.datatypes
    module, 5
PyPop.filters
    module, 8
PyPop.haplo
    module, 13
PyPop.hardyweinberg
    module, 16
PyPop.homozygosity
    module, 19
PyPop.parsers
    module, 22
PyPop.popaggregate
    module, 25
PyPop.popanalysis
    module, 27
PyPop.popmeta
    module, 29
PyPop.pypop
    module, 29
PyPop.randombinning
    module, 29
PyPop.utils
    module, 30
PyPop.xslt
    module, 39
PyPopModuleRenameDeprecationWarning, 43

R

RandomBinsForHomozygosity (*class in PyPop.randombinning*), 29
randomMethod() (*RandomBinsForHomozygosity method*), 30
returnBulkHomozygosityStats() (*HomozygosityEWSlatkinExact method*), 21
runArlequin() (*ArlequinBatch method*), 43
runArlequin() (*ArlequinWrapper method*), 41
runArlequin() (*HaploArlequin method*), 16

S

sequenceMethod() (*RandomBinsForHomozygosity method*), 30
serializeAlleleCountDataAt() (*AlleleCounts method*), 7
serializeAlleleCountDataAt() (*Genotypes method*), 6
serializeAlleleCountDataTo() (*Genotypes method*), 6
serializeEnd() (*Emhaplofreq method*), 14
serializeEnd() (*Haplostats method*), 15
serializeHomozygosityTo() (*Homozygosity method*), 21
serializeHomozygosityTo() (*HomozygosityEWSlatkinExact method*), 21
serializeMetadataTo() (*ParseFile method*), 23
serializeStart() (*Emhaplofreq method*), 13
serializeStart() (*Haplostats method*), 15

serializeSubclassMetadataTo() (*AlleleCounts method*), 7
serializeSubclassMetadataTo() (*Genotypes method*), 6
serializeSubclassMetadataTo() (*ParseAlleleCountFile method*), 25
serializeSubclassMetadataTo() (*ParseGenotypeFile method*), 24
serializeTo() (*HardyWeinberg method*), 17
serializeTo() (*HardyWeinbergEnumeration method*), 18
serializeTo() (*HardyWeinbergGuoThompsonArlequin method*), 19
serializeTo() (*HomozygosityEWSlatkinExactPairwise method*), 22
serializeXMLTableTo() (*HardyWeinberg method*), 17
setup_logger() (*in module PyPop*), 44
splitIntoNGroups() (*in module PyPop.utils*), 39
startFiltering() (*AnthonyNolanFilter method*), 10
startFiltering() (*Filter method*), 8
startFiltering() (*PassThroughFilter method*), 9
startFirstPass() (*AnthonyNolanFilter method*), 10
startFirstPass() (*Filter method*), 8
startFirstPass() (*PassThroughFilter method*), 9
StringMatrix (*class in PyPop.utils*), 32
SubclassError, 8

T

tagContents() (*XMLOutputStream method*), 32
TextOutputStream (*class in PyPop.utils*), 31
translate_file_to_file() (*in module PyPop.popaggregate*), 26
translate_file_to_stdout() (*in module PyPop.popaggregate*), 26
translate_string_to_file() (*in module PyPop.popaggregate*), 26
translate_string_to_stdout() (*in module PyPop.popaggregate*), 26
translateMatrix() (*AnthonyNolanFilter method*), 11

U

unique_elements() (*in module PyPop.utils*), 37
usage_message (*in module PyPop.arlequin*), 43
use_scipy (*in module PyPop.hardyweinberg*), 17

W

write() (*TextOutputStream method*), 31
writeln() (*TextOutputStream method*), 31
writeToLog() (*AnthonyNolanFilter method*), 11
writeToLog() (*Filter method*), 8
writeToLog() (*PassThroughFilter method*), 9

X

XMLOutputStream (*class in PyPop.utils*), 31