GenomeTools Documentation

Release 0.3.4

Florian Wagner

Aug 04, 2017

Contents

GenomeTools is a collection of Python scripts, classes and functions that perform general tasks encountered in the analysis of genomic data. GenomeTools is free and open-source software (see *License*).

CHAPTER 1

Contents

Overview

Main Features

- Basic classes for working with genomic data: *genometools.basic*
- Classes for working with expression data: *genometools.expression*
- Classes for working with Gene Ontology data: *genometools.ontology*
- Classes for performing GO enrichment analyses: *genometools.enrichment*
- Miscellaneous convenience functions: *genometools.misc*

Demo Notebooks

- Scripts.ipynb (`download`): GenomeTools command-line scripts

Installation

GenomeTools can be installed from PyPI using pip:

```
$ pip install genometools
```

API Reference

Basic classes for working with genomic data (`genometools.basic`)

GeneSet	A gene set.
GeneSetCollection	A collection of gene sets.

class genometools.basic.**GeneSet**(*id*, *name*, *genes*, *source=None*, *collection=None*, *description=None*)

A gene set.

A gene set is just what the name implies: A set of genes. Usually, gene sets are used to group genes that share a certain property (e.g., genes that perform related functions, or genes that are frequently co-expressed). The genes in the gene set are not ordered.

GeneSet instances are hashable and should therefore be considered to be immutable.

Parameters

- **id** (*str*) – See `id` attribute.
- **name** (*str*) – See *name* attribute.
- **genes** (*set, list or tuple of str*) – See *genes* attribute.
- **source** (*str, optional*) – See *source* attribute. (None)
- **collection** (*str, optional*) – See *collection* attribute. (None)
- **description** (*str, optional*) – See *description* attribute. (None)

id_
str – The (unique) ID of the gene set.

name
str – The name of the gene set.

genes
set of str – The list of genes in the gene set.

source
None or str – The source / origin of the gene set (e.g., "MSigDB")

collection
None or str – The collection that the gene set belongs to (e.g., "c4" for gene sets from MSigDB).

description
None or str – The description of the gene set.

classmethod from_list(*l*)
Generate an GeneSet object from a list of strings.

Note: See also *to_list()*.

Parameters **l** (*list or tuple of str*) – A list of strings representing gene set ID, name, genes, source, collection, and description. The genes must be comma-separated. See also *to_list()*.

Returns The gene set.

Return type *genometools.basic.GeneSet*

hash
MD5 hash value for the gene set.

size
The size of the gene set (i.e., the number of genes in it).

to_list()
Converts the GeneSet object to a flat list of strings.

Note: see also *from_list()*.

Returns The data from the GeneSet object as a flat list.

Return type list of str

class genometools.basic.**GeneSetCollection**(*gene_sets*)
A collection of gene sets.

This is a class that basically just contains a list of gene sets, and supports different ways of accessing individual gene sets. The gene sets are ordered, so each gene set has a unique position (index) in the database.

Parameters **gene_sets** (list or tuple of *GeneSet*) – See *gene_sets* attribute.

gene_sets
tuple of *GeneSet* – The list of gene sets in the database. Note that this is a read-only property.

get_by_id(*id_*)
Look up a gene set by its ID.

Parameters **id** (str) – The ID of the gene set.

Returns The gene set.

Return type *GeneSet*

Raises ValueError – If the given ID is not in the database.

get_by_index(*i*)
Look up a gene set by its index.

Parameters **i** (int) – The index of the gene set.

Returns The gene set.

Return type *GeneSet*

Raises ValueError – If the given index is out of bounds.

index(*id_*)
Get the index corresponding to a gene set, identified by its ID.

Parameters **id** (str) – The ID of the gene set.

Returns The index of the gene set.

Return type *int*

Raises ValueError – If the given ID is not in the database.

n
The number of gene sets in the database.

classmethod **read_msigdb_xml**(*path*, *entrez2gene*, *species=None*)
Read the complete MSigDB database from an XML file.

The XML file can be downloaded from here: http://software.broadinstitute.org/gsea/msigdb/download_file.jsp?filePath=/resources/msigdb/5.0/msigdb_v5.0.xml

Parameters

- **path** (str) – The path name of the XML file.
- **entrez2gene** (dict *or* OrderedDict *(str: str)*) – A dictionary mapping Entrez Gene IDs to gene symbols (names).

- **species** (*str, optional*) – A species name (e.g., "Homo_sapiens"). Only gene sets for that species will be retained. (None)

Returns The gene set database containing the MSigDB gene sets.

Return type *GeneSetCollection*

classmethod read_tsv(*path*, *encoding=u'utf-8'*)
Read a gene set database from a tab-delimited text file.

Parameters

- **path** (*str*) – The path name of the the file.
- **encoding** (*str*) – The encoding of the text file.

Returns

Return type None

write_tsv(*path*)
Write the database to a tab-delimited text file.

Parameters **path** (*str*) – The path name of the file.

Returns

Return type None

Classes for working with expression data (genometools.expression)

Classes for visualizing expression data (genometools.expression.visualize)

ExpHeatmap	An expression heatmap.
SampleCorrelationHeatmap	A sample correlation heatmap.

Expression heatmaps (ExpHeatmap)

class genometools.expression.visualize.**ExpHeatmap**(*matrix*, *gene_annotations=None*, *sample_annotations=None*, *colorscale=None*, *colorbar_label=u'Expression'*, *title=None*)
An expression heatmap.

An expression heatmap visualizes a gene expression matrix, which is a two-dimensional numerical array with rows corresponding to genes, and columns corresponding to samples.

Parameters

- **matrix** (*ExpMatrix*) – See `matrix` attribute.
- **gene_annotations** (list of *HeatmapGeneAnnotation*, or None, optional) – A list of gene annotations. [None]
- **sample_annotations** (list of *HeatmapSampleAnnotation*, or None, optional) – A list of sample annotations. [None]
- **colorscale** (*list or None, optional*) – A plotly colorscale (see `read_colorscale()`). If None, load the default red-blue colorscale that is included in this package. [None]

- **colorbar_label** (*str or None, optional*) – The colorbar label. If None, "Expression" will be used. [None]
- **title** (*str or None, optional*) – The figure title. If None, the figure will have no title.

Notes

This class provides an intermediate layer between the underlying expression data, which is represented by an *ExpMatrix* object, and the visualization itself, which corresponds to a plotly figure. Its purpose is to store specific additional data such as the figure title, colorbar label, and (visual) annotations, but no data that only concerns the layout or visual appearance of the figure (e.g., its dimensions, margins, font choices, or the expression values corresponding to the lower and upper end of the colorscale. This information is provided by the user when he/she calls the *get_figure()* function, and is not stored anywhere beside the plotly figure object itself.

Gene and sample annotations are represented by *HeatmapGeneAnnotation* and `HeatmapSampleAnnotations` objects, which can be used to highlight individual rows and columns in the heatmap, respectively.

get_figure(*emin=None*, *emax=None*, *width=800*, *height=400*, *margin_left=100*, *margin_bottom=60*, *margin_top=30*, *margin_right=0*, *colorbar_size=0.4*, *xaxis_label=None*, *yaxis_label=None*, *xaxis_nticks=None*, *yaxis_nticks=None*, *xtick_angle=30*, *font=u"'Droid Serif", "Open Serif", serif'*, *font_size=12*, *title_font_size=None*, *show_sample_labels=True*, ***kwargs*)

Generate a plotly figure of the heatmap.

Parameters

- **emin** (*int, float, or None, optional*) – The expression value corresponding to the lower end of the colorscale. If None, determine, automatically. [None]
- **emax** (*int, float, or None, optional*) – The expression value corresponding to the upper end of the colorscale. If None, determine automatically. [None]
- **margin_left** (*int, optional*) – The size of the left margin (in px). [100]
- **margin_right** (*int, optional*) – The size of the right margin (in px). [0]
- **margin_top** (*int, optional*) – The size of the top margin (in px). [30]
- **margin_bottom** (*int, optional*) – The size of the bottom margin (in px). [60]
- **colorbar_size** (*int or float, optional*) – The sze of the colorbar, relative to the figure size. [0.4]
- **xaxis_label** (*str or None, optional*) – X-axis label. If None, use *ExpMatrix* default. [None]
- **yaxis_label** (*str or None, optional*) – y-axis label. If None, use *ExpMatrix* default. [None]
- **xtick_angle** (*int or float, optional*) – X-axis tick angle (in degrees). [30]
- **font** (*str, optional*) – Name of font to use. Can be multiple, separated by comma, to specify a prioritized list. [' "Droid Serif", "Open Serif", "serif"']
- **font_size** (*int or float, optional*) – Font size to use throughout the figure, in points. [12]
- **title_font_size** (*int or float or None, optional*) – Font size to use for labels on axes and the colorbar. If None, use `font_size` value. [None]

- **show_sample_labels** (bool, *optional*) – Whether to show the sample labels. [True]

Returns The plotly figure.

Return type plotly.graph_objs.Figure

Heatmap annotations

class genometools.expression.visualize.**HeatmapGeneAnnotation**(*gene*, *color*, ***kwargs*)

An *ExpHeatmap* gene (row) annotation.

gene
(property) The gene to be annotated.

class genometools.expression.visualize.**HeatmapSampleAnnotation**(*sample*, *color*, ***kwargs*)

An *ExpHeatmap* sample (column) annotation.

sample
(property) The sample to be annotated.

Sample correlation heatmaps (SampleCorrelationHeatmap)

class genometools.expression.visualize.**SampleCorrelationHeatmap**(*corr_matrix*, *sample_annotations=None*, *block_annotations=None*, *colorscale=None*, *colorbar_label=None*)

A sample correlation heatmap.

get_figure(***kwargs*)
Get a plotly figure of the heatmap.

Correlation heatmap annotations

class genometools.expression.visualize.**HeatmapBlockAnnotation**(*start_index*, *end_index*, ***kwargs*)

A *SampleCorrelationHeatmap* block annotation.

ExpGene	A gene in a gene expression analysis.
ExpGenome	A complete set of genes in a gene expression analysis.
ExpMatrix	A gene expression matrix.
ExpProfile	A gene expression profile.

class genometools.expression.**ExpGene**(*name*, *chromosome=None*, *position=None*, *length=None*, *ensembl_id=None*, *source=None*, *type_=None*)

A gene in a gene expression analysis.

Instances are to be treated as immutable, to allow use of ExpGene objects to be used in sets etc.

Parameters

- **name** (str) – See *name* attribute.
- **chromosome** (str or None, optional) – See *chromosome* attribute. [None]
- **position** (int or None, optional) – See *position* attribute. [None]
- **length** (int or None, optional) – See length attribute. [None]
- **ensembl_id** (str or None, optional) – See *ensembl_id* attribute. [None]

name
str – The gene name (use the official gene symbol, if available).

chromosome
str or None – The chromosome that the gene is located on.

position
int or None – The chromosomal location (base-pair index) of the gene. The sign of the this attribute indicates whether the gene is on the plus or minus strand. Base pair indices are 0-based.

ensembl_id
list of str – The Ensembl ID of the gene.

classmethod from_dict(*data*)
Generate an *ExpGene* gene object from a dictionary.

Parameters **data** (dict) – A dictionary with keys corresponding to attribute names. Attributes with missing keys will be assigned None. See also to_list().

Returns The gene.

Return type *ExpGene*

class genometools.expression.**ExpGenome**(*genes*)
A complete set of genes in a gene expression analysis.

The class represents a "genome" in the form of an ordered set of genes. This means that each gene has an index value, i.e. an integer indicating its 0-based position in the genome.

Parameters **genes** (Iterable of *ExpGene* objects) – See *genes* attribute.

genes
list of *ExpGene* – The genes in the genome.

Notes

The implementation is very similar to the *genometools.basic.GeneSetCollection* class. It uses ordered dictionaries to support efficient access by gene name or index, as well as looking up the index of specific gene.

classmethod from_gene_names(*names*)
Generate a genome from a list of gene names.

Parameters **names** (*Iterable of str*) – The list of gene names.

Returns The genome.

Return type *ExpGenome*

gene_names
Returns a list of all gene names.

gene_set
Returns a set of all genes.

genes
Returns a list with all genes.

hash
Returns an MD5 hash value for the genome.

index(*gene_or_name*)
Returns the index of a given gene.

The index is 0-based, so the first gene in the genome has the index 0, and the last one has index `len(genome) - 1`.

Parameters **gene_or_name** (`str`) – The gene or its name (symbol).

Returns The gene index.

Return type *int*

classmethod read_tsv(*path*, *encoding=u'UTF-8'*)
Read genes from tab-delimited text file.

Parameters

- **path** (`str`) – The path of the text file.
- **encoding** (`str, optional`) – The file encoding. ('UTF-8')

Returns

Return type None

write_tsv(*path*, *encoding=u'UTF-8'*, *overwrite=False*)
Write genes to tab-delimited text file in alphabetical order.

Parameters

- **path** (`str`) – The path of the output file.
- **encoding** (`str, optional`) – The file encoding. ["UTF-8"]

Returns

Return type None

class `genometools.expression.`**ExpProfile**(**args*, ***kwargs*)
A gene expression profile.

This class inherits from `pandas.Series`.

Parameters

- **x** (1-dimensional `numpy.ndarray`) – See *x* attribute.
- **Parameters** (*Additional*) –
- **----------** –
- **genes** (*list or tuple of str*) – See *genes* attribute.
- **name** (`str`) – See `name` attribute.
- **Parameters** –
- **----------** –
- **pandas.Series parameters.** (*All*) –

x
1-dimensional `numpy.ndarray` – The vector with expression values.

genes
pandas.Index – Alias for pandas.Series.index. Contains the names of the genes in the matrix.

label
str – Alias for pandas.Series.name. The sample label.

filter_against_genome(*genome*)
Filter the expression matrix against a _genome (set of genes).

Parameters **genome** (*genometools.expression.ExpGenome*) – The genome to filter the genes against.

Returns The filtered expression matrix.

Return type *ExpMatrix*

genes
Alias for Series.index.

genome
Get an *ExpGenome* representation of the genes in the profile.

label
Alias for Series.name.

p
The number of genes.

classmethod read_tsv(*path*, *genome=None*, *encoding=u'UTF-8'*)
Read expression profile from a tab-delimited text file.

Parameters

- **path** (str) – The path of the text file.
- **genome** (*ExpGenome* object, optional) – The set of valid genes. If given, the genes in the text file will be filtered against this set of genes. (None)
- **encoding** (str, *optional*) – The file encoding. ("UTF-8")

Returns The expression profile.

Return type *ExpProfile*

sort_genes(*inplace=False*)
Sort the rows of the profile alphabetically by gene name.

Parameters **inplace** (bool, *optional*) – If set to True, perform the sorting in-place.

Returns

Return type None

Notes

pandas 0.18.0's Series.sort_index method does not support the kind keyword, which is needed to select a stable sort algorithm.

write_tsv(*path*, *encoding=u'UTF-8'*)
Write expression matrix to a tab-delimited text file.

Parameters

- **path** (str) – The path of the output file.

- **encoding** (str, *optional*) – The file encoding. ("UTF-8")

Returns

Return type None

x
Alias for `Series.values`.

class genometools.expression.**ExpMatrix**(**args*, ***kwargs*)
A gene expression matrix.

This class inherits from `pandas.DataFrame`.

Parameters

- **X** (2-dimensional `numpy.ndarray`) – See *X* attribute.
- **Parameters** (*Additional*) –
- **----------** –
- **genes** (*list or tuple of str*) – See *genes* attribute.
- **samples** (*list or tuple of str*) – See *samples* attribute.
- **Parameters** –
- **----------** –
- **pandas.DataFrame parameters.** (*All*) –

genes
tuple of str – The names of the genes (rows) in the matrix.

samples
tuple of str – The names of the samples (columns) in the matrix.

X
2-dimensional `numpy.ndarray` – The matrix of expression values.

X
Alias for `DataFrame.values`.

center_genes(*use_median=False*, *inplace=False*)
Center the expression of each gene (row).

filter_against_genome(*genome*, *inplace=False*)
Filter the expression matrix against a _genome (set of genes).

Parameters

- **genome** (*genometools.expression.ExpGenome*) – The genome to filter the genes against.
- **inplace** (bool, *optional*) – Whether to perform the operation in-place.

Returns The filtered expression matrix.

Return type *ExpMatrix*

genes
Alias for `DataFrame.index`.

genome
Get an *ExpGenome* representation of the genes in the matrix.

get_figure(*heatmap_kw=None*, ***kwargs*)
Generate a plotly figure showing the matrix as a heatmap.

This is a shortcut for `ExpMatrix.get_heatmap(...).get_figure(...)`.

See `ExpHeatmap.get_figure()` for keyword arguments.

Parameters **heatmap_kw** (*dict or None*) – If not None, dictionary containing keyword arguments to be passed to the *ExpHeatmap* constructor.

Returns The plotly figure.

Return type `plotly.graph_objs.Figure`

get_heatmap(*highlight_genes=None*, *highlight_samples=None*, *highlight_color=None*, ***kwargs*)
Generate a heatmap (*ExpHeatmap*) of the matrix.

See `ExpHeatmap` constructor for keyword arguments.

Parameters

- **highlight_genes** (*list of str*) – List of genes to highlight
- **highlight_color** (`str`) – Color to use for highlighting

Returns The heatmap.

Return type *ExpHeatmap*

n
The number of samples.

p
The number of genes.

classmethod read_tsv(*path*, *genome=None*, *encoding=u'UTF-8'*)
Read expression matrix from a tab-delimited text file.

Parameters

- **path** (`str`) – The path of the text file.
- **genome** (*ExpGenome* object, optional) – The set of valid genes. If given, the genes in the text file will be filtered against this set of genes. (None)
- **encoding** (*str, optional*) – The file encoding. ("UTF-8")

Returns The expression matrix.

Return type *ExpMatrix*

sample_correlations
Returns an *ExpMatrix* containing all pairwise sample correlations.

Returns The sample correlation matrix.

Return type *ExpMatrix*

samples
Alias for `DataFrame.columns`.

sort_genes(*stable=True*, *inplace=False*, *ascending=True*)
Sort the rows of the matrix alphabetically by gene name.

Parameters

- **stable** (*bool, optional*) – Whether to use a stable sorting algorithm. [True]

- **inplace** (bool, *optional*) – Whether to perform the operation in place.[False]
- **ascending** (bool, *optional*) – Whether to sort in ascending order [True]

Returns The sorted matrix.

Return type *ExpMatrix*

sort_samples(*stable=True*, *inplace=False*, *ascending=True*)
Sort the columns of the matrix alphabetically by sample name.

Parameters

- **stable** (bool, *optional*) – Whether to use a stable sorting algorithm. [True]
- **inplace** (bool, *optional*) – Whether to perform the operation in place.[False]
- **ascending** (bool, *optional*) – Whether to sort in ascending order [True]

Returns The sorted matrix.

Return type *ExpMatrix*

standardize_genes(*inplace=False*)
Standardize the expression of each gene (row).

write_tsv(*path*, *encoding=u'UTF-8'*)
Write expression matrix to a tab-delimited text file.

Parameters

- **path** (str) – The path of the output file.
- **encoding** (str, *optional*) – The file encoding. ("UTF-8")

Returns

Return type None

genometools.expression.**quantile_normalize**(*matrix*, *inplace=False*, *target=None*)
Quantile normalization, allowing for missing values (NaN).

In case of nan values, this implementation will calculate evenly distributed quantiles and fill in the missing data with those values. Quantile normalization is then performed on the filled-in matrix, and the nan values are restored afterwards.

Parameters

- **matrix** (*ExpMatrix*) – The expression matrix (rows = genes, columns = samples).
- **inplace** (bool) – Whether or not to perform the operation in-place. [False]
- **target** (numpy.ndarray) – Target distribution to use. needs to be a vector whose first dimension matches that of the expression matrix. If None, the target distribution is calculated based on the matrix itself. [None]

Returns The normalized matrix.

Return type numpy.ndarray (ndim = 2)

genometools.expression.**filter_variance**(*matrix*, *top*)
Filter genes in an expression matrix by variance.

Parameters

- **matrix** (ExpMatrix) – The expression matrix.
- **top** (int) – The number of genes to retain.

Returns The filtered expression matrix.

Return type *ExpMatrix*

class genometools.expression.**ExpMatrix**(**args*, ***kwargs*)
A gene expression matrix.

This class inherits from pandas.DataFrame.

Parameters

- **X** (2-dimensional numpy.ndarray) – See *X* attribute.
- **Parameters** (*Additional*) –
- **----------** –
- **genes** (*list or tuple of str*) – See *genes* attribute.
- **samples** (*list or tuple of str*) – See *samples* attribute.
- **Parameters** –
- **----------** –
- **pandas.DataFrame parameters.** (*All*) –

genes
tuple of str – The names of the genes (rows) in the matrix.

samples
tuple of str – The names of the samples (columns) in the matrix.

X
2-dimensional numpy.ndarray – The matrix of expression values.

Classes for working with Gene Ontology data (genometools.ontology)

GOTerm	A GO term.
GeneOntology	A Gene Ontology.

Sub-package for working with Gene Ontology data.

Classes for performing gene set enrichment analysis (genometools.enrichment)

GeneSetEnrichmentAnalysis	Test a set of genes or a ranked list of genes for gene set enrichment.
StaticGSEResult	Result of a hypergeometric test for gene set enrichment.
RankBasedGSEResult	Result of an XL-mHG-based test for gene set enrichment.

class genometools.enrichment.**GeneSetEnrichmentAnalysis**(*genome*, *gene_set_coll*)
Test a set of genes or a ranked list of genes for gene set enrichment.

Parameters

- **genome** (*ExpGenome* object) – See *_genome* attribute.
- **gene_set_coll** (*GeneSetCollection* object) – See *_gene_set_coll* attribute.

_genome
ExpGenome object – The universe of genes.

_gene_set_coll
GeneSetCollection object – The list of gene sets to be tested.

Notes

The class is initialized with a set of valid gene names (an *ExpGenome* object), as well as a set of gene sets (a *GeneSetCollection* object). During initialization, a binary "gene-by-gene set" matrix is constructed, which stores information about which gene is contained in each gene set. This matrix is quite sparse, and requires a significant amount of memory. As an example, for a set of p = 10,000 genes and n = 10,000 gene sets, this matrix is of size 100 MB in the memory (i.e., p x n bytes).

Once the class has been initialized, the function *get_static_enrichment* can be used to test a set of genes for gene set enrichment, and the function *get_rank_based_enrichment* can be used to test a ranked list of genes for gene set enrichment.

Note also that two conventions get mixed here: In the *GeneSet* class, a gene simply corresponds to a string containing the gene name, whereas in the *ExpGenome* class, a gene is an *ExpGene* object. Here, we represent genes as simple strings, since the user can always obtain the corresponding *ExpGene* object from the *ExpGenome* genome.

get_rank_based_enrichment(*ranked_genes*, *pval_thresh=0.05*, *X_frac=0.25*, *X_min=5*, *L=None*, *adjust_pval_thresh=True*, *escore_pval_thresh=None*, *exact_pval=u'always'*, *gene_set_ids=None*, *table=None*)

Test for gene set enrichment at the top of a ranked list of genes.

This function uses the XL-mHG test to identify enriched gene sets.

This function also calculates XL-mHG E-scores for the enriched gene sets, using `escore_pval_thresh` as the p-value threshold "psi".

Parameters

- **ranked_genes** (*list of str*) – The ranked list of genes.
- **pval_thresh** (`float`, *optional*) – The p-value threshold used to determine significance. See also `adjust_pval_thresh`. [0.05]
- **X_frac** (`float`, *optional*) – The min. fraction of genes from a gene set required for enrichment. [0.25]
- **X_min** (`int`, *optional*) – The min. no. of genes from a gene set required for enrichment. [5]
- **L** (`int`, *optional*) – The lowest cutoff to test for enrichment. If `None`, int(0.25*(no. of genes)) will be used. [None]
- **adjust_pval_thresh** (`bool`, *optional*) – Whether to adjust the p-value thershold for multiple testing, using the Bonferroni method. [True]
- **escore_pval_thresh** (*float or None*, *optional*) – The "psi" p-value threshold used in calculating E-scores. [None]
- **exact_pval** (`str`) – Choices are: "always", "if_significant", "if_necessary". Parameter will be passed to `xlmhg.get_xlmhg_test_result`. ["always"]
- **gene_set_ids** (*list of str or None*, *optional*) – A list of gene set IDs to specify which gene sets should be tested for enrichment. If `None`, all gene sets will be tested. [None]

- **table** (*2-dim numpy.ndarray of type numpy.longdouble or None, optional*) – The dynamic programming table used by the algorithm for calculating XL-mHG p-values. Passing this avoids memory re-allocation when calling this function repetitively. [None]

Returns A list of all significantly enriched gene sets.

Return type list of *RankBasedGSEResult*

get_static_enrichment(*genes*, *pval_thresh*, *adjust_pval_thresh=True*, *K_min=3*, *gene_set_ids=None*)

Find enriched gene sets in a set of genes.

Parameters

- **genes** (*set of str*) – The set of genes to test for gene set enrichment.
- **pval_thresh** (float) – The significance level (p-value threshold) to use in the analysis.
- **adjust_pval_thresh** (bool, *optional*) – Whether to adjust the p-value threshold using a Bonferroni correction. (Warning: This is a very conservative correction!) [True]
- **K_min** (int, *optional*) – The minimum number of gene set genes present in the analysis. [3]
- **gene_set_ids** (*Iterable or None*) – A list of gene set IDs to test. If None, all gene sets are tested that meet the K_min criterion.

Returns A list of all significantly enriched gene sets.

Return type list of *StaticGSEResult*

class genometools.enrichment.**StaticGSEResult**(*gene_set*, *N*, *n*, *selected_genes*, *pval*)

Result of a hypergeometric test for gene set enrichment.

Parameters

- **gene_set** (*genometools.basic.GeneSet*) – See *gene_set*.
- **N** (int) – See *N*.
- **n** (int) – See *n*.
- **selected_genes** (iterable of *ExpGene*) – See *selected_genes*.
- **pval** (float) – See *pval*.

gene_set

genometools.basic.GeneSet – The gene set.

N

int – The total number of genes in the analysis.

n

int – The number of genes selected.

selected_genes

set of *ExpGene* – The genes from the gene set found present.

pval

float – The hypergeometric p-value.

fold_enrichment
Returns the fold enrichment of the gene set.

Fold enrichment is defined as ratio between the observed and the expected number of gene set genes present.

get_pretty_format(*max_name_length=0*)
Returns a nicely formatted string describing the result.

Parameters **max_name_length** (int *[0]*) – The maximum length of the gene set name (in characters). If the gene set name is longer than this number, it will be truncated and "..." will be appended to it, so that the final string exactly meets the length requirement. If 0 (default), no truncation is performed. If not 0, must be at least 3.

Returns The formatted string.

Return type *str*

Raises ValueError – If an invalid length value is specified.

class genometools.enrichment.**RankBasedGSEResult**(*gene_set*, *N*, *indices*, *ind_genes*, *X*, *L*, *stat*, *cutoff*, *pval*, *pval_thresh=None*, *escore_pval_thresh=None*, *escore_tol=None*)
Result of an XL-mHG-based test for gene set enrichment.

This class inherits from xlmhg.mHGResult.

Parameters

- **gene_set** (*genometools.basic.GeneSet*) – See *gene_set* attribute.
- **N** (int) – The total number of genes in the ranked list. See also xlmhg.mHGResult.N.
- **indices** (numpy.ndarray of integers) – The indices of the gene set genes in the ranked list.
- **ind_genes** (*list of str*) – See *ind_genes* attribute.
- **X** (int) – The XL-mHG X parameter.
- **L** (int) – The XL-mHG L parameter.
- **stat** (float) – The XL-mHG test statistic.
- **cutoff** (int) – The cutoff at which the XL-mHG test statistic was attained.
- **pval** (float) – The XL-mHG p-value.
- **pval_thresh** (float, *optional*) – The p-value threshold used in the analysis. [None]
- **escore_pval_thresh** (float, *optional*) – The hypergeometric p-value threshold used for calculating the E-score. If not specified, the XL-mHG p-value will be used, resulting in a conservative E-score. [None]
- **escore_tol** (float, *optional*) – The tolerance used for calculating the E-score. [None]

gene_set
genometools.basic.GeneSet – The gene set.

ind_genes
list of str – The names of the genes corresponding to the indices.

Miscellaneous convenience functions (genometools.misc)

exception genometools.misc.**ArithmeticError**
Base class for arithmetic errors.

exception genometools.misc.**AssertionError**
Assertion failed.

exception genometools.misc.**AttributeError**
Attribute not found.

exception genometools.misc.**BaseException**
Common base class for all exceptions

exception genometools.misc.**BufferError**
Buffer error.

exception genometools.misc.**BytesWarning**
Base class for warnings about bytes and buffer related problems, mostly related to conversion from str or comparing to str.

exception genometools.misc.**DeprecationWarning**
Base class for warnings about deprecated features.

exception genometools.misc.**EOFError**
Read beyond end of file.

exception genometools.misc.**EnvironmentError**
Base class for I/O related errors.

errno
exception errno

filename
exception filename

strerror
exception strerror

exception genometools.misc.**Exception**
Common base class for all non-exit exceptions.

exception genometools.misc.**FloatingPointError**
Floating point operation failed.

exception genometools.misc.**FutureWarning**
Base class for warnings about constructs that will change semantically in the future.

exception genometools.misc.**GeneratorExit**
Request that a generator exit.

exception genometools.misc.**IOError**
I/O operation failed.

exception genometools.misc.**ImportError**
Import can't find module, or can't find name in module.

exception genometools.misc.**ImportWarning**
Base class for warnings about probable mistakes in module imports

exception genometools.misc.**IndentationError**
Improper indentation.

exception genometools.misc.**IndexError**
Sequence index out of range.

exception genometools.misc.**KeyError**
Mapping key not found.

exception genometools.misc.**KeyboardInterrupt**
Program interrupted by user.

exception genometools.misc.**LookupError**
Base class for lookup errors.

exception genometools.misc.**MemoryError**
Out of memory.

exception genometools.misc.**NameError**
Name not found globally.

exception genometools.misc.**NotImplementedError**
Method or function hasn't been implemented yet.

exception genometools.misc.**OSError**
OS system call failed.

exception genometools.misc.**OverflowError**
Result too large to be represented.

exception genometools.misc.**PendingDeprecationWarning**
Base class for warnings about features which will be deprecated in the future.

exception genometools.misc.**ReferenceError**
Weak ref proxy used after referent went away.

exception genometools.misc.**RuntimeError**
Unspecified run-time error.

exception genometools.misc.**RuntimeWarning**
Base class for warnings about dubious runtime behavior.

exception genometools.misc.**StandardError**
Base class for all standard Python exceptions that do not represent interpreter exiting.

exception genometools.misc.**StopIteration**
Signal the end from iterator.next().

exception genometools.misc.**SyntaxError**
Invalid syntax.

filename
exception filename

lineno
exception lineno

msg
exception msg

offset
exception offset

print_file_and_line
exception print_file_and_line

text
exception text

exception genometools.misc.**SyntaxWarning**
Base class for warnings about dubious syntax.

exception genometools.misc.**SystemError**
Internal error in the Python interpreter.

Please report this to the Python maintainer, along with the traceback, the Python version, and the hardware/OS platform and version.

exception genometools.misc.**SystemExit**
Request to exit from the interpreter.

code
exception code

exception genometools.misc.**TabError**
Improper mixture of spaces and tabs.

exception genometools.misc.**TypeError**
Inappropriate argument type.

exception genometools.misc.**UnboundLocalError**
Local name referenced but not bound to a value.

exception genometools.misc.**UnicodeDecodeError**
Unicode decoding error.

encoding
exception encoding

end
exception end

object
exception object

reason
exception reason

start
exception start

exception genometools.misc.**UnicodeEncodeError**
Unicode encoding error.

encoding
exception encoding

end
exception end

object
exception object

reason
exception reason

start
exception start

exception genometools.misc.**UnicodeError**
Unicode related error.

exception genometools.misc.**UnicodeTranslateError**
Unicode translation error.

encoding
exception encoding

end
exception end

object
exception object

reason
exception reason

start
exception start

exception genometools.misc.**UnicodeWarning**
Base class for warnings about Unicode related problems, mostly related to conversion problems.

exception genometools.misc.**UserWarning**
Base class for warnings generated by user code.

exception genometools.misc.**ValueError**
Inappropriate argument value (of correct type).

exception genometools.misc.**Warning**
Base class for warning categories.

exception genometools.misc.**ZeroDivisionError**
Second argument to a division or modulo operation was zero.

genometools.misc.**abs**(*number*) → number
Return the absolute value of the argument.

genometools.misc.**all**(*iterable*) → bool
Return True if bool(x) is True for all values x in the iterable. If the iterable is empty, return True.

genometools.misc.**any**(*iterable*) → bool
Return True if bool(x) is True for any x in the iterable. If the iterable is empty, return False.

genometools.misc.**apply**(*object*[, *args*[, *kwargs*]]) → value
Call a callable object with positional arguments taken from the tuple args, and keyword arguments taken from the optional dictionary kwargs. Note that classes are callable, as are instances with a __call__() method.

Deprecated since release 2.3. Instead, use the extended call syntax: function(*args, **keywords).

genometools.misc.**argmax**(*seq*)
Obtains the index of the largest element in a list.

Parameters **seq** (*List*) – The list

Returns The index of the largest element.

Return type *int*

genometools.misc.**argmin**(*seq*)
Obtains the index of the smallest element in a list.

Parameters **seq** (*List*) – The list.

Returns The index of the smallest element.

Return type *int*

genometools.misc.**argsort**(*seq*)
Returns a list of indices that would sort a list.

Parameters **seq** (*List*) – The list.

Returns The list of indices that would sort the given list `seq`.

Return type List[*int*]

Notes

If the returned list of indices can be a NumPy array, use `numpy.lexsort` instead. If the given list `seq` is a NumPy array, use `numpy.argsort` instead.

genometools.misc.**ascii**(*object*) → string
Return the same as repr(). In Python 3.x, the repr() result will contain printable characters unescaped, while the ascii() result will have such characters backslash-escaped.

class genometools.misc.**basestring**
Type basestring cannot be instantiated; it is the base for str and unicode.

genometools.misc.**bin**(*number*) → string
Return the binary representation of an integer or long integer.

genometools.misc.**bisect_index**(*a*, *x*)
Find the leftmost index of an element in a list using binary search.

Parameters

- **a** (`list`) – A sorted list.
- **x** (*arbitrary*) – The element.

Returns The index.

Return type *int*

class genometools.misc.**bool**(*x*) → bool
Returns True when the argument x is true, False otherwise. The builtins True and False are the only two instances of the class bool. The class bool is a subclass of the class int, and cannot be subclassed.

class genometools.misc.**buffer**(*object*[, *offset*[, *size*]])
Create a new buffer object which references the given object. The buffer will reference a slice of the target object from the start of the object (or at the specified offset). The slice will extend to the end of the target object (or with the specified size).

class genometools.misc.**bytearray**(*iterable_of_ints*) → bytearray.
bytearray(string, encoding[, errors]) -> bytearray. bytearray(bytes_or_bytearray) -> mutable copy of bytes_or_bytearray. bytearray(memory_view) -> bytearray.

Construct a mutable bytearray object from:

- an iterable yielding integers in range(256)
- a text string encoded using the specified encoding
- a bytes or a bytearray object
- any object implementing the buffer API.

bytearray(int) -> bytearray.

Construct a zero-initialized bytearray of the given length.

append (*int*) → None
Append a single item to the end of B.

capitalize () → copy of B
Return a copy of B with only its first character capitalized (ASCII) and the rest lower-cased.

center (*width*[, *fillchar*]) → copy of B
Return B centered in a string of length width. Padding is done using the specified fill character (default is a space).

count (*sub*[, *start*[, *end*]]) → int
Return the number of non-overlapping occurrences of subsection sub in bytes B[start:end]. Optional arguments start and end are interpreted as in slice notation.

decode ([*encoding*[, *errors*]]) → unicode object.
Decodes B using the codec registered for encoding. encoding defaults to the default encoding. errors may be given to set a different error handling scheme. Default is 'strict' meaning that encoding errors raise a UnicodeDecodeError. Other possible values are 'ignore' and 'replace' as well as any other name registered with codecs.register_error that is able to handle UnicodeDecodeErrors.

endswith (*suffix*[, *start*[, *end*]]) → bool
Return True if B ends with the specified suffix, False otherwise. With optional start, test B beginning at that position. With optional end, stop comparing B at that position. suffix can also be a tuple of strings to try.

expandtabs ([*tabsize*]) → copy of B
Return a copy of B where all tab characters are expanded using spaces. If tabsize is not given, a tab size of 8 characters is assumed.

extend (*iterable int*) → None
Append all the elements from the iterator or sequence to the end of B.

find (*sub*[, *start*[, *end*]]) → int
Return the lowest index in B where subsection sub is found, such that sub is contained within B[start,end]. Optional arguments start and end are interpreted as in slice notation.

Return -1 on failure.

fromhex (*string*) → bytearray
Create a bytearray object from a string of hexadecimal numbers. Spaces between two numbers are accepted. Example: bytearray.fromhex('B9 01EF') -> bytearray(b'xb9x01xef').

index (*sub*[, *start*[, *end*]]) → int
Like B.find() but raise ValueError when the subsection is not found.

insert (*index*, *int*) → None
Insert a single item into the bytearray before the given index.

isalnum () → bool
Return True if all characters in B are alphanumeric and there is at least one character in B, False otherwise.

isalpha () → bool
Return True if all characters in B are alphabetic and there is at least one character in B, False otherwise.

isdigit () → bool
Return True if all characters in B are digits and there is at least one character in B, False otherwise.

islower() → bool
Return True if all cased characters in B are lowercase and there is at least one cased character in B, False otherwise.

isspace() → bool
Return True if all characters in B are whitespace and there is at least one character in B, False otherwise.

istitle() → bool
Return True if B is a titlecased string and there is at least one character in B, i.e. uppercase characters may only follow uncased characters and lowercase characters only cased ones. Return False otherwise.

isupper() → bool
Return True if all cased characters in B are uppercase and there is at least one cased character in B, False otherwise.

join(*iterable_of_bytes*) → bytes
Concatenates any number of bytearray objects, with B in between each pair.

ljust(*width*[, *fillchar*]) → copy of B
Return B left justified in a string of length width. Padding is done using the specified fill character (default is a space).

lower() → copy of B
Return a copy of B with all ASCII characters converted to lowercase.

lstrip([*bytes*]) → bytearray
Strip leading bytes contained in the argument. If the argument is omitted, strip leading ASCII whitespace.

partition(*sep*) -> (*head*, *sep*, *tail*)
Searches for the separator sep in B, and returns the part before it, the separator itself, and the part after it. If the separator is not found, returns B and two empty bytearray objects.

pop([*index*]) → int
Remove and return a single item from B. If no index argument is given, will pop the last value.

remove(*int*) → None
Remove the first occurrence of a value in B.

replace(*old*, *new*[, *count*]) → bytes
Return a copy of B with all occurrences of subsection old replaced by new. If the optional argument count is given, only the first count occurrences are replaced.

reverse() → None
Reverse the order of the values in B in place.

rfind(*sub*[, *start*[, *end*]]) → int
Return the highest index in B where subsection sub is found, such that sub is contained within B[start,end]. Optional arguments start and end are interpreted as in slice notation.

Return -1 on failure.

rindex(*sub*[, *start*[, *end*]]) → int
Like B.rfind() but raise ValueError when the subsection is not found.

rjust(*width*[, *fillchar*]) → copy of B
Return B right justified in a string of length width. Padding is done using the specified fill character (default is a space)

rpartition(*sep*) -> (*head*, *sep*, *tail*)
Searches for the separator sep in B, starting at the end of B, and returns the part before it, the separator itself, and the part after it. If the separator is not found, returns two empty bytearray objects and B.

rsplit (*sep*[, *maxsplit*]) → list of bytearray
Return a list of the sections in B, using sep as the delimiter, starting at the end of B and working to the front. If sep is not given, B is split on ASCII whitespace characters (space, tab, return, newline, formfeed, vertical tab). If maxsplit is given, at most maxsplit splits are done.

rstrip ([*bytes*]) → bytearray
Strip trailing bytes contained in the argument. If the argument is omitted, strip trailing ASCII whitespace.

split ([*sep*[, *maxsplit*]]) → list of bytearray
Return a list of the sections in B, using sep as the delimiter. If sep is not given, B is split on ASCII whitespace characters (space, tab, return, newline, formfeed, vertical tab). If maxsplit is given, at most maxsplit splits are done.

splitlines (*keepends=False*) → list of lines
Return a list of the lines in B, breaking at line boundaries. Line breaks are not included in the resulting list unless keepends is given and true.

startswith (*prefix*[, *start*[, *end*]]) → bool
Return True if B starts with the specified prefix, False otherwise. With optional start, test B beginning at that position. With optional end, stop comparing B at that position. prefix can also be a tuple of strings to try.

strip ([*bytes*]) → bytearray
Strip leading and trailing bytes contained in the argument. If the argument is omitted, strip ASCII whitespace.

swapcase () → copy of B
Return a copy of B with uppercase ASCII characters converted to lowercase ASCII and vice versa.

title () → copy of B
Return a titlecased version of B, i.e. ASCII words start with uppercase characters, all remaining cased characters have lowercase.

translate (*table*[, *deletechars*]) → bytearray
Return a copy of B, where all characters occurring in the optional argument deletechars are removed, and the remaining characters have been mapped through the given translation table, which must be a bytes object of length 256.

upper () → copy of B
Return a copy of B with all ASCII characters converted to uppercase.

zfill (*width*) → copy of B
Pad a numeric string B with zeros on the left, to fill a field of the specified width. B is never truncated.

genometools.misc.**bytes**
alias of newbytes

genometools.misc.**callable** (*object*) → bool
Return whether the object is callable (i.e., some kind of function). Note that classes are callable, as are instances with a __call__() method.

genometools.misc.**chr** ()
unichr(i) -> Unicode character

Return a Unicode string of one character with ordinal i; 0 <= i <= 0x10ffff.

class genometools.misc.**classmethod** (*function*) → method
Convert a function to be a class method.

A class method receives the class as implicit first argument, just like an instance method receives the instance. To declare a class method, use this idiom:

class C: @classmethod def f(cls, arg1, arg2, ...):

...

It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class. If a class method is called for a derived class, the derived class object is passed as the implied first argument.

Class methods are different than C++ or Java static methods. If you want those, see the staticmethod builtin.

genometools.misc.**cmp**(*x*, *y*) → integer
Return negative if x<y, zero if x==y, positive if x>y.

genometools.misc.**coerce**(*x*, *y*) -> (*x1*, *y1*)
Return a tuple consisting of the two numeric arguments converted to a common type, using the same rules as used by arithmetic operations. If coercion is not possible, raise TypeError.

genometools.misc.**compile**(*source*, *filename*, *mode*[, *flags*[, *dont_inherit*]]) → code object
Compile the source string (a Python module, statement or expression) into a code object that can be executed by the exec statement or eval(). The filename will be used for run-time error messages. The mode must be 'exec' to compile a module, 'single' to compile a single (interactive) statement, or 'eval' to compile an expression. The flags argument, if present, controls which future statements influence the compilation of the code. The dont_inherit argument, if non-zero, stops the compilation inheriting the effects of any future statements in effect in the code calling compile; if absent or zero these statements do influence the compilation, in addition to any features explicitly specified.

class genometools.misc.**complex**(*real*[, *imag*]) → complex number
Create a complex number from a real part and an optional imaginary part. This is equivalent to (real + imag*1j) where imag defaults to 0.

conjugate() → complex
Return the complex conjugate of its argument. (3-4j).conjugate() == 3+4j.

imag
the imaginary part of a complex number

real
the real part of a complex number

genometools.misc.**configure_logger**(*name*, *log_stream=<open file '<stdout>', mode 'w'>*, *log_file=None*, *log_level=20*, *keep_old_handlers=False*, *propagate=False*)
Configures and returns a logger.

This function serves to simplify the configuration of a logger that writes to a file and/or to a stream (e.g., stdout).

Parameters

- **name** (str) – The name of the logger. Typically set to __name__.
- **log_stream** (*a stream object, optional*) – The stream to write log messages to. If None, do not write to any stream. The default value is sys.stdout.
- **log_file** (str, *optional*) – The path of a file to write log messages to. If None, do not write to any file. The default value is None.
- **log_level** (int, *optional*) – A logging level as defined in Python's logging module. The default value is logging.INFO.
- **keep_old_handlers** (bool, *optional*) – If set to True, keep any pre-existing handlers that are attached to the logger. The default value is False.
- **propagate** (bool, *optional*) – If set to True, propagate the loggers messages to the parent logger. The default value is False.

Returns The logger.

Return type `logging.Logger`

Notes

Note that if `log_stream` and `log_file` are both `None`, no handlers will be created.

`genometools.misc.`**`delattr`**(*object*, *name*)
Delete a named attribute on an object; delattr(x, 'y') is equivalent to ''del x.y''.

`genometools.misc.`**`dict`**
alias of `newdict`

`genometools.misc.`**`dir`**([*object*]) → list of strings
If called without an argument, return the names in the current scope. Else, return an alphabetized list of names comprising (some of) the attributes of the given object, and of attributes reachable from it. If the object supplies a method named __dir__, it will be used; otherwise the default dir() logic is used and returns:

for a module object: the module's attributes. for a class object: its attributes, and recursively the attributes

of its bases.

for any other object: its attributes, its class's attributes, and recursively the attributes of its class's base classes.

`genometools.misc.`**`divmod`**(*x*, *y*) -> (*quotient*, *remainder*)
Return the tuple (x//y, x%y). Invariant: div*y + mod == x.

class `genometools.misc.`**`enumerate`**(*iterable*[, *start*]) → iterator for index, value of iterable
Return an enumerate object. iterable must be another object that supports iteration. The enumerate object yields pairs containing a count (from start, which defaults to zero) and a value yielded by the iterable argument. enumerate is useful for obtaining an indexed list:

(0, seq[0]), (1, seq[1]), (2, seq[2]), ...

`next`

`genometools.misc.`**`eval`**(*source*[, *globals*[, *locals*]]) → value
Evaluate the source in the context of globals and locals. The source may be a string representing a Python expression or a code object as returned by compile(). The globals must be a dictionary and locals can be any mapping, defaulting to the current globals and locals. If only globals is given, locals defaults to it.

`genometools.misc.`**`execfile`**(*filename*[, *globals*[, *locals*]])
Read and execute a Python script from a file. The globals and locals are dictionaries, defaulting to the current globals and locals. If only globals is given, locals defaults to it.

class `genometools.misc.`**`file`**(*name*[, *mode*[, *buffering*]]) → file object
Open a file. The mode can be 'r', 'w' or 'a' for reading (default), writing or appending. The file will be created if it doesn't exist when opened for writing or appending; it will be truncated when opened for writing. Add a 'b' to the mode for binary files. Add a '+' to the mode to allow simultaneous reading and writing. If the buffering argument is given, 0 means unbuffered, 1 means line buffered, and larger numbers specify the buffer size. The preferred way to open a file is with the builtin open() function. Add a 'U' to mode to open the file for input with universal newline support. Any line ending in the input file will be seen as a 'n' in Python. Also, a file so opened gains the attribute 'newlines'; the value for this attribute is one of None (no newline read yet), 'r', 'n', 'rn' or a tuple containing all the newline types seen.

'U' cannot be combined with 'w' or '+' mode.

close() → None or (perhaps) an integer. Close the file.
Sets data attribute .closed to True. A closed file cannot be used for further I/O operations. close() may be called more than once without error. Some kinds of file objects (for example, opened by popen()) may return an exit status upon closing.

closed
True if the file is closed

encoding
file encoding

errors
Unicode error handler

fileno() → integer "file descriptor".
This is needed for lower-level file interfaces, such os.read().

flush() → None. Flush the internal I/O buffer.

isatty() → true or false. True if the file is connected to a tty device.

mode
file mode ('r', 'U', 'w', 'a', possibly with 'b' or '+' added)

name
file name

newlines
end-of-line convention used in this file

next

read([*size*]) → read at most size bytes, returned as a string.
If the size argument is negative or omitted, read until EOF is reached. Notice that when in non-blocking mode, less data than what was requested may be returned, even if no size parameter was given.

readinto() → Undocumented. Don't use this; it may go away.

readline([*size*]) → next line from the file, as a string.
Retain newline. A non-negative size argument limits the maximum number of bytes to return (an incomplete line may be returned then). Return an empty string at EOF.

readlines([*size*]) → list of strings, each a line from the file.
Call readline() repeatedly and return a list of the lines so read. The optional size argument, if given, is an approximate bound on the total number of bytes in the lines returned.

seek(*offset*[, *whence*]) → None. Move to new file position.
Argument offset is a byte count. Optional argument whence defaults to 0 (offset from start of file, offset should be >= 0); other values are 1 (move relative to current position, positive or negative), and 2 (move relative to end of file, usually negative, although many platforms allow seeking beyond the end of a file). If the file is opened in text mode, only offsets returned by tell() are legal. Use of other offsets causes undefined behavior. Note that not all file objects are seekable.

softspace
flag indicating that a space needs to be printed; used by print

tell() → current file position, an integer (may be a long integer).

truncate([*size*]) → None. Truncate the file to at most size bytes.
Size defaults to the current file position, as returned by tell().

write(*str*) → None. Write string str to file.
Note that due to buffering, flush() or close() may be needed before the file on disk reflects the data written.

writelines(*sequence_of_strings*) → None. Write the strings to the file.
Note that newlines are not added. The sequence can be any iterable object producing strings. This is equivalent to calling write() for each string.

xreadlines() → returns self.
For backward compatibility. File objects now include the performance optimizations previously implemented in the xreadlines module.

genometools.misc.**filter**
alias of ifilter

genometools.misc.**flatten**(*l*)
Flattens a list of lists.

Parameters **l** (list) – The list of lists.

Returns The flattened list.

Return type *list*

class genometools.misc.**float**(*x*) → floating point number
Convert a string or number to a floating point number, if possible.

as_integer_ratio() -> (*int*, *int*)
Return a pair of integers, whose ratio is exactly equal to the original float and with a positive denominator. Raise OverflowError on infinities and a ValueError on NaNs.

```
>>> (10.0).as_integer_ratio()
(10, 1)
>>> (0.0).as_integer_ratio()
(0, 1)
>>> (-.25).as_integer_ratio()
(-1, 4)
```

conjugate()
Return self, the complex conjugate of any float.

fromhex(*string*) → float
Create a floating-point number from a hexadecimal string. >>> float.fromhex('0x1.ffffp10') 2047.984375 >>> float.fromhex('-0x1p-1074') -4.9406564584124654e-324

hex() → string
Return a hexadecimal representation of a floating-point number. >>> (-0.1).hex() '-0x1.999999999999ap-4' >>> 3.14159.hex() '0x1.921f9f01b866ep+1'

imag
the imaginary part of a complex number

is_integer()
Return True if the float is an integer.

real
the real part of a complex number

genometools.misc.**format**(*value*[, *format_spec*]) → string
Returns value.__format__(format_spec) format_spec defaults to ""

class genometools.misc.**frozenset** → empty frozenset object
frozenset(iterable) -> frozenset object

Build an immutable unordered collection of unique elements.

copy()
Return a shallow copy of a set.

difference()
Return the difference of two or more sets as a new set.

(i.e. all elements that are in this set but not the others.)

intersection()
Return the intersection of two or more sets as a new set.

(i.e. elements that are common to all of the sets.)

isdisjoint()
Return True if two sets have a null intersection.

issubset()
Report whether another set contains this set.

issuperset()
Report whether this set contains another set.

symmetric_difference()
Return the symmetric difference of two sets as a new set.

(i.e. all elements that are in exactly one of the sets.)

union()
Return the union of sets as a new set.

(i.e. all elements that are in either set.)

genometools.misc.**ftp_download**(*url*, *download_file*, *if_exists=u'error'*, *user_name=u'anonymous'*, *password=u''*, *blocksize=4194304*)
Downloads a file from an FTP server.

Parameters

- **url** (str) – The URL of the file to download.
- **download_file** (str) – The path of the local file to download to.
- **if_exists** (str, *optional*) –

  Desired behavior when the download file already exists. One of: 'error' - Raise an OSError 'skip' - Do nothing, only report a warning. 'overwrite' - Overwrite the file. reporting a warning.

  Default: 'error'.
- **user_name** (str, *optional*) – The user name to use for logging into the FTP server. ['anonymous']
- **password** (str, *optional*) – The password to use for logging into the FTP server. ['']
- **blocksize** (int, *optional*) – The blocksize (in bytes) to use for downloading. [4194304]

Returns

Return type None

genometools.misc.**get_file_checksum**(*path*)
Get the checksum of a file (using sum, Unix-only).

This function is only available on certain platforms.

Parameters **path** (str) – The path of the file.

Returns The checksum.

Return type *int*

Raises *IOError* – If the file does not exist.

genometools.misc.**get_file_md5sum**(*path*)
Calculate the MD5 hash for a file.

genometools.misc.**get_file_size**(*path*)
The the size of a file in bytes.

Parameters **path** (str) – The path of the file.

Returns The size of the file in bytes.

Return type *int*

Raises

- *IOError* – If the file does not exist.
- *OSError* – If a file system error occurs.

genometools.misc.**get_fize_size**(*path*)
The the size of a file.

Parameters **path** (str) – The file path.

Returns The size of the file in bytes.

Return type *int*

genometools.misc.**get_logger**(*name=u''*, *log_stream=None*, *log_file=None*, *quiet=False*, *verbose=False*)
Convenience function for getting a logger.

genometools.misc.**get_url_file_name**(*url*)
Get the file name from an url

Parameters **url** (str) –

Returns The file name

Return type *str*

genometools.misc.**get_url_size**(*url*)
Get the size of a URL.

Note: Uses requests, so it does not work for FTP URLs.

Source: StackOverflow user "Burhan Khalid". (http://stackoverflow.com/a/24585314/5651021)

Parameters **url** (str) – The URL.

Returns The size of the URL in bytes.

Return type *int*

genometools.misc.**getattr**(*object*, *name*[, *default*]) → value
Get a named attribute from an object; getattr(x, 'y') is equivalent to x.y. When a default argument is given, it is returned when the attribute doesn't exist; without it, an exception is raised in that case.

genometools.misc.**globals**() → dictionary
Return the dictionary containing the current scope's global variables.

genometools.misc.**gzip_open_text**(*path*, *encoding=None*)
Opens a plain-text file that may be gzip'ed.

Parameters

- **path** (str) – The file.
- **encoding** (str, *optional*) – The encoding to use.

Returns A file-like object.

Return type file-like

Notes

Generally, reading gzip'ed files with gzip.open is very slow, and it is preferable to pipe the file into the python script using `gunzip -c`. The script then reads the file from stdin.

genometools.misc.**hasattr**(*object*, *name*) → bool
Return whether the object has an attribute with the given name. (This is done by calling getattr(object, name) and catching exceptions.)

genometools.misc.**hash**(*object*) → integer
Return a hash value for the object. Two objects with the same value have the same hash value. The reverse is not necessarily true, but likely.

genometools.misc.**hex**(*number*) → string
Return the hexadecimal representation of an integer or long integer.

genometools.misc.**http_download**(*url*, *download_file*, *overwrite=False*, *raise_http_exception=True*)
Download a file over HTTP(S).

See: http://stackoverflow.com/a/13137873/5651021

Parameters

- **url** (str) – The URL.
- **download_file** (str) – The path of the local file to write to.
- **overwrite** (bool, *optional*) – Whether to overwrite an existing file (if present). [False]
- **raise_http_exception** (bool, *optional*) – Whether to raise an exception if there is an HTTP error. [True]

Raises

- *OSError* – If the file already exists and overwrite is set to False.
- `requests.HTTPError` – If an HTTP error occurred and `raise_http_exception` was set to `True`.

genometools.misc.**id**(*object*) → integer
Return the identity of an object. This is guaranteed to be unique among simultaneously existing objects. (Hint: it's the object's memory address.)

genometools.misc.**input**()
raw_input([prompt]) -> string

Read a string from standard input. The trailing newline is stripped. If the user hits EOF (Unix: Ctl-D, Windows: Ctl-Z+Return), raise EOFError. On Unix, GNU readline is used if enabled. The prompt string, if given, is printed without a trailing newline before reading.

genometools.misc.**int**
alias of newint

genometools.misc.**intern**(*string*) → string
"Intern" the given string. This enters the string in the (global) table of interned strings whose purpose is to speed up dictionary lookups. Return the string itself or the previously interned string object with the same value.

genometools.misc.**is_writable**(*path*)
Tests if a file is writable.

genometools.misc.**isinstance**(*object*, *class-or-type-or-tuple*) → bool
Return whether an object is an instance of a class or of a subclass thereof. With a type as second argument, return whether that is the object's type. The form using a tuple, isinstance(x, (A, B, ...)), is a shortcut for isinstance(x, A) or isinstance(x, B) or ... (etc.).

genometools.misc.**issubclass**(*C*, *B*) → bool
Return whether class C is a subclass (i.e., a derived class) of class B. When using a tuple as the second argument issubclass(X, (A, B, ...)), is a shortcut for issubclass(X, A) or issubclass(X, B) or ... (etc.).

genometools.misc.**iter**(*collection*) → iterator
iter(callable, sentinel) -> iterator

Get an iterator from an object. In the first form, the argument must supply its own iterator, or be a sequence. In the second form, the callable is called until it returns the sentinel.

genometools.misc.**len**(*object*) → integer
Return the number of items of a sequence or collection.

genometools.misc.**list**
alias of newlist

genometools.misc.**locals**() → dictionary
Update and return a dictionary containing the current scope's local variables.

class genometools.misc.**long**(*x=0*) → long
long(x, base=10) -> long

Convert a number or string to a long integer, or return 0L if no arguments are given. If x is floating point, the conversion truncates towards zero.

If x is not a number or if base is given, then x must be a string or Unicode object representing an integer literal in the given base. The literal can be preceded by '+' or '-' and be surrounded by whitespace. The base defaults to 10. Valid bases are 0 and 2-36. Base 0 means to interpret the base from the string as an integer literal. >>> int('0b100', base=0) 4L

bit_length() → int or long
Number of bits necessary to represent self in binary. >>> bin(37L) '0b100101' >>> (37L).bit_length() 6

conjugate()
Returns self, the complex conjugate of any long.

denominator
the denominator of a rational number in lowest terms

imag
the imaginary part of a complex number

numerator
the numerator of a rational number in lowest terms

real
the real part of a complex number

genometools.misc.**make_sure_dir_exists**(*dir_*, *create_subfolders=False*)
Ensures that a directory exists.

Adapted from StackOverflow users "Bengt" and "Heikki Toivonen" (http://stackoverflow.com/a/5032238).

Parameters

- **dir** (str) – The directory path.
- **create_subfolders** (bool, *optional*) – Whether to create any inexistent subfolders. [False]

Returns

Return type None

Raises *OSError* – If a file system error occurs.

genometools.misc.**map**
alias of imap

genometools.misc.**max**(*iterable*[, *key=func*]) → value
max(a, b, c, ...[, key=func]) -> value

With a single iterable argument, return its largest item. With two or more arguments, return the largest argument.

class genometools.misc.**memoryview**(*object*)
Create a new memoryview object which references the given object.

genometools.misc.**min**(*iterable*[, *key=func*]) → value
min(a, b, c, ...[, key=func]) -> value

With a single iterable argument, return its smallest item. With two or more arguments, return the smallest argument.

genometools.misc.**next**(*iterator*[, *default*])
Return the next item from the iterator. If default is given and the iterator is exhausted, it is returned instead of raising StopIteration.

genometools.misc.**object**
alias of newobject

genometools.misc.**oct**(*number*) → string
Return the octal representation of an integer or long integer.

genometools.misc.**open**()
Open file and return a stream. Raise IOError upon failure.

file is either a text or byte string giving the name (and the path if the file isn't in the current working directory) of the file to be opened or an integer file descriptor of the file to be wrapped. (If a file descriptor is given, it is closed when the returned I/O object is closed, unless closefd is set to False.)

mode is an optional string that specifies the mode in which the file is opened. It defaults to 'r' which means open for reading in text mode. Other common values are 'w' for writing (truncating the file if it already exists), and 'a' for appending (which on some Unix systems, means that all writes append to the end of the file regardless of the current seek position). In text mode, if encoding is not specified the encoding used is platform dependent. (For reading and writing raw bytes use binary mode and leave encoding unspecified.) The available modes are:

Character	Meaning
'r'	open for reading (default)
'w'	open for writing, truncating the file first
'a'	open for writing, appending to the end of the file if it exists
'b'	binary mode
't'	text mode (default)
'+'	open a disk file for updating (reading and writing)
'U'	universal newline mode (for backwards compatibility; unneeded for new code)

The default mode is 'rt' (open for reading text). For binary random access, the mode 'w+b' opens and truncates the file to 0 bytes, while 'r+b' opens the file without truncation.

Python distinguishes between files opened in binary and text modes, even when the underlying operating system doesn't. Files opened in binary mode (appending 'b' to the mode argument) return contents as bytes objects without any decoding. In text mode (the default, or when 't' is appended to the mode argument), the contents of the file are returned as strings, the bytes having been first decoded using a platform-dependent encoding or using the specified encoding if given.

buffering is an optional integer used to set the buffering policy. Pass 0 to switch buffering off (only allowed in binary mode), 1 to select line buffering (only usable in text mode), and an integer > 1 to indicate the size of a fixed-size chunk buffer. When no buffering argument is given, the default buffering policy works as follows:

- Binary files are buffered in fixed-size chunks; the size of the buffer is chosen using a heuristic trying to determine the underlying device's "block size" and falling back on `io.DEFAULT_BUFFER_SIZE`. On many systems, the buffer will typically be 4096 or 8192 bytes long.
- "Interactive" text files (files for which isatty() returns True) use line buffering. Other text files use the policy described above for binary files.

encoding is the name of the encoding used to decode or encode the file. This should only be used in text mode. The default encoding is platform dependent, but any encoding supported by Python can be passed. See the codecs module for the list of supported encodings.

errors is an optional string that specifies how encoding errors are to be handled—this argument should not be used in binary mode. Pass 'strict' to raise a ValueError exception if there is an encoding error (the default of None has the same effect), or pass 'ignore' to ignore errors. (Note that ignoring encoding errors can lead to data loss.) See the documentation for codecs.register for a list of the permitted encoding error strings.

newline controls how universal newlines works (it only applies to text mode). It can be None, '', 'n', 'r', and 'rn'. It works as follows:

- On input, if newline is None, universal newlines mode is enabled. Lines in the input can end in 'n', 'r', or 'rn', and these are translated into 'n' before being returned to the caller. If it is '', universal newline mode is enabled, but line endings are returned to the caller untranslated. If it has any of the other legal values, input lines are only terminated by the given string, and the line ending is returned to the caller untranslated.
- On output, if newline is None, any 'n' characters written are translated to the system default line separator, os.linesep. If newline is '', no translation takes place. If newline is any of the other legal values, any 'n' characters written are translated to the given string.

If closefd is False, the underlying file descriptor will be kept open when the file is closed. This does not work when a file name is given and must be True in that case.

open() returns a file object whose type depends on the mode, and through which the standard file operations such as reading and writing are performed. When open() is used to open a file in a text mode ('w', 'r', 'wt', 'rt', etc.), it returns a TextIOWrapper. When used to open a file in a binary mode, the returned class varies: in read binary mode, it returns a BufferedReader; in write binary and append binary modes, it returns a BufferedWriter, and in read/write mode, it returns a BufferedRandom.

It is also possible to use a string or bytearray as a file for both reading and writing. For strings StringIO can be used like a file opened in a text mode, and for bytes a BytesIO can be used like a file opened in a binary mode.

genometools.misc.**ord**(*c*) → integer
Return the integer ordinal of a one-character string.

genometools.misc.**pow**(*x*, *y*[, *z*]) → number
With two arguments, equivalent to x**y. With three arguments, equivalent to (x**y) % z, but may be more efficient (e.g. for ints).

genometools.misc.**print**(*value*, ..., *sep=' '*, *end='n'*, *file=sys.stdout*)
Prints the values to a stream, or to sys.stdout by default. Optional keyword arguments: file: a file-like object (stream); defaults to the current sys.stdout. sep: string inserted between values, default a space. end: string appended after the last value, default a newline.

class genometools.misc.**property**(*fget=None*, *fset=None*, *fdel=None*, *doc=None*) → property attribute
fget is a function to be used for getting an attribute value, and likewise fset is a function for setting, and fdel a function for del'ing, an attribute. Typical use is to define a managed attribute x:

class C(object): def getx(self): return self._x def setx(self, value): self._x = value def delx(self): del self._x x = property(getx, setx, delx, "I'm the 'x' property.")

Decorators make defining new properties or modifying existing ones easy:

class C(object): @property def x(self):

"I am the 'x' property." return self._x

@x.setter def x(self, value):

self._x = value

@x.deleter def x(self):

del self._x

deleter()
Descriptor to change the deleter on a property.

getter()
Descriptor to change the getter on a property.

setter()
Descriptor to change the setter on a property.

genometools.misc.**range**
alias of newrange

genometools.misc.**raw_input**([*prompt*]) → string
Read a string from standard input. The trailing newline is stripped. If the user hits EOF (Unix: Ctl-D, Windows: Ctl-Z+Return), raise EOFError. On Unix, GNU readline is used if enabled. The prompt string, if given, is printed without a trailing newline before reading.

genometools.misc.**read_all**(*path*, *encoding=u'UTF-8'*)
Reads a tab-delimited text file.

The file can either be uncompressed or gzip'ed.

Parameters

- **path** (str) – The path of the file.
- **enc** (str, *optional*) – The file encoding.

Returns A list, which each element containing the contents of a row (as a tuple).

Return type List of (tuple of str)

genometools.misc.**read_single**(*path*, *encoding=u'UTF-8'*)
Reads the first column of a tab-delimited text file.

The file can either be uncompressed or gzip'ed.

Parameters

- **path** (str) – The path of the file.
- **enc** (str) – The file encoding.

Returns A list containing the elements in the first column.

Return type List of str

genometools.misc.**reduce**(*function*, *sequence*[, *initial*]) → value
Apply a function of two arguments cumulatively to the items of a sequence, from left to right, so as to reduce the sequence to a single value. For example, reduce(lambda x, y: x+y, [1, 2, 3, 4, 5]) calculates ((((1+2)+3)+4)+5). If initial is present, it is placed before the items of the sequence in the calculation, and serves as a default when the sequence is empty.

genometools.misc.**reload**(*module*) → module
Reload the module. The module must have been successfully imported before.

genometools.misc.**repr**(*object*) → string
Return the canonical string representation of the object. For most object types, eval(repr(object)) == object.

class genometools.misc.**reversed**(*sequence*) → reverse iterator over values of the sequence
Return a reverse iterator

next

genometools.misc.**round**(*number*, *ndigits=None*)
See Python 3 documentation: uses Banker's Rounding.

Delegates to the __round__ method if for some reason this exists.

If not, rounds a number to a given precision in decimal digits (default 0 digits). This returns an int when called with one argument, otherwise the same type as the number. ndigits may be negative.

See the test_round method in future/tests/test_builtins.py for examples.

class genometools.misc.**set** → new empty set object
set(iterable) -> new set object

Build an unordered collection of unique elements.

add()
Add an element to a set.

This has no effect if the element is already present.

clear()
Remove all elements from this set.

copy()
Return a shallow copy of a set.

difference()
Return the difference of two or more sets as a new set.

(i.e. all elements that are in this set but not the others.)

difference_update()
Remove all elements of another set from this set.

discard()
Remove an element from a set if it is a member.

If the element is not a member, do nothing.

intersection()
Return the intersection of two or more sets as a new set.

(i.e. elements that are common to all of the sets.)

intersection_update()
Update a set with the intersection of itself and another.

isdisjoint()
Return True if two sets have a null intersection.

issubset()
Report whether another set contains this set.

issuperset()
Report whether this set contains another set.

pop()
Remove and return an arbitrary set element. Raises KeyError if the set is empty.

remove()
Remove an element from a set; it must be a member.

If the element is not a member, raise a KeyError.

symmetric_difference()
Return the symmetric difference of two sets as a new set.

(i.e. all elements that are in exactly one of the sets.)

symmetric_difference_update()
Update a set with the symmetric difference of itself and another.

union()
Return the union of sets as a new set.

(i.e. all elements that are in either set.)

update()
Update a set with the union of itself and others.

genometools.misc.**setattr**(*object*, *name*, *value*)
Set a named attribute on an object; setattr(x, 'y', v) is equivalent to ''x.y = v''.

class genometools.misc.**slice**(*stop*)
slice(start, stop[, step])

Create a slice object. This is used for extended slicing (e.g. a[0:10:2]).

indices(*len) -> (start*, *stop*, *stride*)
Assuming a sequence of length len, calculate the start and stop indices, and the stride length of the extended slice described by S. Out of bounds indices are clipped in a manner consistent with the handling of normal slices.

genometools.misc.**smart_open_read**(**args*, ***kwds*)
Open a file for reading or return `stdin`.

Adapted from StackOverflow user "Wolph" (http://stackoverflow.com/a/17603000).

genometools.misc.**smart_open_write**(**args*, ***kwds*)
Open a file for writing or return `stdout`.

Adapted from StackOverflow user "Wolph" (http://stackoverflow.com/a/17603000).

genometools.misc.**sorted**()
sorted(iterable, cmp=None, key=None, reverse=False) –> new sorted list

class genometools.misc.**staticmethod**(*function*) → method
Convert a function to be a static method.

A static method does not receive an implicit first argument. To declare a static method, use this idiom:

class C: @staticmethod def f(arg1, arg2, ...):

...

It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class.

Static methods in Python are similar to those found in Java or C++. For a more advanced concept, see the classmethod builtin.

genometools.misc.**str**
alias of `newstr`

genometools.misc.**sum**(*sequence*[, *start*])→ value
Return the sum of a sequence of numbers (NOT strings) plus the value of parameter 'start' (which defaults to 0). When the sequence is empty, return start.

genometools.misc.**super**(*typ=<object object>*, *type_or_obj=<object object>*, *framedepth=1*)
Like builtin super(), but capable of magic.

This acts just like the builtin super() function, but if called without any arguments it attempts to infer them at runtime.

genometools.misc.**test_dir_writable**(*path*)
Test if we can write to a directory.

Parameters **dir** (`str`) – The directory path.

Returns Whether the directory is writable or not.

Return type *bool*

genometools.misc.**test_file_checksum**(*path*, *checksum*)
Test if a file has a given checksum (using `sum`, Unix-only).

Parameters

- **path** (`str`) – The path of the file.
- **checksum** (`int`) – The checksum to compare.

Returns Whether or not the file has the given checksum.

Return type *bool*

Raises *IOError* – If the file does not exist.

genometools.misc.**test_file_writable**(*path*)
Test if we can write to a file.

Parameters **path** (`str`) – The file path.

Returns Whether the file is writable or not.

Return type *bool*

class genometools.misc.**tuple** → empty tuple
tuple(iterable) -> tuple initialized from iterable's items

If the argument is a tuple, the return value is the same object.

count(*value*) → integer – return number of occurrences of value

index(*value*[, *start*[, *stop*]]) → integer – return first index of value.
Raises ValueError if the value is not present.

genometools.misc.**unichr**(*i*) → Unicode character
Return a Unicode string of one character with ordinal i; 0 <= i <= 0x10ffff.

class genometools.misc.**unicode**(*object*='') → unicode object
unicode(string[, encoding[, errors]]) -> unicode object

Create a new Unicode object from the given encoded string. encoding defaults to the current default string encoding. errors can be 'strict', 'replace' or 'ignore' and defaults to 'strict'.

capitalize() → unicode
Return a capitalized version of S, i.e. make the first character have upper case and the rest lower case.

center(*width*[, *fillchar*]) → unicode
Return S centered in a Unicode string of length width. Padding is done using the specified fill character (default is a space)

count(*sub*[, *start*[, *end*]]) → int
Return the number of non-overlapping occurrences of substring sub in Unicode string S[start:end]. Optional arguments start and end are interpreted as in slice notation.

decode([*encoding*[, *errors*]]) → string or unicode
Decodes S using the codec registered for encoding. encoding defaults to the default encoding. errors may be given to set a different error handling scheme. Default is 'strict' meaning that encoding errors raise a UnicodeDecodeError. Other possible values are 'ignore' and 'replace' as well as any other name registered with codecs.register_error that is able to handle UnicodeDecodeErrors.

encode([*encoding*[, *errors*]]) → string or unicode
Encodes S using the codec registered for encoding. encoding defaults to the default encoding. errors may be given to set a different error handling scheme. Default is 'strict' meaning that encoding errors raise a UnicodeEncodeError. Other possible values are 'ignore', 'replace' and 'xmlcharrefreplace' as well as any other name registered with codecs.register_error that can handle UnicodeEncodeErrors.

endswith(*suffix*[, *start*[, *end*]]) → bool
Return True if S ends with the specified suffix, False otherwise. With optional start, test S beginning at that position. With optional end, stop comparing S at that position. suffix can also be a tuple of strings to try.

expandtabs([*tabsize*]) → unicode
Return a copy of S where all tab characters are expanded using spaces. If tabsize is not given, a tab size of 8 characters is assumed.

find(*sub*[, *start*[, *end*]]) → int
Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end]. Optional arguments start and end are interpreted as in slice notation.

Return -1 on failure.

format(**args*, ***kwargs*) → unicode
Return a formatted version of S, using substitutions from args and kwargs. The substitutions are identified by braces ('{' and '}').

index(*sub*[, *start*[, *end*]]) → int
Like S.find() but raise ValueError when the substring is not found.

isalnum() → bool
Return True if all characters in S are alphanumeric and there is at least one character in S, False otherwise.

isalpha() → bool
Return True if all characters in S are alphabetic and there is at least one character in S, False otherwise.

isdecimal() → bool
Return True if there are only decimal characters in S, False otherwise.

isdigit() → bool
Return True if all characters in S are digits and there is at least one character in S, False otherwise.

islower() → bool
Return True if all cased characters in S are lowercase and there is at least one cased character in S, False otherwise.

isnumeric() → bool
Return True if there are only numeric characters in S, False otherwise.

isspace() → bool
Return True if all characters in S are whitespace and there is at least one character in S, False otherwise.

istitle() → bool
Return True if S is a titlecased string and there is at least one character in S, i.e. upper- and titlecase characters may only follow uncased characters and lowercase characters only cased ones. Return False otherwise.

isupper() → bool
Return True if all cased characters in S are uppercase and there is at least one cased character in S, False otherwise.

join(*iterable*) → unicode
Return a string which is the concatenation of the strings in the iterable. The separator between elements is S.

ljust(*width*[, *fillchar*]) → int
Return S left-justified in a Unicode string of length width. Padding is done using the specified fill character (default is a space).

lower() → unicode
Return a copy of the string S converted to lowercase.

lstrip([*chars*]) → unicode
Return a copy of the string S with leading whitespace removed. If chars is given and not None, remove characters in chars instead. If chars is a str, it will be converted to unicode before stripping

partition(*sep) -> (head*, *sep*, *tail*)
Search for the separator sep in S, and return the part before it, the separator itself, and the part after it. If the separator is not found, return S and two empty strings.

replace(*old*, *new*[, *count*]) → unicode
Return a copy of S with all occurrences of substring old replaced by new. If the optional argument count is given, only the first count occurrences are replaced.

rfind(*sub*[, *start*[, *end*]]) → int
Return the highest index in S where substring sub is found, such that sub is contained within S[start:end]. Optional arguments start and end are interpreted as in slice notation.

Return -1 on failure.

rindex(*sub*[, *start*[, *end*]]) → int
Like S.rfind() but raise ValueError when the substring is not found.

rjust(*width*[, *fillchar*]) → unicode
Return S right-justified in a Unicode string of length width. Padding is done using the specified fill character (default is a space).

rpartition(*sep) -> (head*, *sep*, *tail*)
Search for the separator sep in S, starting at the end of S, and return the part before it, the separator itself, and the part after it. If the separator is not found, return two empty strings and S.

rsplit([*sep*[, *maxsplit*]]) → list of strings
Return a list of the words in S, using sep as the delimiter string, starting at the end of the string and working to the front. If maxsplit is given, at most maxsplit splits are done. If sep is not specified, any whitespace string is a separator.

rstrip([*chars*]) → unicode
Return a copy of the string S with trailing whitespace removed. If chars is given and not None, remove characters in chars instead. If chars is a str, it will be converted to unicode before stripping

split([*sep*[, *maxsplit*]]) → list of strings
Return a list of the words in S, using sep as the delimiter string. If maxsplit is given, at most maxsplit splits are done. If sep is not specified or is None, any whitespace string is a separator and empty strings are removed from the result.

splitlines(*keepends=False*) → list of strings
Return a list of the lines in S, breaking at line boundaries. Line breaks are not included in the resulting list unless keepends is given and true.

startswith(*prefix*[, *start*[, *end*]]) → bool
Return True if S starts with the specified prefix, False otherwise. With optional start, test S beginning at that position. With optional end, stop comparing S at that position. prefix can also be a tuple of strings to try.

strip([*chars*]) → unicode
Return a copy of the string S with leading and trailing whitespace removed. If chars is given and not None, remove characters in chars instead. If chars is a str, it will be converted to unicode before stripping

swapcase() → unicode
Return a copy of S with uppercase characters converted to lowercase and vice versa.

title() → unicode
Return a titlecased version of S, i.e. words start with title case characters, all remaining cased characters have lower case.

translate(*table*) → unicode
Return a copy of the string S, where all characters have been mapped through the given translation table, which must be a mapping of Unicode ordinals to Unicode ordinals, Unicode strings or None. Unmapped characters are left untouched. Characters mapped to None are deleted.

upper() → unicode
Return a copy of S converted to uppercase.

zfill(*width*) → unicode
Pad a numeric string S with zeros on the left, to fill a field of the specified width. The string S is never

truncated.

genometools.misc.**vars**([*object*]) → dictionary
Without arguments, equivalent to locals(). With an argument, equivalent to object.__dict__.

class genometools.misc.**xrange**(*stop*) → xrange object
xrange(start, stop[, step]) -> xrange object

Like range(), but instead of returning a list, returns an object that generates the numbers in the range on demand. For looping, this is slightly faster than range() and more memory efficient.

genometools.misc.**zip**
alias of izip

License

GenomeTools Documentation

GenomeTools

CHAPTER 2

Indices and Tables

- genindex
- modindex

Python Module Index

g

Index

Symbols

A

B

C

D

E

F

G

H

I

J

S

T

U

V

W

X

Z