synbiopython Documentation

Release 0.0.1

Global Biofoundries Alliance

Mar 16, 2021

CONTENTS:

CHAPTER

ONE

INTRODUCTION

`SynBioPython` is an open-source software library which provides generic tools for Synthetic Biology community. This library is developed collectively by members of Global Biofoundries Alliance.

1.1 Objectives

- collation and development of synthetic biology-oriented codes and tools in Python
- catering to both beginner and advanced developers of synthetic biology software
- prevention of repeated efforts

1.2 Specific Aims

- standardization of read/write operations and other procedures and automation related tools to allow ease of access and interaction
- simplication of parsing of different synthetic biology related file formats
- development of APIs and wrapper functions on top of more complex codes to make them more intuitive to use

CHAPTER TWO

SYNBIOPYTHON PACKAGE

SynBioPython provides Python tools for Synthetic Biology.

2.1 Modules

2.1.1 synbiopython.codon module

Synbiopython (c) Global BioFoundry Alliance 2020

Synbiopython is licensed under the MIT License.

To view a copy of this license, visit <http://opensource.org/licenses/MIT/>.

@author: neilswainston

synbiopython.codon.table

Synbiopython (c) Global BioFoundry Alliance 2020

Synbiopython is licensed under the MIT License.

To view a copy of this license, visit <http://opensource.org/licenses/MIT/>.

@author: neilswainston

`synbiopython.codon.table.`**`get_table`**(*table_id*, *dna=True*)

Gets a codon table from from supplied parameter, which may be either an organism name or a NCBI Taxonomy id.

Parameters

- **`table_id`** (*`str`*) – an organism name or a NCBI Taxonomy id (as either a str or int).
- **`dna`** (*`bool`*) – boolean parameter specifying whether the codon table returned should contain DNA or RNA codons (default is DNA).

Returns a codon usage table.

Return type dict

synbiopython.codon.taxonomy_utils

@author: neilswainston

`synbiopython.codon.taxonomy_utils.`**`get_organism_name`**(*table_id*)

Gets an organism name from supplied parameter, which may be either an organism name or a NCBI Taxonomy id.

Parameters **`table_id`** (`str`) – an organism name or a NCBI Taxonomy id (as either a str or int).

Returns an organism name

Return type str

`synbiopython.codon.taxonomy_utils.`**`get_tax_id`**(*table_id*)

Gets a NCBI Taxonomy id from supplied parameter, which may be either an organism name or a NCBI Taxonomy id.

Parameters **`table_id`** (`str`) – an organism name or a NCBI Taxonomy id (as either a str or int).

Returns a NCBI Taxonomy id

Return type str

synbiopython.codon.utils

@author: neilswainston

`synbiopython.codon.utils.`**`optimise`**(*table*, *aa_seq*)

Codon optimise an amino acid sequence.

Parameters

- **`table`** (`str`) – a codon usage table.
- **`aa_seq`** – an amino acid sequence.

Returns a codon-optimised nucleic acid sequence, encoding the supplied amino acid sequence

Return type str

`synbiopython.codon.utils.`**`sample`**(*table*, *amino_acid*)

Sample a codon for a given amino acid probabilistically, based on its codon usage frequency.

Parameters

- **`table`** (`dict`) – a codon usage table.
- **`amino_acid`** (`str`) – a single-character string representing an amino acid.

Returns a codon encoding the supplied amino acid, sampled probabilistically.

Return type str

2.1.2 synbiopython.genbabel module

synbiopython.genbabel.gensbolconv

Synbiopython (c) Global BioFoundry Alliance 2020

Synbiopython is licensed under the MIT License.

This module provides code to work with SBOL validator https://validator.sbolstandard.org/

Use sample sbol file from github (can also be obtained from iBioSim)

Reference: https://github.com/SynBioDex/SBOL-Validator/blob/master/src/test/sequence1.xml http://synbiodex.github.io/SBOL-Validator/#query-parameters http://synbiodex.github.io/SBOL-Validator/#options http://biopython.org/DIST/docs/tutorial/Tutorial.html (Chapter 17 Graphics)

install: pip install biopython reportlab

The URI prefix is required for FASTA and GenBank conversion, and optional for SBOL 1 conversion

class synbiopython.genbabel.gensbolconv.GenSBOLconv.**GenSBOLconv**

Bases: `object`

Class to convert standard files (SBOL1, SBOL2, GenBank, Fasta, GFF3).

static access_sbolvalidator(*input_file*, *Output*, *uri_Prefix=''*)

Code to invoke the SBOL Validator server over the internet.

Parameters

- **input_file** (*str*) – input filename or filepath
- **Output** (*str, ('GenBank', 'FASTA', 'GFF3', 'SBOL1', 'SBOL2')*) – the type of Output file
- **uri_Prefix** (*str, optional*) – '' as default, URI Prefix is required for FASTA and GenBank input conversion

Returns POST request response from webpage

Return type object

export_outputfile(*input_filename*, *Response*, *Output*, *outputfile=None*)

Export the converted output file.

Parameters

- **input_filename** (*str*) – input filename or filepath
- **Response** (*object*) – response from POST request to sbolvalidator web page
- **Output** (*str, ('GenBank', 'FASTA', 'GFF3', 'SBOL1', 'SBOL2')*) – the type of Output file
- **outputfile** (*str, optional*) – provide specific outputfilename or filepath

static export_plasmidmap(*gbfile*, *filename=None*)

Export Linear and Circular Plasmid Map for the imported GenBank file.

Parameters

- **gbfile** (*str*) – a genbank file in .gb format or the path the file if not in the same folder.
- **filename** (*tuple, optional*) – the filenames/path to the filenames for the linear and circular plasmids in tuple

Returns the version from the genbank file

Return type str

static get_outputfile_extension(*Filetype*)
Get the output file extension based on the requested output language.

Parameters **Filetype** – the type of Output file

Returns the specific file extension

Return type str

run_sbolvalidator(*Input_file*, *Output*, *uri_Prefix=''*, ***kwargs*)
Wrapper function for the SBOL Validator.

Parameters

- **Input_file** (*str, filename or filepath*) – input file or path to input file
- **Output** (*str, ('GenBank', 'FASTA', 'GFF3', 'SBOL1', 'SBOL2')*) – the type of Output file
- **uri_Prefix** (*str, optional*) – '' as default, URI Prefix is required for FASTA and GenBank input conversion

Returns the validity of the Response, and export output file.

Return type str, "valid: True" if the conversion is done properly

Keyword Arguments

- *outputfile*: specify outputfile

synbiopython.genbabel.sbmlgen

Synbiopython (c) Global BioFoundry Alliance 2020

Synbiopython is licensed under the MIT License.

This module is to create SBML file for ODE model using simplesbml package, which relies on libSBML.

Reference: https://github.com/sys-bio/simplesbml

class synbiopython.genbabel.sbmlgen.SBMLgen.**SBMLgen**
Bases: object

Class to generate SBML file for ODE model.

static export_sbml(*ODE*, *Variable*, *Init*, *ParamName*, *Param*, *ParamUnit*, ***kwargs*)
Function to generate the SBML xml file.

Parameters

- **ODE** (*list*) – The ODEs in the form of string stored in a list
- **Variable** (*list*) – The names of variable in a list of string
- **Init** (*list*) – Initial conditions for the variables in a list of values
- **ParamName** (*list*) – The names of the parameters stored in a list
- **Param** (*list*) – The parameters values
- **ParamUnit** (*list*) – The unit for the parameter according to available unit definition

Returns SBML in str

Return type str

This module is the simplesbml package from https://simplesbml.readthedocs.io/en/latest/ with minor modifications to include more unit definitions

Reference: https://github.com/sys-bio/simplesbml/blob/master/simplesbml/__init__.py

class `synbiopython.genbabel.sbmlgen.simplesbml.`**`sbmlModel`**(*time_units='second'*, *extent_units='mole'*, *sub_units='mole'*, *level=3*, *version=1*)

Bases: `object`

Class to generate sbml model file using libsbml method.

`addAssignmentRule`(*var*, *math*)
To assign a state variable with an expression.

Parameters

- **`var`** (*`str`*) – id of the state variable
- **`math`** (*`str`*) – expression in str

`addCompartment`(*vol=1*, *comp_id=''*)
Create compartment of volume litres to the model.

Parameters

- **`vol`** (*`float`*) – volume of compartment in L
- **`comp_id`** (*`str`*) – compartment id

`addEvent`(*trigger*, *assignments*, *persistent=True*, *initial_value=False*, *priority=0*, *delay=0*, *event_id=''*)
Add event supplied with when an event is triggered and what happens using assignments in a dictionary.

Parameters

- **`trigger`** (*`str`*) – define when an event is triggered (logical expression)
- **`assignments`** (*`dict`*) – keys are the variables to be modified and the values are the new values
- **`persistent`** (*`boolean`*) – determine if the event will still be executed if trigger turns from True to False
- **`initial_value`** (*`boolean`*) – value of trigger before t=0
- **`priority`** (*`float`*) – determine which event is executed, event with larger priority is executed
- **`delay`** (*`float`*) – time between when the event is triggered and the assignment is implemented
- **`event_id`** (*`str`*) – id of the event

`addInitialAssignment`(*symbol*, *math*)
Describe the initial value of the variable in terms of other variables or parameters.

Parameters

- **`symbol`** (*`str`*) – id of the variable
- **`math`** (*`str`*) – expression

`addParameter`(*param_id*, *val*, *units='per_second'*)
Add Parameter with value and unit.

Parameters

- **param_id** (*str*) – parameter id/name
- **val** (*float*) – value for the parameter
- **units** (*str*) – unit for the parameter

addRateRule (*var*, *math*, *rr_id=''*)
Describe the derivative of the state variable wrt time as an expression.

Parameters

- **var** (*str*) – id of the state variable
- **math** (*str*) – expression in str
- **rr_id** (*str, optional*) – id for the reaction rate

addReaction (*reactants*, *products*, *expression*, *local_params=None*, *rxn_id=''*)
Create reaction provided with reactants and products in lists

Parameters

- **reactants** (*list*) – list of species id for reactants
- **products** (*list*) – list of species id for products
- **expression** (*str*) – reaction rate expression
- **local_params** (*dict*) – keys are the param id and values are their respective values
- **rxn_id** (*str, optional*) – id for the reaction

addSpecies (*species_id*, *amt*, *comp='c1'*)
Create Species with the provided amount.

Parameters

- **species_id** (*str*) – id or name of the species.
- **amt** (*float*) – initial amount.
- **comp** (*str*) – compartment id

check (*value*, *message*)
Return value to string using libsbml.

getCompartment (*comp_id*)
Return compartment.

getDocument ()
Return document.

getEvent (*event_id*)
Return event.

getInitialAssignment (*var*)
Return initial assignment.

getListOfCompartments ()
Return list of compartments.

getListOfEvents ()
Return list of events.

getListOfInitialAssignments ()
Return list of initial assignments.

getListOfParameters()
Return list of parameters.

getListOfReactions()
Return list of reactions.

getListOfRules()
Return list of rules.

getListOfSpecies()
Return list of species.

getModel()
Return Model.

getParameter(*param_id*)
Return parameter.

getReaction(*rxn_id*)
Return Reaction.

getRule(*var*)
Return rule.

getSpecies(*species_id*)
Return Species.

setLevelAndVersion(*level*, *version*)
Set the level and version of the SBML.

Parameters

- **level** (*int*) – level of the sbml
- **version** (*int*) – version of the sbml

toSBML()
Return the model in SBML format as strings.

synbiopython.genbabel.sbmlgen.simplesbml.**writeCode**(*doc*)
Return string containing calls to functions that reproduce the model in SBML doc.

synbiopython.genbabel.sbmlgen.simplesbml.**writeCodeFromFile**(*filename*)
Read the SBML format model and returns strings containing calls to functions to reproduce the model in an sbmlModel object.

synbiopython.genbabel.sbmlgen.simplesbml.**writeCodeFromString**(*sbmlstring*)
Read sbmlstring as the SBML format model and return strings containing calls to functions to reproduce the model in an sbmlModel object.

synbiopython.genbabel.sedmlomexgen

Synbiopython (c) Global BioFoundry Alliance 2020

Synbiopython is licensed under the MIT License.

This module is to generate SEDML and COMBINE OMEX files.

class synbiopython.genbabel.sedmlomexgen.SEDMLOMEXgen.**SEDMLOMEXgen**
Bases: object

Class to generate the SEDML and COMBINE OMEX files.

static execute_inlineomex(*inline_omex*)
Execute the inline omex and generate the simulation figures.

export_omex(*antimony_str*, *phrasedml_str*, ***kwargs*)
Generate COMBINE OMEX file.

Parameters

- **antimony_str** (`str`) – represent the SBML
- **phrasedml_str** (`str`) – represent the SEDML

Returns omex inline

Return type str

Keyword Arguments

- *outputfile*: specify outputfile

static find_between(*s*, *first*, *last*)
Get the substring from string based on indexes.

Parameters

- **s** (`str`) – string to be searched
- **first** (`str`) – part of the string at the front
- **last** (`str`) – part of the string at the end

static get_omexfilename()
Return filename to the OMEX file according to the export time.

static get_sbml_biomodel(*Biomodels_ID*, ***kwargs*)
Get SBML model from biomodel. Use outputfile keyword argument to export the SBML model into .xml file at the specific path. A default .xml file will be generated by default at the temp directory.

Parameters **Biomodels_ID** (`str`) – the ID for the Biomodels

Returns the sbml in string format

Return type str

Keyword Arguments

- *outputfile*: specify outputfile

phrasedmltosedml(*phrasedml_str*, *sbml_file*, ***kwargs*)
Generate SEDML file from phrasedml. Example of phrasedml_str: phrasedml_str = '" model1 = model "{}" … … '"

Parameters

- **phrasedml_str** (`str`) – text-based way to represent SEDML
- **sbml_file** (`str`) – the SBML xml file/path to the file

Returns the sedml string

Return type str

static sbmltoantimony(*sbmlfile*)
Get the sbml file and return the antimony string.

synbiopython.genbabel.simplednaplot

Synbiopython (c) Global BioFoundry Alliance 2020

Synbiopython is licensed under the MIT License.

This module is to implement the simple plotting of the gene circuit using the modified code from quickplot.py in DNAplotlib library # added quickplot style for writing Regulations. # added new Regulation type: Derepression

Reference: https://github.com/VoigtLab/dnaplotlib

Install: pip install dnaplotlib

class `synbiopython.genbabel.simplednaplot.SimpleDNAplot.`**`SimpleDNAplot`**
Bases: `object`

Class to generate SBOL visual compliant gene circuit diagram. Regulation type: Connection, Activation, Repression, Derepression Each of the part will be numbered sequentially based on the part type/module from left to right starting from index 0. Example: p0-r0-c0-t0-p1-r1-c1-t1 for two modules with promoter, RBS, coding region, and terminator. Input = "p.pTet r.rbs34 c.orange.LacI t p.pLac r.rbs32 c.green.TetR t" Regulations = "c0->p1.Repression c1->p0.Repression" # The default color is black if color is not specified

static compute_dnalength(*part*, *part_length*)
Calculate the position for the to_part or from_part for plotting arrows automatically.

Parameters

- **part** (*str*) – the to_part or from_part
- **part_length** (*list of str*) – all the parts with sequential numbering starting from 0

Returns dna length

Return type float

plot_circuit(*Input*, *Regulation=None*, *savefig=None*)
Plot the SBOL-compliant gene circuit figure.

Parameters

- **Input** (*str*) – Input design from users
- **Regulation** (*str*) – Regulation strings from users
- **savefig** (*str, optional*) – path to store the output figure

Returns max dna design length and export the gene circuit figure

Return type float

set_circuit_design(*Input*, *Regulation=None*)
Generate the dictionary list containing circuit design information.

Parameters

- **Input** (*str*) – a string containing the individual type of part, followed by color and name separated by a space.
- **Regulation** (*str, optional*) – a string containing the from_part to the to_part connected by an arrow. Type of interaction is specified after the topart followed by the color. Default color of black is used is not specified.

Returns The part information and Regulations stored in the form of list of dictionaries.

Return type list of dict

synbiopython.genbabel.utilities

Miscellaneous functions for Genbabel package.

`synbiopython.genbabel.utilities.`**`getfilename`**`()`

Return the filename based on the datetime.

Returns the filename in year-month-day_hour-minute

Return type str

2.1.3 Synbiopython lab_automation module

synbiopython.lab_automation.containers

This module implements the Base class for all plates.

See synbiopython.lab_automation.containers for more specific plate subclasses, with set number of wells, well format, etc.

`exception` `synbiopython.lab_automation.containers.Plate.`**`NoUniqueWell`**

Bases: `Exception`

NoUniqueWell exception class.

`class` `synbiopython.lab_automation.containers.Plate.`**`Plate`**`(`*name=None*, *wells_data=None*, *plate_data=None*`)`

Bases: `object`

Base class for all plates.

See the builtin_containers for usage classes, such as generic microplate classes (Plate96, Plate384, etc).

Parameters

- **`name`** – Name or ID of the Plate as it will appear in strings and reports
- **`wells_data`** – A dict {"A1": {data}, "A2": ... }. The format of the data is left free
- **`plate_data`** – plate data

`find_unique_well_by_condition``(`*condition*`)`

Return the unique well of the plate satisfying the condition.

The `condition` method should have a signature of Well=>True/False.

Raises a NoUniqueWell error if 0 or several wells satisfy the condition.

`find_unique_well_containing``(`*query*`)`

Return the unique well whose content contains the query.

`get_well_at_index``(`*index*, *direction='row'*`)`

Return the well at the corresponding index.

Examples:

```
>>> plate.get_well_at_index(1)  # well A1
>>> plate.get_well_at_index(2)  # well A2
>>> plate.get_well_at_index(2, direction="column")  # well B1
```

index_to_wellname (*index*, *direction='row'*)
Return the name of the well at the corresponding index.

Examples:

```
>>> plate.index_to_wellname(1)  # "A1"
>>> plate.get_well_at_index(2)  # "A2"
>>> plate.get_well_at_index(2, direction="column")  # "B1"
```

iter_wells (*direction='row'*)
Iter through the wells either by row or by column.

Examples:

```
>>> for well in plate.iter_wells():
>>>     print (well.name)
```

list_filtered_wells (*well_filter*)
List filtered wells.

Examples:

```
>>> def condition(well):
>>>     return well.volume > 50
>>> for well in myplate.list_filtered_wells(condition):
>>>     print(well.name)
```

list_well_data_fields ()
Return all fields used in well data in the plate.

list_wells_in_column (*column_number*)
Return the list of all wells of the plate in the given column.

Examples:

```
>>> for well in plate.list_wells_in_column(5):
>>>      print(well.name)
```

list_wells_in_row (*row*)
Return the list of all wells of the plate in the given row.

The *row* can be either a row number (1,2,3) or row letter(s) (A,B,C).

Examples:

```
>>> for well in plate.list_wells_in_row("H"):
>>>      print(well.name)
```

return_column (*column_number*)
Return the list of all wells of the plate in the given column.

return_row (*row*)
Return the list of all wells of the plate in the given row.

The *row* can be either a row number (1,2,3) or row letter(s) (A,B,C).

to_dict (*replace_nans_by='null'*)
Convert plate to dict.

to_pandas_dataframe (*fields=None*, *direction='row'*)
Return a dataframe with the info on each well.

`well_class`
alias of `synbiopython.lab_automation.containers.Well.Well`

`wellname_to_index`(*wellname*, *direction='row'*)
Return the index of the well in the plate.

Examples: >>> plate.wellname_to_index("A1") # 1 >>> plate.wellname_to_index("A2") # 2 >>> plate.wellname_to_index("A1", direction="column") # 9 (8x12 plate)

`wells_grouped_by`(*data_field=None*, *key=None*, *sort_keys=False*, *ignore_none=False*, *direction_of_occurence='row'*)
Return wells grouped by key.

`wells_sorted_by`(*sortkey*)
Return wells sorted by sortkey

This module contains a generic class for a well.

`class` `synbiopython.lab_automation.containers.Well.`**`Well`**(*plate*, *row*, *column*, *name*, *data=None*)
Bases: `object`

Generic class for a well.

Parameters

- **`plate`** – The plate on which the well is located
- **`row`** – The well's row (a number, starting from 0)
- **`column`** – The well's column (a number, starting from 0)
- **`name`** – The well's name, for instance "A1"
- **`data`** – A dictionary storing data on the well, used in algorithms and reports.

`add_content`(*components_quantities*, *volume=None*, *unit_volume='L'*)
Add content to well.

Parameters

- **`components_quantities`** – Dictionary of components and quantities (default: gram). Example *{"Compound_1": 5}*.
- **`volume`** – Volume (default: liter).
- **`unit_volume`** – Unit of volume (default: liter). Options: liter (L), milliliter (mL), microliter (uL), nanoliter (nL).

`capacity = None`

`property coordinates`
Return (well.row, well.column).

`dead_volume_per_transfer_class = None`

`empty_completely`()
Empty the well.

`index_in_plate`(*direction='row'*)
Return the index of the well in the plate.

`is_after`(*other*, *direction='row'*)
Return whether this well is located strictly after the other well.

Example: iterate over all free wells after the last non-free well:

```
>>> direction = 'row'
>>> last_occupied_well = plate.last_nonempty_well(direction=direction)
>>> free_wells = (w for w in plate.iter_wells(direction=direction)
>>>                 if w.is_after(last_occupied_well))
>>> for well in free_wells: ...
```

property is_empty
Return true if the well's volume is 0.

iterate_sources_tree()
Iterate through the tree of sources.

pretty_summary()
Return a summary string of the well.

subtract_content(*components_quantities*, *volume=0*)
Subtract content from well.

to_dict()
Convert well to dict

property volume
Return volume.

This module contains a class to represent the volume and quantities of a well.

class synbiopython.lab_automation.containers.WellContent.**WellContent**(*quantities=None*, *volume=0*)

Bases: `object`

Class to represent the volume and quantities of a well.

Having the well content represented as a separate object makes it possible to have several wells share the same content, e.g. in throughs.

components_as_string(*separator=' '*)
Return a string representation of what's in the well mix.

concentration(*component=None*, *default=0*)
Return concentration of component.

make_empty()
Empty the well.

to_dict()
Return a dict {volume: 0.0001, quantities: {. . . :. . . }}.

synbiopython.lab_automation.picklist

Classes to represent picklists and liquid transfers in general.

class synbiopython.lab_automation.picklist.PickList.**PickList**(*transfers_list=()*, *data=None*)

Bases: `object`

Representation of a list of well-to-well transfers.

Parameters

- **transfers_list** – A list of Transfer objects that will be part of the same dispensing operation, in the order in which they are meant to be simulated.

- **data** – A dict with information on the picklist.

add_transfer(*source_well=None*, *destination_well=None*, *volume=None*, *data=None*, *transfer=None*)
Add a transfer to the picklist's tranfers list.

You can either provide a `Transfer` object with the `transfer` parameter, or the parameters.

enforce_maximum_dispense_volume(*max_dispense_volume*)
Return a new picklist were every too-large dispense is broken down into smaller dispenses.

static merge_picklists(*picklists_list*)
Merge the list of picklists into a single picklist.

The transfers in the final picklist are the concatenation of the transfers in the different picklists, in the order in which they appear in the list.

restricted_to(*transfer_filter=None*, *source_well=None*, *destination_well=None*)
Return a version of the picklist restricted to transfers with the right source/destination well.

You can provide `source_well` and `destination_well` or alternatively just a function `transfer_filter` with signature (transfer)=>True/False that will be used to filter out transfers (for which it returns false).

simulate(*content_field='content'*, *inplace=True*)
Simulate the execution of the picklist.

sorted_by(*sorting_method='source_well'*)
Return a new version of the picklist sorted by some parameter.

The `sorting_method` is either the name of an attribute of the transfers, such as "source_well", or a function f(transfer) -> value.

to_plain_string()
Return the list of transfers in human-readable format.

to_plain_textfile(*filename*)
Write the picklist in a file in a human reable format.

total_transferred_volume()
Return the sum of all volumes from all transfers.

class `synbiopython.lab_automation.picklist.Transfer.`**Transfer**(*source_well*, *destination_well*, *volume*, *data=None*)
Bases: `object`

Class representing a transfer from a source well to a destination well.

Parameters

- **source_well** – A Well object from which to transfer.
- **destination_well** – A Well object to which to transfer.
- **volume** – Volume to be transferred, expressed in liters.
- **data** – A dict containing any useful information about the transfer. This information can be used later e.g. as parameters for the transfer when exporting a picklist.

apply()

to_plain_string()
Return "Transfer {volume}L from {source_well} into {dest_well}".

to_short_string()
Return "Transfer {volume}L {source_well} -> {dest_well}".

with_new_volume(*new_volume*)
Return a version of the transfer with a new volume.

exception synbiopython.lab_automation.picklist.Transfer.**TransferError**
Bases: `ValueError`

synbiopython.lab_automation.tools

Miscellaneous useful functions.

In particular, methods for converting to and from plate coordinates.

synbiopython.lab_automation.tools.**find_best_volume_unit**(*vols*)
Find the best volume unit for a list of volumes.

synbiopython.lab_automation.tools.**human_seq_size**(*n*)
Return the given sequence as a human friendly 35b, 1.4k, 15k, etc.

synbiopython.lab_automation.tools.**human_volume**(*vol*, *unit='auto'*)
Return a human-readable volume.

synbiopython.lab_automation.tools.**replace_nans_in_dict**(*dictionary*, *replace_by='null'*)
Replace NaNs in a dictionary with a string.

Parameters

- **dictionary** (*dict*) – the dictionary
- **replace_by** (*str*) – replacement

synbiopython.lab_automation.tools.**round_at**(*value*, *rounding=None*)
Round value at the nearest rounding.

Parameters **value** – the value to round

CHAPTER THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX