Documentation

Release 1.0.0

Sebastien Ravel (CIRAD)

Dec 03, 2020

SCRIPTS DOCUMENTATION

scripts

Table of contents

cli.py

Process some sdfsf.

```
usage: cli.py [-h] -i IDENTITY --sum N [N ...]
```

positional arguments

n
: An integer for the accumulator.

optional arguments

-h, --help
: show this help message and exit

-i <identity>, **--identity** <identity>
: the default result for no arguments (default: 0)

--sum
: Sum the integers (default: find the max).

filter_mummer.py

More information: Script version: 0.0.1

```
usage: filter_mummer.py -l path/to/file/csv [-v] [-h] [-d] [-p] [-c int]
                        [-f int]
```

Documentation avail at: https://yoda-powers.readthedocs.io/en/latest/

Input mandatory infos for running

-l <path/to/file/csv>, **--lib** <path/to/file/csv>
: path to file with size library, use to normalize data

Input infos not mandatory

-v, --version
Use if you want to know which version of filter_mummer.py you are using

-h, --help
show this help message and exit

-d, --debug
enter verbose/debug mode

-p, --plot
plot connections distribution

-c <int>, **--chromosome** <int>
Minimum of scaffold size (default = 1000000)

-f <int>, **--fragments** <int>
Minimum of connection size (default = 5000)

Use it to import very handy functions.

Warning: This module run with

Install

Required Module install

Modules `BioPython` will be install with :class:`yoda_powers`:

Global install

```
# for all users (require root privilege)
sudo pip3 install yoda_powers

# for own
pip3 install yoda_powers --user
```

Developing version

If you want to use an **unofficial version** of the `yoda_powers` module, you need to work from a clone of this `git` repository. following actions:

1. Clone from github

```
$ git clone https://github.com/sravel/yoda-powers.git
```

2. Go in the cloned directory

```
$ cd yoda-powers
```

3. Install `yoda-powers` in editable mode

```
$ sudo pip3 install -e .
```

Sub module information

This module are split on three sub-module

- `yoda_powers.toolbox`: handy functions common for many scripts to easy check file/directory, load file info, ...
- `yoda_powers.display`: handy functions to display/write python object like dict of dict.
- `yoda_powers.bio`: handy functions to manipulate biological data with Biopython.

Example of usage

Example:

```
>>> from yoda_powers.display import dict_2_txt
>>> dico = {'key1':'value1','key2':'value2','key3':'value3'}
>>> dict_2_txt(dico)
key1    value1
key2    value2
key3    value3
```

more info at https://yoda-powers.readthedocs.io/en/latest/

yoda_powers.toolbox Package

Handy functions common for many scripts to easy check file/directory, load file info, ...

Functions

compare_list(list1, list2)	Function to compare two list and return common, uniq1 and uniq2
existant_file(path)	'Type' for argparse - checks that file exists and return the absolute path as PosixPath() with pathlib
load_in_dict(filename[, sep])	Check file exist and return a `dict` with load rows first column is the key and value are other column.
load_in_dict_dict(filename[, sep])	Check file exist and return a `dict` with load rows first column is the key and value are other column.
load_in_dict_selected(filename[, ...])	Check file exist and return a `dict` with load rows first column is the key and value are other column.
load_in_list(filename)	Check file exist and create generator with no line break file can be a gzip file with '.gz' extension
load_in_list_col(filename[, col, sep])	Check file exist and create generator with only selected column with no line break file can be a gzip file with '.gz' extension
max_key_dict(dico)	Function return the key of max value in dico values()
readable_dir(prospective_dir)	'Type' for argparse - checks that directory exists and if readable, then return the absolute path as PosixPath() with pathlib
replace_all(repls, str)	Function that take a dictionnary and text variable and return text variable with replace 'Key' from dictionnary with 'Value'.
sort_human(in_list[, _nsre])	Sort a `list` with alpha/digit on the way that humans expect,
welcome_args(version_arg, parser_arg)	use this Decorator to add information to scripts with arguments

compare_list

yoda_powers.toolbox.**compare_list**(*list1*, *list2*)

Function to compare two list and return common, uniq1 and uniq2

Parameters

- **list1** (*list*) – the first python `list`
- **list2** (*list*) – the second python `list`

Returns common, u1, u2 common: the common elements of the 2 list, u1: uniq to list1, u2: uniq to list2

Return type list

Notes

ens1 = set([1, 2, 3, 4, 5, 6])

ens2 = set([2, 3, 4])

ens3 = set([6, 7, 8, 9])

print(ens1 & ens2) set([2, 3, 4]) car ce sont les seuls à être en même temps dans ens1 et ens2

print(ens1 | ens3) set([1, 2, 3, 4, 5, 6, 7, 8, 9]), les deux réunis

print(ens1 & ens3) set([6]), même raison que deux lignes au dessus

print(ens1 ^ ens3) set([1, 2, 3, 4, 5, 7, 8, 9]), l'union moins les éléments communs

print(ens1 - ens2) set([1, 5, 6]), on enlève les éléments de ens2

Examples

```
>>> l1 = [1, 2, 3, 4, 5, 6]
>>> l2 = [6, 7, 8, 9]
>>> com, u1, u2 = compare_list(l1, l2)
>>> print(com)
[6]
>>> print(u1)
[1, 2, 3, 4, 5]
>>> print(u2)
[7, 8, 9]
```

---

existant_file

yoda_powers.toolbox.**existant_file**(*path*)

'Type' for argparse - checks that file exists and return the absolute path as PosixPath() with pathlib

---

Notes

function need modules:

- pathlib
- argparse

Parameters **path** (*str*) – a path to existent file

Returns `Path(path).resolve()`

Return type `PosixPath`

Raises

- **ArgumentTypeError** – If file *path* does not exist.
- **ArgumentTypeError** – If *path* is not a valid file.

Examples

```
>>> import argparse
>>> parser = argparse.ArgumentParser(prog='test.py', description=''
↪'This is demo''')
>>> parser.add_argument('-f', '--file', metavar="<path/to/file>",
↪type=existant_file, required=True,
    dest='path_file', help='path to file')
```

load_in_dict

yoda_powers.toolbox.**load_in_dict**(*filename*, *sep='\t'*)

Check file exist and return a `dict` with load rows first column is the key and value are other column. File can be a gzip file with '.gz' extension.

Parameters

- **filename** (*str*) – a path to existent file
- **sep** (*str, default*) – the string separator. Default=" "

Returns a python `dict` of file

Return type `dict`

Raises **FileNotFoundError** – If file *filename* does not exist or not valid file

Example

```
>>> dico = load_in_dict(filename)
>>> dico
{
"col1",["col2","col3"],
"indiv1",["valeurcol2","valeurcol3"],
"indiv2",["valeurcol2","valeurcol3"]
}
```

load_in_dict_dict

yoda_powers.toolbox.**load_in_dict_dict**(*filename*, *sep='\t'*)

Check file exist and return a `dict` with load rows first column is the key and value are other column. File can be a gzip file with '.gz' extension.

Parameters

- **filename** (*str*) – a path to existent file
- **sep** (*str, default*) – the string separator. Default=" "

Returns a python `dict` of file

Return type `dict`

Raises

- **FileNotFoundError** – If file *filename* does not exist or not valid file
- **IndexError** – If missing data

Example

```
>>> dico = load_in_dict_dict(filename)
>>> dico
{
"indiv1",{"headerCol2":"toto","headerCol3":"tata"},
"indiv2",{"headerCol2":"tutu","headerCol3":"titi"},
"indiv3",{"headerCol2":"tete","headerCol3":"tyty"},
}
```

load_in_dict_selected

yoda_powers.toolbox.**load_in_dict_selected**(*filename*, *column_key=0*, *column_value=1*, *sep='\t'*)

Check file exist and return a `dict` with load rows first column is the key and value are other column. File can be a gzip file with '.gz' extension.

Parameters

- **filename** (*str*) – a path to existent file
- **column_key** (*int, default*) – the index for dict keys (python index). Default=0
- **column_value** (*int, default*) – the index for dict value (python index). Default=1
- **sep** (*str, default*) – the string separator. Default=" "

Returns a python `dict` of file

Return type `dict`

Raises

- **FileNotFoundError** – If file *filename* does not exist or not valid file

- **IndexError** – If missing data

Example

```
>>> dico = load_in_dict(filename)
>>> dico
{
"col1",["col2","col3"],
"indiv1",["valeurcol2","valeurcol3"],
"indiv2",["valeurcol2","valeurcol3"]
}
```

load_in_list

yoda_powers.toolbox.**load_in_list**(*filename*)

Check file exist and create generator with no line break file can be a gzip file with '.gz' extension

Notes

function need modules:

- pathlib

Parameters **filename** (*str*) – a path to existent file

Yields str – generator of rows without line break

Raises **FileNotFoundError** – If file *filename* does not exist or not valide file

Example

```
>>> rows = load_in_list("filename")
>>> list(rows)
["i like pears, but apples scare me","i like apples, but pears scare me
↪","End of file"]
```

load_in_list_col

yoda_powers.toolbox.**load_in_list_col**(*filename*, *col=0*, *sep='\t'*)

Check file exist and create generator with only selected column with no line break file can be a gzip file with '.gz' extension

Parameters

- **filename** (*str*) – a path to existent file
- **col** (*int, default*) – the selected column (python index). Default=0
- **sep** (*str, default*) – the string separator. Default=" "

Yields `str` – generator of rows without line break

Raises **`FileNotFoundError`** – If file *filename* does not exist or not valide file

---

Example

```
>>> rows = load_in_list_col("filename", col=1, sep=";")
>>> list(rows)
["i like pears, but apples scare me","i like apples, but pears scare me
 ↪","End of file"]
```

---

max_key_dict

`yoda_powers.toolbox.`**`max_key_dict`**(*dico*)

Function return the key of max value in dico values()

Parameters **`dico`** (`dict`) – a python `dict`

Returns key of the dict

Return type str

Example

```
>>> dico = {"A":0.5, "C":0.7, "T":0.01, "G":0.9}
>>> key_max = max_key_dict(dico)
>>> print(key_max)
G
```

---

readable_dir

`yoda_powers.toolbox.`**`readable_dir`**(*prospective_dir*)

'Type' for argparse - checks that directory exists and if readable, then return the absolute path as PosixPath() with pathlib

---

Notes

function need modules:

- pathlib
- argparse

Parameters **`prospective_dir`** (*`str`*) – a path to existent path

Returns `Path(path).resolve()`

Return type `PosixPath`

Raises

- **ArgumentTypeError** – If directory *path* does not exist.
- **ArgumentTypeError** – If *path* is not a valid directory.

Examples

```
>>> import argparse
>>> parser = argparse.ArgumentParser(prog='test.py', description=''
↪'This is demo''')
>>> parser.add_argument('-f', '--file', metavar="<path/to/file>",
↪type=readable_dir, required=True,
    dest='path_file', help='path to file')
```

replace_all

yoda_powers.toolbox.**replace_all**(*repls*, *str*)

Function that take a dictionnary and text variable and return text variable with replace 'Key' from dictionnary with 'Value'.

Parameters

- **repls** (*dict()*) – a python dictionary
- **str** (*str()*) – a string where remplace some words

Return type str()

Returns

- txt with replace 'Key' of dictionnary with 'Value' in the input txt

Example

```
>>> text =  "i like apples, but pears scare me"
>>> print(replace_all({"apple": "pear", "pear": "apple"}, text))
i like pears, but apples scare me
```

sort_human

yoda_powers.toolbox.**sort_human**(*in_list*, *_nsre=None*)

Sort a `list` with alpha/digit on the way that humans expect,

use list.sort(key=sort_human) or

sorted(list, key=sort_human)).

Parameters

- **in_list** (`list`) – a python `list`
- **_nsre** (`re.compil`, optional) – re expression use for compare , defaults re.compile('([0-9]+)'

Returns sorted with human sort number

Return type list

Example

```
>>> list_to_sorted = ["something1","something32","something17",
↪"something2","something29","something24"]
>>> print(sorted(list_to_sorted, key=sort_human))
['something1', 'something2', 'something17', 'something24', 'something29
↪', 'something32']
>>> list_to_sorted.sort(key=sort_human)
>>> print(list_to_sorted)
['something1', 'something2', 'something17', 'something24', 'something29
↪', 'something32']
```

welcome_args

yoda_powers.toolbox.**welcome_args**(*version_arg*, *parser_arg*)

use this Decorator to add information to scripts with arguments

Parameters

- **version_arg** – the program version
- **parser_arg** – the function which return `argparse.ArgumentParser`

Returns

Return type None

Notes

use at main() decorator for script with `argparse.ArgumentParser`

Examples

```
>>> from yoda_powers.toolbox import welcome_args
>>> @welcome_args(version, build_parser())
>>> def main():
>>>     # some code
>>> main()
>>> ##################################################################
↪#############
>>> #                                prog_name and version
↪            #
>>> ##################################################################
↪#############
>>> Start time: 16-09-2020 at 14:39:02
>>> Commande line run: ./filter_mummer.py -l mummer/GUY0011.pp1.fasta.
↪PH0014.pp1.fasta.mum
```

(continues on next page)

(continued from previous page)

```
>>>
>>> - Intput Info:
>>>         - debug: False
>>>         - plot: False
>>>         - scaff_min: 1000000
>>>         - fragments_min: 5000
>>>         - csv_file: blabla
>>> PROGRAMME CODE HERE
>>> Stop time: 16-09-2020 at 14:39:02       Run time: 0:00:00.139732
>>> ##########################################################################
↪#############
>>> #                                  End of execution
↪             #
>>> ##########################################################################
↪#############
```

Classes

AutoVivification	Implementation of perl's autovivification feature.
Directory(*args, **kwargs)	Class which derives from PosixPath.
Path(*args, **kwargs)	PurePath subclass that can make system calls.
PosixPath(*args, **kwargs)	Path subclass for non-Windows systems.
PrintCol()	Classe qui ajoute des méthodes à print pour afficher de la couleur
datetime(year, month, day[, hour[, minute[, ...)	The year, month and day arguments are required.

AutoVivification

class yoda_powers.toolbox.**AutoVivification**

Bases: dict

Implementation of perl's autovivification feature.

Example:

```
>>> a = AutoVivification()
>>> a[1][2][3] = 4
>>> a[1][3][3] = 5
>>> a[1][2]['test'] = 6
>>> print(a)
>>> {1: {2: {'test': 6, 3: 4}, 3: {3: 5}}}
```

Methods Summary

clear()	
copy()	
fromkeys([value])	Create a new dictionary with keys from iterable and values set to value.
get(key[, default])	Return the value for key if key is in the dictionary, else default.
items()	
keys()	
pop(k[,d])	If key is not found, d is returned if given, otherwise KeyError is raised
popitem()	2-tuple; but raise KeyError if D is empty.
setdefault(key[, default])	Insert key with a value of default if key is not in the dictionary.
update([E,]**F)	If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]
values()	

Methods Documentation

clear() → None. Remove all items from D.

copy() → a shallow copy of D

fromkeys(*value=None*, /)
Create a new dictionary with keys from iterable and values set to value.

get(*key*, *default=None*, /)
Return the value for key if key is in the dictionary, else default.

items() → a set-like object providing a view on D's items

keys() → a set-like object providing a view on D's keys

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised

popitem() → (k, v), remove and return some (key, value) pair as a
2-tuple; but raise KeyError if D is empty.

setdefault(*key*, *default=None*, /)
Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

update([*E*], ***F*) → None. Update D from dict/iterable E and F.
If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

values() → an object providing a view on D's values

Directory

class yoda_powers.toolbox.**Directory**(**args*, ***kwargs*)

Bases: pathlib.PosixPath

Class which derives from PosixPath. Checks that the string is and path to valid directory add function like list all files/dirs

---

Example

```
>>> dir = Directory("./")
>>> print(dir)
>>> print(dir.list_files)
>>> for file in dir.list_files_ext([".py"]):
>>>     print(file)
```

Attributes Summary

anchor	The concatenation of the drive and root, or ''.
drive	The drive prefix (letter or UNC path), if any.
list_dir	Generator of directory include on folder
list_files	Generator of files include on folder
list_path	Generator of files/directory include on folder
name	The final path component, if any.
parent	The logical parent of the path.
parents	A sequence of this path's logical parents.
parts	An object providing sequence-like access to the components in the filesystem path.
root	The root of the path, if any.
stem	The final path component, minus its last suffix.
suffix	The final component's last suffix, if any.
suffixes	A list of the final component's suffixes, if any.

Methods Summary

absolute()	Return an absolute version of this path.
as_posix()	Return the string representation of the path with forward (/) slashes.
as_uri()	Return the path as a 'file' URI.
chmod(mode)	Change the permissions of the path, like os.chmod().

continues on next page

Table 5 – continued from previous page

cwd()	Return a new path pointing to the current working directory (as returned by os.getcwd()).
exists()	Whether this path exists.
expanduser()	Return a new path with expanded ~ and ~user constructs (as returned by os.path.expanduser)
glob(pattern)	Iterate over this subtree and yield all existing files (of any kind, including directories) matching the given relative pattern.
group()	Return the group name of the file gid.
home()	Return a new path pointing to the user's home directory (as returned by os.path.expanduser('~')).
is_absolute()	True if the path is absolute (has both a root and, if applicable, a drive).
is_block_device()	Whether this path is a block device.
is_char_device()	Whether this path is a character device.
is_dir()	Whether this path is a directory.
is_fifo()	Whether this path is a FIFO.
is_file()	Whether this path is a regular file (also True for symlinks pointing to regular files).
is_mount()	Check if this path is a POSIX mount point
is_reserved()	Return True if the path contains one of the special names reserved by the system, if any.
is_socket()	Whether this path is a socket.
is_symlink()	Whether this path is a symbolic link.
iterdir()	Iterate over the files in this directory.
joinpath(*args)	Combine this path with one or several arguments, and return a new path representing either a subpath (if all arguments are relative paths) or a totally different path (if one of the arguments is anchored).
lchmod(mode)	Like chmod(), except if the path points to a symlink, the symlink's permissions are changed, rather than its target's.
list_files_ext([ext])	Generator of files with specify extension include on folder
lstat()	Like stat(), except if the path points to a symlink, the symlink's status information is returned, rather than its target's.
match(path_pattern)	Return True if this path matches the given pattern.
mkdir([mode, parents, exist_ok])	Create a new directory at this given path.
open([mode, buffering, encoding, errors, ...])	Open the file pointed by this path and return a file object, as the built-in open() function does.
owner()	Return the login name of the file owner.
read_bytes()	Open the file in bytes mode, read it, and close the file.

continues on next page

Table 5 – continued from previous page

read_text([encoding, errors])	Open the file in text mode, read it, and close the file.
relative_to(*other)	Return the relative path to another path identified by the passed arguments.
rename(target)	Rename this path to the given path.
replace(target)	Rename this path to the given path, clobbering the existing destination if it exists.
resolve([strict])	Make the path absolute, resolving all symlinks on the way and also normalizing it (for example turning slashes into backslashes under Windows).
rglob(pattern)	Recursively yield all existing files (of any kind, including directories) matching the given relative pattern, anywhere in this subtree.
rmdir()	Remove this directory.
samefile(other_path)	Return whether other_path is the same or not as this file (as returned by os.path.samefile()).
stat()	Return the result of the stat() system call on this path, like os.stat() does.
symlink_to(target[, target_is_directory])	Make this path a symlink pointing to the given path.
touch([mode, exist_ok])	Create this file with the given access mode, if it doesn't exist.
unlink()	Remove this file or link.
with_name(name)	Return a new path with the file name changed.
with_suffix(suffix)	Return a new path with the file suffix changed.
write_bytes(data)	Open the file in bytes mode, write to it, and close the file.
write_text(data[, encoding, errors])	Open the file in text mode, write to it, and close the file.

Attributes Documentation

anchor
The concatenation of the drive and root, or ‘’.

drive
The drive prefix (letter or UNC path), if any.

list_dir
Generator of directory include on folder

list_files
Generator of files include on folder

list_path
Generator of files/directory include on folder

name
The final path component, if any.

parent
The logical parent of the path.

parents
A sequence of this path's logical parents.

parts
An object providing sequence-like access to the components in the filesystem path.

root
The root of the path, if any.

stem
The final path component, minus its last suffix.

suffix
The final component's last suffix, if any.

This includes the leading period. For example: '.txt'

suffixes
A list of the final component's suffixes, if any.

These include the leading periods. For example: ['.tar', '.gz']

Methods Documentation

absolute()
Return an absolute version of this path. This function works even if the path doesn't point to anything.

No normalization is done, i.e. all '.' and '..' will be kept along. Use resolve() to get the canonical path to a file.

as_posix()
Return the string representation of the path with forward (/) slashes.

as_uri()
Return the path as a 'file' URI.

chmod(*mode*)
Change the permissions of the path, like os.chmod().

classmethod cwd()
Return a new path pointing to the current working directory (as returned by os.getcwd()).

exists()
Whether this path exists.

expanduser()
Return a new path with expanded ~ and ~user constructs (as returned by os.path.expanduser)

glob(*pattern*)
Iterate over this subtree and yield all existing files (of any kind, including directories) matching the given relative pattern.

group()
Return the group name of the file gid.

classmethod home()
Return a new path pointing to the user's home directory (as returned by os.path.expanduser('~')).

is_absolute()
True if the path is absolute (has both a root and, if applicable, a drive).

is_block_device()
Whether this path is a block device.

is_char_device()
Whether this path is a character device.

is_dir()
Whether this path is a directory.

is_fifo()
Whether this path is a FIFO.

is_file()
Whether this path is a regular file (also True for symlinks pointing to regular files).

is_mount()
Check if this path is a POSIX mount point

is_reserved()
Return True if the path contains one of the special names reserved by the system, if any.

is_socket()
Whether this path is a socket.

is_symlink()
Whether this path is a symbolic link.

iterdir()
Iterate over the files in this directory. Does not yield any result for the special paths '.' and '..'.

joinpath(**args*)
Combine this path with one or several arguments, and return a new path representing either a subpath (if all arguments are relative paths) or a totally different path (if one of the arguments is anchored).

lchmod(*mode*)
Like chmod(), except if the path points to a symlink, the symlink's permissions are changed, rather than its target's.

list_files_ext(*ext=None*)
Generator of files with specify extension include on folder

Parameters **ext** (*list*) – a list of extension like [".py"]

Yields PosixPath – Generator of files with specify extension include on folder

lstat()
Like stat(), except if the path points to a symlink, the symlink's status information is returned, rather than its target's.

match(*path_pattern*)
Return True if this path matches the given pattern.

mkdir(*mode=511*, *parents=False*, *exist_ok=False*)
Create a new directory at this given path.

open(*mode='r'*, *buffering=- 1*, *encoding=None*, *errors=None*, *newline=None*)
Open the file pointed by this path and return a file object, as the built-in open() function does.

owner()
Return the login name of the file owner.

read_bytes()
Open the file in bytes mode, read it, and close the file.

read_text(*encoding=None*, *errors=None*)
Open the file in text mode, read it, and close the file.

relative_to(**other*)
Return the relative path to another path identified by the passed arguments. If the operation is not possible (because this is not a subpath of the other path), raise ValueError.

rename(*target*)
Rename this path to the given path.

replace(*target*)
Rename this path to the given path, clobbering the existing destination if it exists.

resolve(*strict=False*)
Make the path absolute, resolving all symlinks on the way and also normalizing it (for example turning slashes into backslashes under Windows).

rglob(*pattern*)
Recursively yield all existing files (of any kind, including directories) matching the given relative pattern, anywhere in this subtree.

rmdir()
Remove this directory. The directory must be empty.

samefile(*other_path*)
Return whether other_path is the same or not as this file (as returned by os.path.samefile()).

stat()
Return the result of the stat() system call on this path, like os.stat() does.

symlink_to(*target*, *target_is_directory=False*)
Make this path a symlink pointing to the given path. Note the order of arguments (self, target) is the reverse of os.symlink's.

touch(*mode=438*, *exist_ok=True*)
Create this file with the given access mode, if it doesn't exist.

unlink()
Remove this file or link. If the path is a directory, use rmdir() instead.

with_name(*name*)
Return a new path with the file name changed.

with_suffix(*suffix*)
Return a new path with the file suffix changed. If the path has no suffix, add given suffix. If the given suffix is an empty string, remove the suffix from the path.

write_bytes(*data*)
Open the file in bytes mode, write to it, and close the file.

write_text(*data*, *encoding=None*, *errors=None*)
Open the file in text mode, write to it, and close the file.

PrintCol

class yoda_powers.toolbox.**PrintCol**
Bases: object

Classe qui ajoute des méthodes à print pour afficher de la couleur

Example:

```
>>> PrintCol.red("j'affiche en rouge")
j'affiche en rouge
```

Methods Summary

green(s)
lightPurple(s)
purple(s)
red(s)
yellow(s)

Methods Documentation

classmethod green(*s*)

classmethod lightPurple(*s*)

classmethod purple(*s*)

classmethod red(*s*)

classmethod yellow(*s*)

yoda_powers.display Package

Handy functions to display/write python object like dict of dict.

Functions

dict_2_txt(dico[, sep])	Function that takes a dictionary and returns a string with separator:
dict_dict_2_txt(dico[, first, sep])	Function that takes a dictionary and returns a tabular string with:
dict_list_2_txt(dico[, sep])	Function that takes a dictionary of list and returns a tabular string with:
sort_human(in_list[, _nsre])	Sort a `list` with alpha/digit on the way that humans expect,

dict_2_txt

yoda_powers.display.**dict_2_txt**(*dico*, *sep='\t'*)

Function that takes a dictionary and returns a string with separator:

Parameters

- **dico** (dict) – the python dict to translate to formated string.
- **sep** (str) – the separator for join . Defaults to 't'.

Returns formated dict to string

Return type str

Examples

```
>>> dico = {"key1":"value1","key2":"value2","key3":"value3"}
>>> dict_2_txt(dico)
key1    value1
key2    value2
key3    value3
>>> dict_2_txt(dico, sep=";")
key1;value1
key2;value2
key3;value3
```

Warning: if the value of the dict is list or dictionary, the display will be plain and without formatting data.

dict_dict_2_txt

yoda_powers.display.**dict_dict_2_txt**(*dico*, *first='Info'*, *sep='\t'*)

Function that takes a dictionary and returns a tabular string with:

Parameters

- **dico** (dict) – the python dict to translate to formated string.
- **first** (str) – the first column header name. Default to 'Info'.
- **sep** (str) – the separator for join. Defaults to 't'.

Returns formated dict to string

Return type str

Examples

```
>>> dico = {"Souche1":{"NUM":"171","MIN":"2042","MAX":"3133578","N50 BP
↪":"938544","N50 NUM":"11"},
            "Souche2":{"NUM":"182","MIN":"5004","MAX":"74254","N50 BP":
↪"45245"}}
>>> dict_dict_2_txt(dico,"souches")
souches NUM     MIN     MAX     N50 BP  N50 NUM
Souche1 171     2042    3133578 938544  11
Souche2 182     5004    74254   45245   None
```

dict_list_2_txt

yoda_powers.display.**dict_list_2_txt**(*dico*, *sep='\t'*)

Function that takes a dictionary of list and returns a tabular string with:

Parameters

- **dico** (dict) – the python dict to translate to formated string.
- **sep** (str) – the separator for join . Defaults to 't'.

Returns formated dict to string

Return type str

Examples

```
>>> dico = {"key1":["value1","value1"], "key2":["value2","value2"],
↪"key3":["value3","value3"]}
>>> dict_list_2_txt(dico)
key1    value1  value1
key2    value2  value2
key3    value3  value3
```

yoda_powers.bio Package

Handy functions to manipulate biological data with Biopython.

Functions

concat_fasta_files(path_directory)	Return a fasta dictionnary of concatenation fasta file's find in directory ("fasta", "fa", "fas")
convert_fasta_2_nexus(path_directory, ...)	Return the number of fasta file's convert find in directory ("fasta", "fa", "fas") where are converted
dict_2_fasta(dico, fasta_out)	Function that takes a dictionary where key are ID and value Seq, and write a fasta file.
extract_seq_from_fasta(fasta_file, wanted_file)	Function to extract sequence from fasta file
fasta_2_dict(fasta_file)	Function that take a file name (fasta), and return a dictionnary of sequence
len_seq_2_dict(fasta_file)	Function that take a file name (fasta), and return a dictionnary with length of sequence
nb_seq_files_2_dict(path_directory)	Function that take a Path Directory and returna dictionnary with number of sequences in fasta file's

concat_fasta_files

yoda_powers.bio.**concat_fasta_files**(*path_directory*)

Return a fasta dictionnary of concatenation fasta file's find in directory ("fasta", "fa", "fas")

> **Warning:** Sequence on fasta must have the same name

Notes

function need modules:

- pathlib
- BioPython

Parameters **path_directory** (*str*) – a path to fasta file directory

Returns python dict with the concatenation of fasta filename in path_directory (file with extention "fa", "fasta", "fas")

Return type dict

Raises

- **ValueError** – If *path_directory* does not exist.

- **ValueError** – If *path_directory* is not a valid directory.

Examples

```
>>> dico_concat = concat_fasta_files('path/to/directory/')
>>> print(dico_concat)
    {"Seq1":"ATGCTGCAGTAG","Seq2":"ATGCCGATCGATG","Seq3":
↪"ATGCTCAGTCAGTAG"}
```

convert_fasta_2_nexus

yoda_powers.bio.**convert_fasta_2_nexus**(*path_directory*, *path_directory_out*)

Return the number of fasta file's convert find in directory ("fasta", "fa", "fas") where are converted

Warning: Sequence on fasta must align and have the same length

Notes

function need modules:

- pathlib
- BioPython

Parameters

- **path_directory** (*str*) – a path to fasta file directory
- **path_directory_out** (*str*) – a directory path to write nexus file

Returns the number of file converted

Return type int

Raises

- **ValueError** – If *path_directory* does not exist.
- **ValueError** – If *path_directory* is not a valid directory.
- **ValueError** – If fasta is not align.

Examples

```
>>> nb_file = convert_fasta_2_nexus('path/to/directory/',' path/to/
↪directory/')
>>> print(nb_file)
    "4172"
```

dict_2_fasta

yoda_powers.bio.**dict_2_fasta**(*dico*, *fasta_out*)

Function that takes a dictionary where key are ID and value Seq, and write a fasta file.

Notes

function need modules:

- pathlib
- BioPython

Parameters

- **dico** (*dict*) – python dict with ID in key and Seq on values
- **fasta_out** (*str*) – a directory path to write nexus file

Returns the output fasta file name

Return type str

Examples

```
>>> dico = {"Seq1":"ATGCTGCAGTAG","Seq2":"ATGCCGATCGATG","Seq3":
↪"ATGCTCAGTCAGTAG"}
>>> dict_2_fasta(dico)
>Seq1
ATGCTGCAGTAG
>Seq2
ATGCCGATCGATG
>Seq3
ATGCTCAGTCAGTAG
```

---

extract_seq_from_fasta

yoda_powers.bio.**extract_seq_from_fasta**(*fasta_file*, *wanted_file*, *include=True*)

Function to extract sequence from fasta file

Notes

function need modules:

- pathlib
- BioPython

Parameters

- **fasta_file** (*str*) – a path to fasta file directory

- **wanted_file** (*str*) – a path file with id (one per line)
- **include** (*bool, optional*) – if True keep id on wanted file, else keep id not in wanted file

Returns the fasta dict with sequence extract

Return type dict

Raises

- **ValueError** – If *wanted_file* or *fasta_file* does not exist.
- **ValueError** – If *wanted_file* or *fasta_file*` is not a valid file.
- **ValueError** – If *include* is not valid boolean.

Example

```
>>> dict_sequences = extract_seq_from_fasta(fasta_file, wanted_file)
>>> dict_sequences
{'Seq2': SeqRecord(seq=Seq('ATGCCGATCGATG', SingleLetterAlphabet()),␣
↪id='Seq2', name='Seq2', description='Seq2'
, dbxrefs=[]), 'Seq3': SeqRecord(seq=Seq('ATGCTCAGTCAGTAG',␣
↪SingleLetterAlphabet()), id='Seq3', name='Seq3',
description='Seq3', dbxrefs=[])}
```

---

fasta_2_dict

yoda_powers.bio.**fasta_2_dict**(*fasta_file*)

Function that take a file name (fasta), and return a dictionnary of sequence

---

Notes

function need modules:

- pathlib
- BioPython

Parameters **fasta_file** (*str*) – a path to fasta file directory

Returns the fasta dict with sequence extract

Return type dict

Raises

- **ValueError** – If *fasta_file* does not exist.
- **ValueError** – If *fasta_file* is not a valid file.

Example

```
>>> filename = "sequence.fasta"
>>> fasta_2_dict(filename)
{'Seq1': SeqRecord(seq=Seq('ATGCTGCAGTAG', SingleLetterAlphabet()), id=
↪'Seq1', name='Seq1', description='Seq1', dbxrefs=[]),
'Seq2': SeqRecord(seq=Seq('ATGCCGATCGATG', SingleLetterAlphabet()), id=
↪'Seq2', name='Seq2', description='Seq2', dbxrefs=[]),
'Seq3': SeqRecord(seq=Seq('ATGCTCAGTCAGTAG', SingleLetterAlphabet()),␣
↪id='Seq3', name='Seq3', description='Seq3', dbxrefs=[])}
```

len_seq_2_dict

yoda_powers.bio.**len_seq_2_dict**(*fasta_file*)

Function that take a file name (fasta), and return a dictionnary with length of sequence

Notes

function need modules:

- pathlib
- BioPython

Parameters **fasta_file** (*str*) – a path to fasta file directory

Returns the fasta dict with sequence length

Return type dict

Raises

- **ValueError** – If *fasta_file* does not exist.
- **ValueError** – If *fasta_file`* is not a valid file.

Example

```
>>> filename = "sequence.fasta"
>>> len_seq_2_dict(filename)
{'Seq1': 12, 'Seq2': 13, 'Seq3': 15}
```

nb_seq_files_2_dict

yoda_powers.bio.**nb_seq_files_2_dict**(*path_directory*)

Function that take a Path Directory and returna dictionnary with number of sequences in fasta file's

Parameters **pathDirectory** (*Path*) – a directory Path

Return type dict1(), dict2()

Returns

- contient le nombre de sequences dans les fichiers (key = nom de fichier value = nombre de sequences)
- contient le nombre de fichier qui ont x sequences (key = nombre de sequence = nombre de fichier)

Raises **print** – print("ERROR: Sequence: "+nameFichier+" allready read") with nameFichier is the current file read.

Example

```
>>> dico1,dico2 = nbSeqInFile2dict(path/to/directory/)
>>> print(dict2txt(dico1))
./out/gemo10_4497_ortho_rename_add.fasta        58
./out/gemo10_6825_ortho_rename_add.fasta        59
./out/gemo10_3497_ortho_rename_add.fasta        59
./out/gemo10_6254_ortho_rename_add.fasta        59
>>> print(dict2txt(dico2))
58      1
59      3
```

Classes

ParseGFF(filename)	Parser of GFF3 file write in python.

ParseGFF

class yoda_powers.bio.**ParseGFF**(*filename*)

Bases: object

Parser of GFF3 file write in python. return an object iterable contain GFFRecord()

line in GFF3 return:

Example

```
>>> scaffold_44     prediction      gene    46      6942    0       +
↪       .       ID=gene_1;Name=jgi.p|Mycfi2|180833;portal_id=Mycfi2;
↪proteinId=180833;transcriptId=180833
```

(continues on next page)

(continued from previous page)

```
>>> GFFRecord(seqid='scaffold_44', source='prediction', type='gene',␣
↪start=46, end=6942, score=0.0, strand='+', phase=None,
>>>              attributes={'portal_id': 'Mycfi2', 'transcriptId':
↪'180833', 'proteinId': '180833', 'Name': 'jgi.p|Mycfi2|180833', 'ID
↪': 'gene_1'}, seq=None, len=6896)
```

GFFRecord has attributes can acces with record.value (ex: record.seqid):

attribute	infos
seqid	first column of gff3
source	second column of gff3
type	third column of gff3 contain type
start	start position
end	end position
score	score
strand	DNA brin
phase	phase
attributes	dict() with key corresponding to GFFAttributes
seq	if fasta load can add sequence but by default = None
len	size of sequence

Example

```
>>> objGFF = ParseGFF(gffFile)
>>> for record in objGFF.parseGFF3():
>>>     print(record.seqid)
>>>     if record.type == "mRNA" :
>>>             transcriptID = record.attributes["transcriptId"]
```

Methods Summary

parseGFF3()	A minimalistic GFF3 format parser.
parseGFFAttributes(self, attributeString)	Parse the GFF3 attribute column and return a dict

Methods Documentation

parseGFF3()
A minimalistic GFF3 format parser. Yields objects that contain info about a single GFF3 feature.

Supports transparent gzip decompression.

static parseGFFAttributes(*self*, *attributeString*)
Parse the GFF3 attribute column and return a dict

PYTHON MODULE INDEX

s

y

INDEX

Symbols

A

C

M

N

O

P

R

S

T

U

V

W

Y