

## SNPHeatMap

Generated by Doxygen 1.9.1



<b>1 Snp HeatMap</b>	<b>1</b>
1.1 Installation :	1
1.2 File format :	1
1.3 Usage :	2
1.3.1 Quick usage :	2
1.3.2 Complete usage :	2
1.3.2.1 Arguments	3
1.3.2.2 Parameters	3
1.4 Example chart	5
1.4.1 Quantitative chart :	5
1.4.2 Cumulative chart :	5
1.4.3 Cumulative Heatmap :	5
1.4.4 Global Heatmap :	5
1.5 NOTE FOR LATER	5
<b>2 Namespace Index</b>	<b>7</b>
2.1 Namespace List	7
<b>3 Hierarchical Index</b>	<b>9</b>
3.1 Class Hierarchy	9
<b>4 Class Index</b>	<b>11</b>
4.1 Class List	11
<b>5 File Index</b>	<b>13</b>
5.1 File List	13
<b>6 Namespace Documentation</b>	<b>15</b>
6.1 main Namespace Reference	15
6.1.1 Detailed Description	15
6.2 scripts Namespace Reference	15
6.3 scripts.getopts_parser Namespace Reference	15
6.4 scripts.getopts_parser.getopts_parser Namespace Reference	16
6.4.1 Function Documentation	16
6.4.1.1 getopts()	17
6.4.1.2 getopts_digest_available_options()	18
6.4.1.3 getopts_digester_check_item_endings()	19
6.4.1.4 getopts_parser()	20
6.4.1.5 getopts_parser_boolean_option()	20
6.4.1.6 getopts_parser_complex_option()	21
6.4.1.7 getopts_retrieve_options()	21
6.4.2 Variable Documentation	22
6.4.2.1 __author__	22
6.4.2.2 __credits__	22

6.5 scripts.getopts_parser.test_getopts_parser Namespace Reference	22
6.5.1 Function Documentation	23
6.5.1.1 test_aliases()	23
6.5.1.2 test_all()	23
6.5.1.3 test_defaults_values()	23
6.5.1.4 test_simple_option()	23
6.5.2 Variable Documentation	23
6.5.2.1 __author__	23
6.5.2.2 __credits__	24
6.6 scripts.snp_analyser Namespace Reference	24
6.6.1 Function Documentation	24
6.6.1.1 compile_gene_snp()	24
6.6.1.2 filter_integer_greater_or_equal_to_0_ignore_0()	25
6.6.1.3 generate_cumulative_list()	26
6.6.1.4 help_usage()	26
6.6.1.5 main()	26
6.6.1.6 main_using_getopts()	28
6.6.1.7 make_data_matrix()	29
6.6.2 Variable Documentation	30
6.6.2.1 __author__	30
6.6.2.2 __credits__	30
6.6.2.3 __getopts__	30
6.7 scripts.test_snp_analyser Namespace Reference	30
6.7.1 Function Documentation	30
6.7.1.1 test_all()	31
6.7.1.2 test_compile_gene_snp()	31
6.7.1.3 test_generate_cumulative_list()	31
6.7.1.4 test_greater_than_0_int_filter()	31
6.7.1.5 test_make_data_matrix()	31
6.8 scripts.utilities Namespace Reference	31
6.9 scripts.utilities.utilities Namespace Reference	32
6.9.1 Function Documentation	32
6.9.1.1 chart_export()	32
6.9.1.2 export_list_in_tsv_as_rows()	33
6.9.1.3 extract_data_from_table()	34
6.9.1.4 make_bar_char()	35
6.9.1.5 make_heatmap()	35
6.9.1.6 parse_line()	37
<b>7 Class Documentation</b>	<b>39</b>
7.1 scripts.utilities.utilities.FilterError Class Reference	39
7.1.1 Detailed Description	39

7.2 scripts.getopts_parser.getopts_parser.GetoptsDigestionError Class Reference . . . . .	39
7.2.1 Detailed Description . . . . .	40
7.3 scripts.getopts_parser.getopts_parser.GetoptsOptionError Class Reference . . . . .	40
7.3.1 Detailed Description . . . . .	40
7.4 scripts.getopts_parser.getopts_parser.GetoptsParsingError Class Reference . . . . .	40
7.4.1 Detailed Description . . . . .	40
<b>8 File Documentation</b>	<b>41</b>
8.1 __init__.py File Reference . . . . .	41
8.2 __init__.py File Reference . . . . .	41
8.3 __init__.py File Reference . . . . .	41
8.4 getopts_parser.py File Reference . . . . .	41
8.5 main.py File Reference . . . . .	42
8.6 README.md File Reference . . . . .	42
8.7 snp_analyser.py File Reference . . . . .	42
8.7.1 Detailed Description . . . . .	43
8.7.1.1 to use this file . . . . .	43
8.7.1.2 Charts . . . . .	43
8.7.1.3 Formats . . . . .	44
8.7.2 Author(s) . . . . .	44
8.7.3 Libraries/Modules . . . . .	44
8.8 test_getopts_parser.py File Reference . . . . .	44
8.9 test_snp_analyser.py File Reference . . . . .	45
8.10 utilities.py File Reference . . . . .	45
8.10.1 Detailed Description . . . . .	45
<b>Index</b>	<b>47</b>



# Chapter 1

## Snp HeatMap

The goal of this project is to create a number of chart related to snp (simple nucleotide polymorphism) analysis

Those charts are :

- [Quantitative chart](#) (-q) : Show the number of gene (y) per number of snp (x)
- [Cumulative chart](#) (-c) : Show the number of gene (y) that have at least n snp (x)
- [Cumulative chart](#) (-u) : Cumulative bar chart but it's a heatmap
- [Global Heatmap](#) (-g) : concatenation of all Cumulative chart

### 1.1 Installation :

- Download
  1. Download Repository ("[<> Code](#)" button top right --> "Download zip")
  2. Unzip the Downloaded files
- Using linux terminal
  1. `git clone https://github.com/F-Marchal/SnpHeatMap.git`
  2. `cd SnpHeatMap`

see [Quick usage](#) and [Quick usage](#) in order to run [main.py](#)

### 1.2 File format :

Files used by this script are expected to have the following pattern :

Gene name / id	Other	Snp counter
Gene1	Other info1	x
Gene2	Other info2	y

- All files have to have the same headers
- When a gene is present multiple times, only the last occurrence is used
- Snp counter should be integer greater or equal to 0. No dot ('.') or comma (',') are allowed in this cell.

**1.2.0.0.1 Path to your files** By default, the program assumes that all your files are in the `Data/` folder.

Path toward your files can be give using two methods :

- Using folder path
  - All files inside the folder are used.
  - Names in the graph are file's names
  - Files that start with "." are ignored.
  - Is the default option
- Using a json's path
  - This file should have the following format `{path_to_a_file: common_name}`. You can open `test/TargetedFiles.json` if you need an example
  - Names in the graph are `common_name`
  - If multiple path have the same `common_name` they will be considered as the same file.

All files represent a species or a group of related individuals.

Can also be provided using parameters see [path](#)

## 1.3 Usage :

Please read [File format](#) before using `main.py` and [Path to your files](#).

### 1.3.1 Quick usage :

- Put your files inside the `data` folder
- Open a terminal in this folder or use `cd` to change terminal's current directory
- In Linux :
  - `run python3 main.py [Gene name Column] [Snp column]`
- If you use windows :
  - `run python3 main.py [Gene name Column] [Snp column]`

### 1.3.2 Complete usage :

- linux : `python3 main.py [Gene name Column] [Snp column] [Path to your files] [Options]`
- Windows : `python3 main.py [Gene name Column] [Snp column] [Path to your files] [Options]` This command will make a "global heatmap" for all your files in data



### 1.3.2.1 Arguments

- **Warning:** Arguments should always be before the parameters

**1.3.2.1.1 Gene name Column (First argument)** Name of the column that contain gene's names / ids in files. e.g. "Gene name / id" in [File format](#)

Can also be provided using parameters see [name\\_column](#)

**1.3.2.1.2 Snp column (Second argument)** Name of the column that contain snp in files. e.g. "Snp counter" in [File format](#)

Can also be provided using parameters see [snp\\_column](#)

**1.3.2.1.3 File path (Third argument)** See [Path to your files](#)

Can also be provided using parameters See [path](#)

### 1.3.2.2 Parameters

**Warning:** Parameters should always be after the arguments

**1.3.2.2.1** `<tt>--name_column</tt>` or `<tt>-n</tt>` This parameter must be followed by a string. See [gene name Column](#)

**1.3.2.2.2** `<tt>--snp_column</tt>` or `<tt>-s</tt>` This parameter must be followed by a string. See [snp column](#)

**1.3.2.2.3** `<tt>--path</tt>` or `<tt>-p</tt>` This parameter must be followed by a string that represent a path. See [path to your files](#)

**1.3.2.2.4** `<tt>--output_path</tt>` or `<tt>-o</tt>` This parameter must be followed by a string that represent a path. This parameter modify where charts and tsv are saved.

Default = ./output

**1.3.2.2.5** `<tt>--job_name</tt>` or `<tt>-j</tt>` This parameter must be followed by a string that can be used as folder name. This added a prefix to all files generated by this script.

Default = "Unnamed"

**1.3.2.2.6** `<tt>--max_length</tt>` or `<tt>-m</tt>` This parameter must be followed by an integer. If this inger is lower than 0, max\_length is ignored. Limit the number of snp shown inside each graph.

Default = 20

**1.3.2.2.7** `<tt>--help</tt>` or `<tt>-h</tt>` Display a help message.

Others parameters are ignored.

**1.3.2.2.8** `<tt>--output_warning</tt>` or `<tt>-w</tt>` Disable the warning when you are about to generate files inside a non-empty folder.

**1.3.2.2.9** `<tt>--sort_by_name</tt>` or `<tt>-r</tt>` Disable sort species by names in global heatmap.

**1.3.2.2.10** `<tt>--simplified</tt>` or `<tt>-i</tt>` Ignore snp number represented by 0 genes. THIS MAY CREATE A DISCONTINUOUS X-AXIS

**1.3.2.2.11** `<tt>--global_heatmap</tt>` or `<tt>-g</tt>` Generate a heatmap that represent all species.

**1.3.2.2.12** `<tt>--quantitative_barchart</tt>` or `<tt>-q</tt>` Generate a barchart that represent snp distribution for each file (Number of gene that have n snp)

**1.3.2.2.13** `<tt>--cumulative_barchart</tt>` or `<tt>-c</tt>` Generate a barchart that represent snp distribution for each file (Number of gene that **at least** n snp)

**1.3.2.2.14** `<tt>--cumulative_heatmap</tt>` or `<tt>-u</tt>` Generate a heatmap for each file. This heatmap contain only one line.

**1.3.2.2.15** `<tt>--tsv</tt>` or `<tt>-t</tt>` Generate a tsv for all generated charts.

**1.3.2.2.16** `<tt>--png</tt>` or `<tt>-k</tt>` Generate a png for all generated charts.

**1.3.2.2.17** `<tt>--show</tt>` or `<tt>-d</tt>` Show all generated charts during the execution. ALL CHARTS WILL STOP THE EXECUTION UNTIL IT IS CLOSED.

**1.3.2.2.18** `<tt>--svg</tt>` or `<tt>-v</tt>` Generate a svg for all generated charts.

**1.3.2.2.19** `<tt>--show_values</tt>` or `<tt>-e</tt>` An integer (positive or negative)

If greater or equal to 0, all heatmap's cells will contain theirs values. if lower than 0, text size in cell is automatically determined (can be ugly in the windows displayed by -d, but assure that the text is well sized in png and svg). If unspecified, cells are empty.

**1.3.2.2.20** `<tt>--uniform_y</tt>` or `<tt>--uniform -y</tt>` Uniformize all y-axis. All Quantitative chart and all Cumulative chart will have the same upper limit and the same scale on their y-axis. (Quantitative and Cumulative charts still have disjoint y-axis)

## 1.4 Example chart

Charts generated using `python3 Contig_name BiAllelic_SNP tests/TargetedFiles.json -m 10 -kgqcuw -j Examples --show_values -1`

The data used in these graphs comes from a random generator

### 1.4.1 Quantitative chart :

A barchart that show the proportion of gene that have a certain number of snp.

### 1.4.2 Cumulative chart :

A barchart that show the proportion of gene that have at least certain number of snp

### 1.4.3 Cumulative Heatmap :

A heatmap that show the proportion of gene that have at least certain number of snp (variation of Cumulative chart)

### 1.4.4 Global Heatmap :

A heatmap that show the proportion of gene that have at least certain number of snp for all species.

## 1.5 NOTE FOR LATER

Since the 1.1.0, the parameter `--name_column` is almost useless. The code should be reworked to remove any usage of this option.



## Chapter 2

# Namespace Index

### 2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

<a href="#">main</a>		
	Run <a href="#">scripts/snp_analyser.py</a>	15
<a href="#">scripts</a>		15
<a href="#">scripts.getopts_parser</a>		15
<a href="#">scripts.getopts_parser.getopts_parser</a>		16
<a href="#">scripts.getopts_parser.test_getopts_parser</a>		22
<a href="#">scripts.snp_analyser</a>		24
<a href="#">scripts.test_snp_analyser</a>		30
<a href="#">scripts.utilities</a>		31
<a href="#">scripts.utilities.utilities</a>		32



## Chapter 3

# Hierarchical Index

### 3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

getopt.GetoptError	
scripts.getopts_parser.getopts_parser.GetoptsDigestionError . . . . .	<a href="#">39</a>
scripts.getopts_parser.getopts_parser.GetoptsOptionError . . . . .	<a href="#">40</a>
scripts.getopts_parser.getopts_parser.GetoptsParsingError . . . . .	<a href="#">40</a>
ValueError	
scripts.utilities.utilities.FilterError . . . . .	<a href="#">39</a>





## Chapter 4

# Class Index

### 4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

<a href="#">scripts.utilities.utilities.FilterError</a>	39
<a href="#">scripts.getopts_parser.getopts_parser.GetoptsDigestionError</a>	
Error raised during the digestion of the dictionary that contain options	39
<a href="#">scripts.getopts_parser.getopts_parser.GetoptsOptionError</a>	
Error raised when there is an error with an option during the getopts	40
<a href="#">scripts.getopts_parser.getopts_parser.GetoptsParsingError</a>	
Error raised when there is an error during the parsing of a command-line	40



## Chapter 5

# File Index

### 5.1 File List

Here is a list of all files with brief descriptions:

<a href="#">__init__.py</a>	41
<a href="#">getopts_parser/__init__.py</a>	41
<a href="#">utilities/__init__.py</a>	41
<a href="#">getopts_parser.py</a>	41
<a href="#">main.py</a>	42
<a href="#">snp_analyser.py</a>	
Create a number of chart related to snp (simple nucleotide polymorphism) analysis	42
<a href="#">test_getopts_parser.py</a>	44
<a href="#">test_snp_analyser.py</a>	45
<a href="#">utilities.py</a>	
Contain a number of function related to file reading and generation of figure using matplotlib	45



## Chapter 6

# Namespace Documentation

### 6.1 main Namespace Reference

Run [scripts/snp\\_analyser.py](#).

#### 6.1.1 Detailed Description

Run [scripts/snp\\_analyser.py](#).

or any information refer to [README.md](#) or [scripts/snp\\_analyser.py](#).

### 6.2 scripts Namespace Reference

#### Namespaces

- [getopts\\_parser](#)
- [snp\\_analyser](#)
- [test\\_snp\\_analyser](#)
- [utilities](#)

### 6.3 scripts.getopts\_parser Namespace Reference

#### Namespaces

- [getopts\\_parser](#)
- [test\\_getopts\\_parser](#)

## 6.4 scripts.getopts\_parser.getopts\_parser Namespace Reference

### Classes

- class [GetoptsDigestionError](#)  
*Error raised during the digestion of the dictionary that contain options.*
- class [GetoptsOptionError](#)  
*Error raised when there is an error with an option during the getopts.*
- class [GetoptsParsingError](#)  
*Error raised when there is an error during the parsing of a command-line.*

### Functions

- any [getopts\\_parser\\_boolean\\_option](#) (tuple[any, any] value\_restriction=None, use\_default=False)  
*Internal function used by [getopts\\_parser](#).*
- any [getopts\\_parser\\_complex\\_option](#) (value\_restriction, value, use\_default=False)  
*Internal function used by [getopts\\_parser](#).*
- bool or None [getopts\\_digester\\_check\\_item\\_endings](#) (list[str] list\_of\_values, str last\_char=":")  
*Check if the last char of each string in `list_of_values` is equal to `last_char`.*
- tuple[dict[str, tuple[any, any]], dict[str, str], dict[str, str], str, list[str]] [getopts\\_digest\\_available\\_options](#) (dict[str, tuple[any, any]] getopts\_options, dict[str, any] dict\_of\_default\_value=None)  
*Transform `getopts_options` into usable instruction for [getopts\\_retrieve\\_options](#) and assure that option's aliases are coherent.*
- list[tuple[str, str]] [getopts\\_retrieve\\_options](#) (list[str] argv, str short\_string, list[str] long\_list, list[str] main\_↵ options)  
*Use a list of string from a command-line (e.g.*
- dict [getopts\\_parser](#) (list[str] or str argv, dict[str, tuple[any, any]] getopts\_options, \*str mandatory, fill\_with\_↵ default\_values=True)  
*Internal version of [getopts](#).*
- dict[str, any] or int [getopts](#) (list[str] or str argv, dict[str or tuple, None or tuple[any, any] or any] getopts\_options, \*str mandatory, str help\_message=None, fill\_with\_default\_values=True, raise\_errors=False, str or tuple[str] help\_options=("help",))  
*Directly extract options from a command line into a dict.*

### Variables

- string [\\_\\_author\\_\\_](#) = "Marchal Florent"
- list [\\_\\_credits\\_\\_](#) = ["Marchal Florent"]

#### 6.4.1 Function Documentation

## 6.4.1.1 getopts()

```
dict[str, any] or int scripts.getopts_parser.getopts_parser.getopts (
    list[str] or str argv,
    dict[str or tuple, None or tuple[any, any] or any] getopts_options,
    *str mandatory,
    str help_message = None,
    fill_with_default_values = True,
    raise_errors = False,
    str or tuple[str] help_options = ("help", ) )
```

Directly extract options from a command line into a dict.

All option's values are cast according to a dictionary @getopts\_options which specifies option's short name (-h), option's long names (--help), option's aliases (--HELP), option's type (str, int ...) and option's default value.

This function use getopt.getopt.

You can find some usage example in [./test\\_getopts\\_parser.py](#)

## Parameters

<i>argv</i>	<p>: list[str] or str =&gt;</p> <ul style="list-style-type: none"> <li>• A command line e.g. : "Argument1 Argument2 --Option1 value --Option2 -abc"</li> <li>• list of string (such as sys.argv[1:]) e.g. : ["Argument1", "Argument2", "--Option1", "value", "--Option2", "-abc"]</li> </ul>
<i>getopts_options</i>	<p>: dict[str, tuple[any, any]] =&gt;</p> <ul style="list-style-type: none"> <li>• Structure for this dict :  <pre>{name: None} {tuple_of_name: None} {name: (short_name, None)} {tuple_of_name: (tuple_of_short_name, None)} {name: (tuple_of_short_name, default_value)} {name: (short_name, default_value)} {tuple_of_name: (tuple_of_short_name, default_value)} {tuple_of_name: (short_name, default_value)} {name: (tuple_of_short_name, (default_value, cast))} {name: (short_name, (default_value, cast))} {tuple_of_name: (tuple_of_short_name, (default_value, cast))} {tuple_of_name: (short_name, (default_value, cast))}</pre> </li> <li>• Option that accept values should have a "=" at the end of each name and a "." at the end of each short_name.</li> <li>• Option that require a value : (e.g. -file path) <ul style="list-style-type: none"> <li>– default_value can be of any type, if callable the result of the function is used.</li> <li>– The cast can be any function. if None, nothing happen.</li> </ul> </li> <li>• Option that does not require any value : (e.g. -help) : <ul style="list-style-type: none"> <li>– The default value is False</li> <li>– The "on" value is True</li> <li>– If you specify a "cast" it will be used as the "on" value</li> </ul> </li> </ul>

**Note**

When an option has multiple long names, the first one in the tuple is selected as the "main\_name". The "main\_name" is the key used inside the returned dict.

short\_names can be composed by only one character (plus ":" if needed)

**Warning**

do not use "-" in your options names

**Parameters**

<i>*mandatory</i>	: str => a list of options whose value must be entered.
<i>help_message</i>	: str = None => A message displayed when an error is encountered or when "help" is triggered.
<i>fill_with_default_values</i>	= True => When True, the returned dict contains all options
<i>raise_errors</i>	= False => Do caught errors are raised ?
<i>help_options</i>	: str or tuple[str] = ("help", ) => A tuple of option name that trigger help_message. Those options are always removed from the returned dictionary.

**Returns**

dict[str, any] or int =>

- dict[str, any] A dictionary that contain values related to options triggered by the command-line.
- Only when an error occur and `raise_errors` is False or when help message is displayed.
  - 1 = An error has been caught
  - 2 = help message was required.

**Example :**

```
options_ = {
    'Alpha': ("a:", (None, int)),
    'Beta': ("b:", (None, list)),
    'Gamma': ("g", (True, False)),
    'Delta': None,
    'Epsilon': (None, ("Star", None)),
    'Zeta': ("Z", None),
    'Eta': None,
    'Theta': None,
    'Iota': None,
    'Kappa': None,
    'Lambda': None,
    'Mu': None,
}
val = getopt("1 --Eta --Iota --Beta Test2 -z", options_, fill_with_default_values=True)
# We have :
# val["Alpha"] == 1
# val["Beta"] == ["T", "e", "s", "t", "2"]
# val["Gamma"] is True
# val["Epsilon"] == "Star"
# val["Zeta"] is True
# val["Mu"] is False
```

**6.4.1.2 getopt\_parser.getopts\_parser.getopts\_digest\_available\_options()**

tuple[dict[str, tuple[any, any]], dict[str, str], dict[str, str], str, list[str]] scripts.↔  
 getopt\_parser.getopts\_parser.getopts\_digest\_available\_options (



```
dict[str, tuple[any, any]] getopts_options,
dict[str, any] dict_of_default_value = None )
```

Transform `getopts_options` into usable instruction for [getopts\\_retrieve\\_options](#) and assure that option's aliases are coherent.

```
@param getopts_options : dict[str, tuple[any, any]] => A dictionary that follows one of the following structure
@code
{name: None}
{tuple_of_name: None}
{name: (short_name, None)}
{tuple_of_name: (tuple_of_short_name, None)}

{name: (tuple_of_short_name, default_value)}
{name: (short_name, default_value)}
{tuple_of_name: (tuple_of_short_name, default_value)}
{tuple_of_name: (short_name, default_value)}

{name: (tuple_of_short_name, (default_value, cast))}
{name: (short_name, (default_value, cast))}
{tuple_of_name: (tuple_of_short_name, (default_value, cast))}
{tuple_of_name: (short_name, (default_value, cast))}
@endcode
@param dict_of_default_value : dict[str, any] = None => Facultative, A dictionary that will be filled using optional
default values.

@return tuple[dict[str, tuple[any, any]], dict[str, str], dict[str, str], str, list[str]] =>
- option_dict : dict[str, tuple[any, any]] => Contain each option's "main_name" associated with their default
  cast options. The "main_name" is the named that will be used into the dictionary
  returned by @ref getopts_parser.
- boolean_keys : dict[str, str] => Contain all options aliases related to options that can handle only two states
- complex_keys : dict[str, str] => Contain all options aliases related to options that can handle more than two states
- short_string : str => string that correspond to "shortopts" in @ref getopt.getopt
- long_list : list[str] => list of string that correspond to "longopts" in @ref getopt.getopt
```

#### 6.4.1.3 getopts\_digester\_check\_item\_endings()

```
bool or None scripts.getopts_parser.getopts_parser.getopts_digester_check_item_endings (
    list[str] list_of_values,
    str last_char = ":" )
```

Check if the last char of each string in `list_of_values` is equal to `last_char`.

if some are equals and others are not, an error is raised.

##### Parameters

<i>list_of_values</i>	: list[str] => A list of string
<i>last_char</i>	: str = ":" => A character

##### Returns

bool or None =>

- True : All items in `list_of_values` end by `last_char`
- False : No item in `list_of_values` end by `last_char`

- None : `list_of_values` is empty
- ValueError : Some value(s) end by `last_char` but other(s) do(es) not."

#### 6.4.1.4 `getopts_parser()`

```
dict scripts.getopts_parser.getopts_parser.getopts_parser (
    list[str] or str argv,
    dict[str, tuple[any, any]] getopts_options,
    *str mandatory,
    fill_with_default_values = True )
```

Internal version of [getopts](#).

You can use this function if you don't need [getopts](#) ' overcoat. (help message display and error handling)

This docstring is a lighten version of [getopts](#) 's docstring.

See also

[getopts](#)

##### Parameters

<i>argv</i>	: list[str] or str => List of argument (strings). Usually <code>sys.argv[1:]</code> or str command line.
<i>getopts_options</i>	: dict[str, tuple[any, any]] => A dictionary that contain options.
<i>*mandatory</i>	: str =>a list of options whose value must be entered
<i>fill_with_default_values</i>	= True => Every unused options are added to the final dict using defaults values

##### Returns

dict => A dictionary that contain options.

#### 6.4.1.5 `getopts_parser_boolean_option()`

```
any scripts.getopts_parser.getopts_parser.getopts_parser_boolean_option (
    tuple[any, any] value_restriction = None,
    use_default = False )
```

Internal function used by [getopts\\_parser](#).

##### Parameters

<i>value_restriction</i>	: tuple[any, any] = None => A tuple of two value. (Default value, normal value). (False, True) when None
<i>use_default</i>	= False => Do the default value is returned

## Returns

any =>

- True if value\_restriction is None
- Default value if use\_default is True (value\_restriction [1])
- Normal value if use\_default is False (value\_restriction [0])

## See also

[getopts\\_parser](#)

## 6.4.1.6 getopts\_parser\_complex\_option()

```
any scripts.getopts_parser.getopts_parser.getopts_parser_complex_option (
    value_restriction,
    value,
    use_default = False )
```

Internal function used by [getopts\\_parser](#).

Apply a number of restriction materialised by value\_restriction on a value.

```
@param value_restriction : tuple[any, callable] => A tuple of two value :
- Default value => Any, if callable, the function is called each time the default value is requested.
- restrictions => None or a callable.
  (False, True) when None
  - if value_restriction[0] is callable AND default value is requested : value_restriction [0] ()
  - if value_restriction[0] is not callable and default value is requested : value_restriction [0]
  - if value_restriction[1] is callable : value_restriction is called using @p value as argument, the result
    used as return value. If the result is None, directly return the @p value.
@param value => Value on which @p value_restriction is applied.
@param use_default = False => If True, return the default value.

@return any => @p value transformed by @p value_restriction [1]

@see getopts_parser
```

## 6.4.1.7 getopts\_retrieve\_options()

```
list[tuple[str, str]] scripts.getopts_parser.getopts_parser.getopts_retrieve_options (
    list[str] argv,
    str short_string,
    list[str] long_list,
    list[str] main_options )
```

Use a list of string from a command-line (e.g.

sys.argv[1:]) to extract selected options and theirs associated values.

**Parameters**

<i>argv</i>	: list[str] => A list of string (Same format as sys.argv[1:])
<i>short_string</i>	: str => string that correspond to shortopts in getopt.getopt
<i>long_list</i>	: list[str] => list of string that correspond to longopts in getopt.getopt
<i>main_options</i>	: list[str] => An ordered list of all options. Is used to determine the correspondence between arguments and parameters.

**Returns**

list[Tuple[str, str]] => A list composed of tuple that contain all selected options and associated with their values.

**6.4.2 Variable Documentation****6.4.2.1 `__author__`**

```
string scripts.getopts_parser.getopts_parser.__author__ = "Marchal Florent" [private]
```

**6.4.2.2 `__credits__`**

```
list scripts.getopts_parser.getopts_parser.__credits__ = ["Marchal Florent"] [private]
```

**6.5 `scripts.getopts_parser.test_getopts_parser` Namespace Reference****Functions**

- def [test\\_simple\\_option](#) ()  
*Test simple functionality of [getopts\\_parser](#).*
- def [test\\_aliases](#) ()  
*Test aliases functionality.*
- def [test\\_defaults\\_values](#) ()  
*Test default value and cast.*
- def [test\\_all](#) ()  
*Call each test functions.*

**Variables**

- string [\\_\\_author\\_\\_](#) = "Marchal Florent"
- list [\\_\\_credits\\_\\_](#) = ["Marchal Florent"]

## 6.5.1 Function Documentation

### 6.5.1.1 test\_aliases()

```
def scripts.getopts_parser.test_getopts_parser.test_aliases ( )
```

Test aliases functionality.

### 6.5.1.2 test\_all()

```
def scripts.getopts_parser.test_getopts_parser.test_all ( )
```

Call each test functions.

### 6.5.1.3 test\_defaults\_values()

```
def scripts.getopts_parser.test_getopts_parser.test_defaults_values ( )
```

Test default value and cast.

### 6.5.1.4 test\_simple\_option()

```
def scripts.getopts_parser.test_getopts_parser.test_simple_option ( )
```

Test simple functionality of [getopts\\_parser](#).

## 6.5.2 Variable Documentation

### 6.5.2.1 \_\_author\_\_

```
string scripts.getopts_parser.test_getopts_parser.__author__ = "Marchal Florent" [private]
```

### 6.5.2.2 `__credits__`

```
list scripts.getopts_parser.test_getopts_parser.__credits__ = ["Marchal Florent"] [private]
```

## 6.6 `scripts.snp_analyser` Namespace Reference

### Functions

- def [help\\_usage](#) ()
- bool [filter\\_integer\\_greater\\_or\\_equal\\_to\\_0\\_ignore\\_0](#) (str or int key, dict dictionary=None)  
*Test if the value in front of the key key inside dictionary can be cast into an integer bigger than 0.*
- dict[int, dict[str, int]] [compile\\_gene\\_snp](#) (iter genes\_snp, dict[int, dict[str, int]] dict\_of\_number=None, str group="None")  
*Extract the number of snp of all genes contained in genes\_snp (snp = genes\_snp's values).*
- (list[list[int]], list[int]) [make\\_data\\_matrix](#) (dict[int, dict[str, int]] compiled\_dict, str group, \*str groups, bool simplified=True, int max\_length=None)  
*Use a compiled dict from [compile\\_gene\\_snp](#) to create a matrix of value.*
- list[int] [generate\\_cumulative\\_list](#) (list[int] or list[float] list\_of\_numbers, reversed\_=False)  
*Take a list of number and sum all values.*
- int [main](#) (str path, str name\_column, str snp\_column, str file\_separator="\t", bool simplified=True, int max\_length=None, str output\_path="output", bool output\_warning=True, str job\_name=None, bool global\_heatmap=True, bool quantitative\_barchart=False, bool cumulative\_barchart=False, bool cumulative\_heatmap=False, bool tsv=False, bool png=False, bool show=False, bool svg=True, bool sort\_by\_name=True, bool uniform\_y=True, int show\_values=-1)  
*Create a number of chart related to snp analysis.*
- def [main\\_using\\_getopts](#) (list[str] or str argv)

### Variables

- string [\\_\\_author\\_\\_](#) = "Marchal Florent"
- list [\\_\\_credits\\_\\_](#) = ["Florent Marchal", "Vetea Jacot", "Concetta Burgarella", "Vincent Ranwez", "Nathalie Chantret"]
- dictionary [\\_\\_getopts\\_\\_](#)

### 6.6.1 Function Documentation

#### 6.6.1.1 `compile_gene_snp()`

```
dict[int, dict[str, int]] scripts.snp_analyser.compile_gene_snp (
    iter genes_snp,
    dict[int, dict[str, int]] dict_of_number = None,
    str group = "None" )
```

Extract the number of snp of all genes contained in genes\_snp (snp = genes\_snp's values).

Each number of snp is stored inside a new dictionary (dict\_of\_number's keys). A dict is created in front of all keys (i.e. snp number). This dict contain the group (key) and the number of occurrences of this snp number for this key.

## Parameters

<i>genes_snp</i>	: iterable => A generator from <code>extract_data_from_table</code> . e.g. <code>(number_of_snp_in_gene_1, ) =&gt; (3, 5, 3)</code>
------------------	---

## Note

Values (number of snp) inside this dict are trans typed into integers.

## Parameters

<i>dict_of_number</i>	: dict[int, dict[str,int]] = None. A dict with the same structure as dictionaries returned by this function.
<i>group</i>	: str = "None" => Each occurrence of a number of snp increment the counter related to this group.

## Returns

dict[int, dict[str, int]] => A dictionary that store all number of snp found along with the number of occurrences  
`{number_of_snp_1 : {group1: number_of_occurrences_of_number_of_snp_1_in_this_group}}`

## Warning

values `genes_snp` are cast into integer. Also, there is no verification made to see if the values are positive. We assume that data has been filtered using [filter\\_integer\\_greater\\_or\\_equal\\_to\\_0\\_ignore\\_0](#) in `extract_data_from_table`

## 6.6.1.2 filter\_integer\_greater\_or\_equal\_to\_0\_ignore\_0()

```
bool scripts.snp_analyser.filter_integer_greater_or_equal_to_0_ignore_0 (
    str or int key,
    dict dictionary = None )
```

Test if the value in front of the key `key` inside `dictionary` can be cast into an integer bigger than 0.

Meant to be used inside `extract_data_from_table` as a "filter\_" using a lambda.

- value == 0 : Line is ignored
- value > 0 : Line is kept
- Value < 0 : Error is raised

## Warning

does not support "1.0 nor "1,0" format

**Parameters**

<i>key</i>	: str => A key contained by dictionary.
<i>dictionary</i>	: dict = None => A dictionary that contain <i>key</i> . If None, the function will assume that <code>dictionary = {key: key}</code>

**Returns**

bool => True : Yes False : value equal to 0 @raise ValueError when the value is an integer lower than 0

**6.6.1.3 generate\_cumulative\_list()**

```
list[int] scripts.snp_analyser.generate_cumulative_list (
    list[int] or list[float] list_of_numbers,
    reversed_ = False )
```

Take a list of number and sum all values.

```
[0, 5, 6, 1] => [0+5+6+1, 5+6+1, 6+1, 1] == [12, 12, 7, 1]
```

**Parameters**

<i>list_of_numbers</i>	: list[int] or list[float] => A list that contain numbers.
<i>reversed_</i>	= False => Do the accumulation start at the end and end at the beginning.

**Returns**

list[int] => A list of number

**6.6.1.4 help\_usage()**

```
def scripts.snp_analyser.help_usage ( )
```

**6.6.1.5 main()**

```
int scripts.snp_analyser.main (
    str path,
    str name_column,
    str snp_column,
    str file_separator = "\t",
    bool simplified = True,
    int max_length = None,
    str output_path = "output",
```



```

bool  output_warning = True,
str   job_name = None,
bool  global_heatmap = True,
bool  quantitative_barchart = False,
bool  cumulative_barchart = False,
bool  cumulative_heatmap = False,
bool  tsv = False,
bool  png = False,
bool  show = False,
bool  svg = True,
bool  sort_by_name = True,
bool  uniform_y = True,
int   show_values = -1 )

```

Create a number of chart related to snp analysis.

As an example we will consider the following flatFile :

GeneName	GeneID	NumberOfSnp	GeneSize
Gene1	123	5	1000
Gene2	456	10	2000

#### Parameters

<i>path</i>	: str => Path that lead to a number of flatfile : <ul style="list-style-type: none"> <li>• .json : {complete file path : Species name}</li> <li>• folder : Use file inside the folder (does not scan the folder recursively)</li> </ul>
<i>name_column</i>	: str => Name of the column that contain a primary key e.g. GeneName, GeneID. If two line have the same "primary key", the last one will be used.
<i>snp_column</i>	: str => Name of the column that contain a count of snp e.g. NumberOfSnp
<i>file_separator</i>	: str = "\t" => The separator used in all flat file considered.
<i>simplified</i>	: bool = True => Do number of snp represented by 0 gene are deleted from the result
<i>max_length</i>	: int = None => Maximum length of each graph. keep the nth first result. Should be greater or equal to 1 otherwise, it would be ignored.
<i>output_path</i>	: str = "output" => Where graphs are saved.
<i>output_warning</i>	: bool = True => Ask confirmation when at least one file can be erased by this program.
<i>job_name</i>	: str = None => A name for this execution. (This creates a separated folder in output_path). Default = "unnamed"
<i>global_heatmap</i>	: bool = True => Do a heatmap that is the combination of all cumulative_heatmap is created
<i>quantitative_barchart</i>	: bool = False => Do this program create a barchart of snp distribution for each file (Number of gene that have n snp)
<i>cumulative_barchart</i>	: bool = False => Do this program create a barchart of snp distribution for each file ? (Number of gene that have AT LEAST n snp)
<i>cumulative_heatmap</i>	: bool = False => Do this program create a heatmap of snp distribution for each file ? (Number of gene that have AT LEAST n snp)
<i>tsv</i>	: bool = False => Do values used for chart are saved in a flatfile (.tsv)
<i>png</i>	: bool = False => Do created charts are saved as png
<i>show</i>	: bool = False => Do created charts are saved are shown

**Warning**

Each time a chart is shown, the program stop. It will resume when the chart is closed.

**Parameters**

<i>svg</i>	: bool = True => Do created charts are saved as svg (vectorize image)
<i>sort_by_name</i>	: bool = True => Do species are sorted in lexicographic order ?
<i>uniform_y</i>	: bool = True => Do all barchart share the same y-axis ?
<i>show_values</i>	: int = None => If greater or equal to 0, all cells will contain theirs values. if lower than 0, text in cell in automatically determined (can be ugly when show is True, but assure that the text is good in png and svg). If None, nothing happen.

**Returns**

int => if greater than 0, an error occurred.

- 1 job stopped by user
- 2 no species found

**6.6.1.6 main\_using\_getopts()**

```
def scripts.snp_analyser.main_using_getopts (
    list[str] or str argv )
```

@brief start @ref snp\_charts.main using a string or sys.argv[1:]

Example :

```
- main_using_getopts(sys.argv[1:])
- main_using_getopts("name_column snp_column tests/TargetedFiles.json -m 20 -gv -w -j Tests -e -1")
```

@param argv : list[str] or str => List of argument (strings). Usually sys.argv[1:].

@note If the first values are not known option (option in @p getopts\_options), the function will consider that those values correspond to the nth first option that require an argument in @p getopts\_options.

In below example, you can give two arguments. The first will be matched with "Alpha" and the last with "Beta".

@code

```
argv = ["5", "6"]
getopts_options = {
    "Alpha=": ("a:", None),
    "Beta=": ('b', (0, int)),
    "Gamma=": ("g", ("data/", str)),
    "Delta=": ("d:", (0, lambda integer : int(integer) - 1)),
}
```

@endcode

@return int => exit code

## 6.6.1.7 make\_data\_matrix()

```
(list[list[int]], list[int]) scripts.snp_analyser.make_data_matrix (
    dict[int, dict[str, int]] compiled_dict,
    str group,
    *str groups,
    bool simplified = True,
    int max_length = None )
```

Use a compiled dict from [compile\\_gene\\_snp](#) to create a matrix of value.

Each lines represent one group (groups & group). Each line contain the number of genes that contain at least n genes

## Note

Only groups specified in groups and group are extracted from compiled\_dict

## Note

For below example we will consider that :

```
• compiled_dict = {1: {"E. coli": 5},
                   2: {"E. coli": 3, "HIV": 4},
                   3: {"HIV": 2},
                   4: {"E. coli": 1, "HIV": 1}
                   }
```

## Parameters

<i>compiled_dict</i>	: dict[int,dict[str,int]] => a compiled dict from <a href="#">compile_gene_snp</a>
<i>group</i>	: str => A group name (e.g. Species name : "E. coli")
<i>*groups</i>	: str => Same as group. Additional species name.
<i>simplified</i>	: bool = True => Do number of snp represented by 0 gene are deleted from the result ?

## Note

This the presence / absence of a number is affected by other groups

- with group="E. coli":
  - If True:
 

```
(([5, 3, 1]], [1, 2, 4])
```
  - If False:
 

```
(([5, 3, 0, 1]], [1, 2, 3, 4])
```
- with group="E. coli" and groups= ("HIV", ):
  - If True:
 

```
(([5, 3, 0, 1], [0, 4, 2, 1]], [1, 2, 4])
```

## Parameters

<i>max_length</i>	: int = None => Limit the length of each lines of the matrix. should be greater than 0. If not, max_length is ignored.
-------------------	--

### Returns

(list[list[int]], list[int]) => Return a matrix and a list. Each lines of the matrix represent the number of genes of a species (group) that contain n snp : If the position 1 of a line is equal to 4, there is 4 genes in the selected species that contain list[i] snp. The y axes can be labeled using `group` and `groups` (order in conserved) and the x-axis is labeled by the list[int].

### Note

For some return example, see `simplified` for return example

## 6.6.2 Variable Documentation

### 6.6.2.1 `__author__`

```
string scripts.snp_analyser.__author__ = "Marchal Florent" [private]
```

### 6.6.2.2 `__credits__`

```
list scripts.snp_analyser.__credits__ = ["Florent Marchal", "Vetea Jacot", "Concetta Burgarella",  
"Vincent Ranwez", "Nathalie Chantret"] [private]
```

### 6.6.2.3 `__getopts__`

```
dictionary scripts.snp_analyser.__getopts__ [private]
```

## 6.7 `scripts.test_snp_analyser` Namespace Reference

### Functions

- def [test\\_greater\\_than\\_0\\_int\\_filter](#) ()
- def [test\\_compile\\_gene\\_snp](#) ()
- def [test\\_make\\_data\\_matrix](#) ()
- def [test\\_generate\\_cumulative\\_list](#) ()
- def [test\\_all](#) ()

### 6.7.1 Function Documentation

#### 6.7.1.1 test\_all()

```
def scripts.test_snp_analyser.test_all ( )
```

```
@brief Call all test functions
```

#### 6.7.1.2 test\_compile\_gene\_snp()

```
def scripts.test_snp_analyser.test_compile_gene_snp ( )
```

```
@brief Test compile_gene_snp
```

#### 6.7.1.3 test\_generate\_cumulative\_list()

```
def scripts.test_snp_analyser.test_generate_cumulative_list ( )
```

#### 6.7.1.4 test\_greater\_than\_0\_int\_filter()

```
def scripts.test_snp_analyser.test_greater_than_0_int_filter ( )
```

```
@brief Test greater_than_0_int_filter
```

#### 6.7.1.5 test\_make\_data\_matrix()

```
def scripts.test_snp_analyser.test_make_data_matrix ( )
```

```
@brief Test make_data_matrix
```

## 6.8 scripts.utilities Namespace Reference

### Namespaces

- [utilities](#)

## 6.9 scripts.utilities.utilities Namespace Reference

### Classes

- class [FilterError](#)

### Functions

- def [export\\_list\\_in\\_tsv\\_as\\_rows](#) (str path, \*rows, file\_mode="w", encoding="UTF-8", list y\_legend=None, list x\_legend=None)  
*Accept a number of list that represent rows of a tab and turn it into a tsv (flat file).*
- def [chart\\_export](#) (list[list[int]] data, bool show=False, str png=None, str tsv=None, str svg=None, list x\_↵ legend=None, list y\_legend=None)  
*Export the current chart.*
- dict[str, str] [parse\\_line](#) (list[str] legend, str line, str separator="\t")  
*Turn a line form a flat File with its legend and turn it into a dictionary.*
- dict[str, any] [extract\\_data\\_from\\_table](#) (str path, str key, str value, str separator="\t", list legend=None, callable filter\_=None)  
*Read a table contained inside a flatFile (e.g.*
- def [make\\_bar\\_char](#) (list[int] data, list x\_legend=None, bool x\_legend\_is\_int=True, bool y\_legend\_is\_int=True, str chart\_name=None, str title=None, str xlabel=None, str ylabel=None, bool show=False, str png=None, str tsv=None, str svg=None, bool erase\_last\_plt=True, int y\_max\_value=None)  
*Create a plt.bar using a bunch of argument.*
- def [make\\_heatmap](#) (list[list[int]] data, list x\_legend=None, list y\_legend=None, str title=None, str xlabel=None, str ylabel=None, bool show=False, str png=None, str tsv=None, str svg=None, bool erase\_last\_plt=True, int contain\_number=None, str uniq\_color=None, str cmap="jet", int y\_max\_value=None)  
*Create a heatmap using a bunch of argument.*

### 6.9.1 Function Documentation

#### 6.9.1.1 chart\_export()

```
def scripts.utilities.utilities.chart_export (
    list[list[int]] data,
    bool show = False,
    str png = None,
    str tsv = None,
    str svg = None,
    list x_legend = None,
    list y_legend = None )
```

Export the current chart.

#### Note

if data is the only argument, nothing will happen.

## Parameters

<i>data</i>	: list[list[int]] => A matrix of values
<i>show</i>	: bool = False => Do current plot will be displayed in a pop-up ?

## Warning

this will stop program execution until the pop-up is closed.

## Parameters

<i>png</i>	: str = None => Give a path to export the current plot as png
<i>tsv</i>	: str = None => Give a path to export <i>data</i> into a tsv.
<i>svg</i>	: str = None => Give a path to export the current plot as svg
<i>y_legend</i>	: list = None => When <i>tsv</i> is not none: A list of item to be display in the first column ( <a href="#">export_list_in_tsv_as_rows</a> )
<i>x_legend</i>	: list = None => When <i>tsv</i> is not none: A list of item to be display in the first line ( <a href="#">export_list_in_tsv_as_rows</a> )

## 6.9.1.2 export\_list\_in\_tsv\_as\_rows()

```
def scripts.utilities.utilities.export_list_in_tsv_as_rows (
    str path,
    * rows,
    file_mode = "w",
    encoding = "UTF-8",
    list y_legend = None,
    list x_legend = None )
```

Accept a number of list that represent rows of a tab and turn it into a tsv (flat file).

## Warning

Any "\t" or "\n" in rows' values will disrupt lines and / or columns

- each "\t" will generate additional columns
- each "\n" will break the current line and start a new one.

## Parameters

<i>path</i>	: str => path (and name) of the file that will be written.
<i>*rows</i>	=> A number of list
<i>file_mode</i>	= "w" => "w" or "a" <ul style="list-style-type: none"> <li>• "w": if a file with the same path exist, old file is erased</li> <li>• "a": if a file with the same path exist, old file append new values</li> </ul>
<i>encoding</i>	= "UTF-8" => File encoding
<i>y_legend</i>	: list = None => A list of item to be display in the first column
<i>x_legend</i>	: list = None => A list of item to be display in the first line

### 6.9.1.3 extract\_data\_from\_table()

```
dict[str, any] scripts.utilities.utilities.extract_data_from_table (
    str path,
    str key,
    str value,
    str separator = "\t",
    list legend = None,
    callable filter_ = None )
```

Read a table contained inside a flatFile (e.g.

tsv, csv, ...)

#### Warning

If the column `key` contains the same value multiple times, only the last one is kept.

#### Parameters

<i>path</i>	: Path to a flatFile.
<i>key</i>	: A column name that can be found in the legend. This will be used as a key in the returned dict. Example: "Column3"
<i>value</i>	: A column name that can be found in the legend. This will be used as a value in the returned dict WHEN <i>filter</i> returns None or True. Example: "Column2"
<i>separator</i>	: The symbol that splits line's values. Example: " ", "\n", "\t" ...
<i>legend</i>	: If None: The first non-empty line in the file split using <i>separator</i> . Else: A list of column names. Example: [Column1, Column2, Column3]
<i>filter_</i>	: A function that accepts 3 arguments: <i>key</i> , <i>value</i> , and the parsed line (dict). It selects/generates the value present next to each key. <ul style="list-style-type: none"> <li>• If it returns True or None: value in the column <i>value</i>.</li> <li>• If it returns False: this line is ignored.</li> <li>• Else: The returned value is used (instead of the content of the column <i>value</i>).</li> </ul>

#### Note

*filter\_* is called one time per line.

#### Returns

A generator: (values in the column *key* (values that do not pass *filter\_* are ignored), values in the column *value* OR value returned by *filter\_*)



#### 6.9.1.4 make\_bar\_char()

```
def scripts.utilities.utilities.make_bar_char (
    list[int] data,
    list x_legend = None,
    bool x_legend_is_int = True,
    bool y_legend_is_int = True,
    str chart_name = None,
    str title = None,
    str xlabel = None,
    str ylabel = None,
    bool show = False,
    str png = None,
    str tsv = None,
    str svg = None,
    bool erase_last_plt = True,
    int y_max_value = None )
```

Create a plt.bar using a bunch of argument.

This function is made to assure a correct looking legend when used for snp.

##### Parameters

<i>data</i>	: list[int] => A list of integer
<i>x_legend</i>	: list = None => Values used to legend the x-axis.
<i>x_legend_is_int</i>	: bool = True => Do x-axis represent oly integer
<i>y_legend_is_int</i>	: bool = True => Do y-axis represent oly integer
<i>chart_name</i>	: str = None => A name that will be used if tsv is not None to name a line.
<i>title</i>	: str = None => A title for this chart
<i>xlabel</i>	: str = None => A title for the x-axis
<i>ylabel</i>	: str = None => A title for the y-axis
<i>show</i>	: bool = False => Do current plot will be displayed ?

##### Warning

this will stop program execution until the pop-up is closed.

##### Parameters

<i>png</i>	: str = None => Give a path to export the current plot as png
<i>tsv</i>	: str = None => Give a path to export <i>data</i> into a tsv.
<i>svg</i>	: str = None => Give a path to export the current plot as svg
<i>y_max_value</i>	: int = None => The y-axis will stop at this value
<i>erase_last_plt</i>	: bool = True => If True, last plot is removed from matplotlib.pyplot display

#### 6.9.1.5 make\_heatmap()

```
def scripts.utilities.utilities.make_heatmap (
    list[list[int]] data,
```

```

list  x_legend = None,
list  y_legend = None,
str   title = None,
str   xlabel = None,
str   ylabel = None,
bool  show = False,
str   png = None,
str   tsv = None,
str   svg = None,
bool  erase_last_plt = True,
int   contain_number = None,
str   uniq_color = None,
str   cmap = "jet",
int   y_max_value = None )

```

Create a heatmap using a bunch of argument.

This function is made to assure a correct looking legend when used for snp.

#### Parameters

<i>data</i>	: list[int] => A list of integer
<i>x_legend</i>	: list = None => Values used to label the x-axis.
<i>y_legend</i>	: list = None => Values used to label the y-axis.
<i>title</i>	: str = None => A title for this chart
<i>xlabel</i>	: str = None => A title for the x-axis
<i>ylabel</i>	: str = None => A title for the y-axis
<i>show</i>	: bool = False => Do current plot will be displayed ?

#### Warning

this will stop program execution until the pop-up is closed.

#### Parameters

<i>png</i>	: str = None => Give a path to export the current plot as png
<i>tsv</i>	: str = None => Give a path to export <i>data</i> into a tsv.
<i>svg</i>	: str = None => Give a path to export the current plot as svg
<i>erase_last_plt</i>	: bool = True => If True, last plot is removed from matplotlib.pyplot memory
<i>contain_number</i>	: int = None => If greater or equal to 0, all cells will contain theirs values. if lower than 0, text in cell in automatically determined. If None, nothing happen.
<i>uniq_color</i>	: str = #a0a0a0 => HTML color code for text inside cells

#### Note

Only when *contain\_number* is True

#### Parameters

<i>y_max_value</i>	: int = None => The y-axis will stop at this value
--------------------	--

## Parameters

<i>cmap</i>	: str = jet => Color mod. supported values are 'Accent', 'Accent_r', 'Blues', 'Blues_r', 'BrBG', 'BrBG_r', 'BuGn', 'BuGn_r', 'BuPu', 'BuPu_r', 'CMRmap', 'CMRmap_r', 'Dark2', 'Dark2_r', 'GnBu', 'GnBu_r', 'Greys', 'Greens', 'Greens_r', 'Greys', 'Greys_r', 'OrRd', 'OrRd_r', 'Oranges', 'Oranges_r', 'PRGn', 'PRGn_r', 'Paired', 'Paired_r', 'Pastel1', 'Pastel1_r', 'Pastel2', 'Pastel2_r', 'PiYG', 'PiYG_r', 'PuBu', 'PuBuGn', 'PuBuGn_r', 'PuBu_r', 'PuOr', 'PuOr_r', 'PuRd', 'PuRd_r', 'Purples', 'Purples_r', 'RdBu', 'RdBu_r', 'RdGy', 'RdGy_r', 'RdPu', 'RdPu_r', 'RdYlBu', 'RdYlBu_r', 'RdYlGn', 'RdYlGn_r', 'Reds', 'Reds_r', 'Set1', 'Set1_r', 'Set2', 'Set2_r', 'Set3', 'Set3_r', 'Spectral', 'Spectral_r', 'Wistia', 'Wistia_r', 'YlGn', 'YlGnBu', 'YlGnBu_r', 'YlGn_r', 'YlOrBr', 'YlOrBr_r', 'YlOrRd', 'YlOrRd_r', 'afmhot', 'afmhot_r', 'autumn', 'autumn_r', 'binary', 'binary_r', 'bone', 'bone_r', 'brg', 'brg_r', 'bwr', 'bwr_r', 'cividis', 'cividis_r', 'cool', 'cool_r', 'coolwarm', 'coolwarm_r', 'copper', 'copper_r', 'cubehelix', 'cubehelix_r', 'flag', 'flag_r', 'gist_earth', 'gist_earth_r', 'gist_gray', 'gist_gray_r', 'gist_grey', 'gist_heat', 'gist_heat_r', 'gist_ncar', 'gist_ncar_r', 'gist_rainbow', 'gist_rainbow_r', 'gist_stern', 'gist_stern_r', 'gist_yarg', 'gist_yarg_r', 'gist_yerg', 'gnuplot', 'gnuplot2', 'gnuplot2_r', 'gnuplot_r', 'gray', 'gray_r', 'grey', 'hot', 'hot_r', 'hsv', 'hsv_r', 'inferno', 'inferno_r', 'jet', 'jet_r', 'magma', 'magma_r', 'nipy_spectral', 'nipy_spectral_r', 'ocean', 'ocean_r', 'pink', 'pink_r', 'plasma', 'plasma_r', 'prism', 'prism_r', 'rainbow', 'rainbow_r', 'seismic', 'seismic_r', 'spring', 'spring_r', 'summer', 'summer_r', 'tab10', 'tab10_r', 'tab20', 'tab20_r', 'tab20b', 'tab20b_r', 'tab20c', 'tab20c_r', 'terrain', 'terrain_r', 'turbo', 'turbo_r', 'twilight', 'twilight_r', 'twilight_shifted', 'twilight_shifted_r', 'viridis', 'viridis_r', 'winter', 'winter_r'
-------------	--

## 6.9.1.6 parse\_line()

```
dict[str, str] scripts.utilities.utilities.parse_line (
    list[str] legend,
    str line,
    str separator = "\t" )
```

Turn a line form a flat File with its legend and turn it into a dictionary.

## Parameters

<i>legend</i>	: Names all the line's columns. Example: ["A", "B", "C"]
<i>line</i>	: Contains all the line's values. Example: "1 2 3 ", "1 2 3" ...
<i>separator</i>	: The symbol that splits the line's values. Example: " ", "\n", "\t" ...

## Returns

A dictionary composed of legend's values and line's values. Example (using previous examples):  
{"A": "1", "B": "2", "C": "3"}

## Note

The returned dict always contain the same number of object than legend.

- If legend > line : part of the legend's values will point to an empty string
- If legend < line : part of the line will be ignored

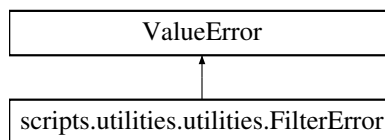


## Chapter 7

# Class Documentation

### 7.1 scripts.utilities.utilities.FilterError Class Reference

Inheritance diagram for scripts.utilities.utilities.FilterError:



#### 7.1.1 Detailed Description

An error raised by `@ref extract_data_from_table` when a `filter_` raise an error

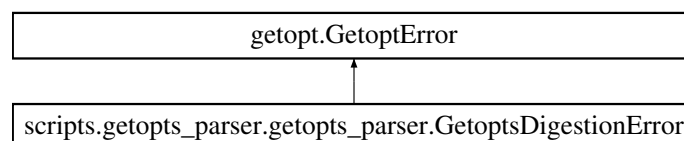
The documentation for this class was generated from the following file:

- [utilities.py](#)

### 7.2 scripts.getopts\_parser.getopts\_parser.GetoptsDigestionError Class Reference

Error raised during the digestion of the dictionary that contain options.

Inheritance diagram for scripts.getopts\_parser.getopts\_parser.GetoptsDigestionError:



### 7.2.1 Detailed Description

Error raised during the digestion of the dictionary that contain options.

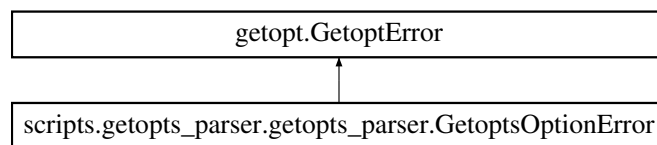
The documentation for this class was generated from the following file:

- [getopts\\_parser.py](#)

## 7.3 `scripts.getopts_parser.getopts_parser.GetoptsOptionError` Class Reference

Error raised when there is an error with an option during the getopts.

Inheritance diagram for `scripts.getopts_parser.getopts_parser.GetoptsOptionError`:



### 7.3.1 Detailed Description

Error raised when there is an error with an option during the getopts.

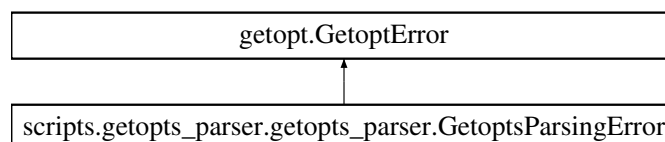
The documentation for this class was generated from the following file:

- [getopts\\_parser.py](#)

## 7.4 `scripts.getopts_parser.getopts_parser.GetoptsParsingError` Class Reference

Error raised when there is an error during the parsing of a command-line.

Inheritance diagram for `scripts.getopts_parser.getopts_parser.GetoptsParsingError`:



### 7.4.1 Detailed Description

Error raised when there is an error during the parsing of a command-line.

The documentation for this class was generated from the following file:

- [getopts\\_parser.py](#)

## Chapter 8

# File Documentation

### 8.1 `__init__.py` File Reference

#### Namespaces

- [scripts](#)

### 8.2 `__init__.py` File Reference

#### Namespaces

- [scripts.getopts\\_parser](#)

### 8.3 `__init__.py` File Reference

#### Namespaces

- [scripts.utilities](#)

### 8.4 `getopts_parser.py` File Reference

#### Classes

- class [scripts.getopts\\_parser.getopts\\_parser.GetoptsDigestionError](#)  
*Error raised during the digestion of the dictionary that contain options.*
- class [scripts.getopts\\_parser.getopts\\_parser.GetoptsOptionError](#)  
*Error raised when there is an error with an option during the getopts.*
- class [scripts.getopts\\_parser.getopts\\_parser.GetoptsParsingError](#)  
*Error raised when there is an error during the parsing of a command-line.*

## Namespaces

- [scripts.getopts\\_parser.getopts\\_parser](#)

## Functions

- any [scripts.getopts\\_parser.getopts\\_parser.getopts\\_parser\\_boolean\\_option](#) (tuple[any, any] value\_↔ restriction=None, use\_default=False)

*Internal function used by [getopts\\_parser](#).*

- any [scripts.getopts\\_parser.getopts\\_parser.getopts\\_parser\\_complex\\_option](#) (value\_restriction, value, use\_↔ default=False)

*Internal function used by [getopts\\_parser](#).*

- bool or None [scripts.getopts\\_parser.getopts\\_parser.getopts\\_digester\\_check\\_item\\_endings](#) (list[str] list\_of\_↔ values, str last\_char=":")

*Check if the last char of each string in list\_of\_values is equal to last\_char.*

- tuple[dict[str, tuple[any, any]], dict[str, str], dict[str, str], str, list[str]] [scripts.getopts\\_parser.getopts\\_parser.getopts\\_digest\\_available](#) (dict[str, tuple[any, any]] getopts\_options, dict[str, any] dict\_of\_default\_value=None)

*Transform getopts\_options into usable instruction for [getopts\\_retrieve\\_options](#) and assure that option's aliases are coherent.*

- list[tuple[str, str]] [scripts.getopts\\_parser.getopts\\_parser.getopts\\_retrieve\\_options](#) (list[str] argv, str short\_↔ string, list[str] long\_list, list[str] main\_options)

*Use a list of string from a command-line (e.g.*

- dict [scripts.getopts\\_parser.getopts\\_parser.getopts\\_parser](#) (list[str] or str argv, dict[str, tuple[any, any]] getopts\_options, \*str mandatory, fill\_with\_default\_values=True)

*Internal version of [getopts](#).*

- dict[str, any] or int [scripts.getopts\\_parser.getopts\\_parser.getopts](#) (list[str] or str argv, dict[str or tuple, None or tuple[any, any] or any] getopts\_options, \*str mandatory, str help\_message=None, fill\_with\_default\_↔ values=True, raise\_errors=False, str or tuple[str] help\_options=("help",))

*Directly extract options from a command line into a dict.*

## Variables

- string [scripts.getopts\\_parser.getopts\\_parser.\\_\\_author\\_\\_](#) = "Marchal Florent"
- list [scripts.getopts\\_parser.getopts\\_parser.\\_\\_credits\\_\\_](#) = ["Marchal Florent"]

## 8.5 main.py File Reference

### Namespaces

- [main](#)

*Run [scripts/snp\\_analyser.py](#).*

## 8.6 README.md File Reference

## 8.7 snp\_analyser.py File Reference

Create a number of chart related to snp (simple nucleotide polymorphism) analysis.



## Namespaces

- [scripts.snp\\_analyser](#)

## Functions

- def [scripts.snp\\_analyser.help\\_usage](#) ()
- bool [scripts.snp\\_analyser.filter\\_integer\\_greater\\_or\\_equal\\_to\\_0\\_ignore\\_0](#) (str or int key, dict dictionary=None)  
*Test if the value in front of the key `key` inside `dictionary` can be cast into an integer bigger than 0.*
- dict[int, dict[str, int]] [scripts.snp\\_analyser.compile\\_gene\\_snp](#) (iter genes\_snp, dict[int, dict[str, int]] dict\_of\_↵  
number=None, str group="None")  
*Extract the number of snp of all genes contained in `genes_snp` (`snp = genes_snp` 's values).*
- (list[list[int]], list[int]) [scripts.snp\\_analyser.make\\_data\\_matrix](#) (dict[int, dict[str, int]] compiled\_dict, str group,  
\*str groups, bool simplified=True, int max\_length=None)  
*Use a compiled dict from `compile_gene_snp` to create a matrix of value.*
- list[int] [scripts.snp\\_analyser.generate\\_cumulative\\_list](#) (list[int] or list[float] list\_of\_numbers, reversed\_=False)  
*Take a list of number and sum all values.*
- int [scripts.snp\\_analyser.main](#) (str path, str name\_column, str snp\_column, str file\_separator="\t", bool  
simplified=True, int max\_length=None, str output\_path="output", bool output\_warning=True, str job\_↵  
name=None, bool global\_heatmap=True, bool quantitative\_barchart=False, bool cumulative\_barchart=False,  
bool cumulative\_heatmap=False, bool tsv=False, bool png=False, bool show=False, bool svg=True, bool  
sort\_by\_name=True, bool uniform\_y=True, int show\_values=-1)  
*Create a number of chart related to snp analysis.*
- def [scripts.snp\\_analyser.main\\_using\\_getopts](#) (list[str] or str argv)

## Variables

- string [scripts.snp\\_analyser.\\_\\_author\\_\\_](#) = "Marchal Florent"
- list [scripts.snp\\_analyser.\\_\\_credits\\_\\_](#) = ["Florent Marchal", "Vetea Jacot", "Concetta Burgarella", "Vincent Ran-  
wez", "Nathalie Chantret"]
- dictionary [scripts.snp\\_analyser.\\_\\_getopts\\_\\_](#)

### 8.7.1 Detailed Description

Create a number of chart related to snp (simple nucleotide polymorphism) analysis.

#### 8.7.1.1 to use this file

Python3 [Column that contain gene's names] [Column tha contain the number of snp] [path to a folder that contain  
your files] [options]

See string returned by [snp\\_charts.help\\_usage\(\)](#) for information

#### 8.7.1.2 Charts

- Quantitative bar chart (-q) : Show the number of gene (y) per number of snp (x)
- Cumulative bar chart (-c): Show the number of gene (y) that have at least n snp (x)
- Monoheatmap (-u): Cumulative bar chart but it's a heatmap
- global heatmap (-g): concatenation of all Monoheatmap

### 8.7.1.3 Formats

- as tsv (-t)
- as png (-k)
- as svg (-b)
- as a pop-up (-d)

### 8.7.2 Author(s)

- Created by Marchal Florent on 06/05/2024. This module has been made in 2024 during an internship at the UMR Agap, GE2pop (France, Montpellier)

### 8.7.3 Librairies/Modules

- os
- json
- matplotlib.pyplot
- sys
- getopts\_parser
- utilities

## 8.8 test\_getopts\_parser.py File Reference

### Namespaces

- [scripts.getopts\\_parser.test\\_getopts\\_parser](#)

### Functions

- def [scripts.getopts\\_parser.test\\_getopts\\_parser.test\\_simple\\_option](#) ()  
*Test simple functionality of [getopts\\_parser](#).*
- def [scripts.getopts\\_parser.test\\_getopts\\_parser.test\\_aliases](#) ()  
*Test aliases functionality.*
- def [scripts.getopts\\_parser.test\\_getopts\\_parser.test\\_defaults\\_values](#) ()  
*Test default value and cast.*
- def [scripts.getopts\\_parser.test\\_getopts\\_parser.test\\_all](#) ()  
*Call each test functions.*

### Variables

- string [scripts.getopts\\_parser.test\\_getopts\\_parser.\\_\\_author\\_\\_](#) = "Marchal Florent"
- list [scripts.getopts\\_parser.test\\_getopts\\_parser.\\_\\_credits\\_\\_](#) = ["Marchal Florent"]

## 8.9 test\_snp\_analyser.py File Reference

### Namespaces

- [scripts.test\\_snp\\_analyser](#)

### Functions

- def [scripts.test\\_snp\\_analyser.test\\_greater\\_than\\_0\\_int\\_filter](#) ()
- def [scripts.test\\_snp\\_analyser.test\\_compile\\_gene\\_snp](#) ()
- def [scripts.test\\_snp\\_analyser.test\\_make\\_data\\_matrix](#) ()
- def [scripts.test\\_snp\\_analyser.test\\_generate\\_cumulative\\_list](#) ()
- def [scripts.test\\_snp\\_analyser.test\\_all](#) ()

## 8.10 utilities.py File Reference

Contain a number of function related to file reading and generation of figure using matplotlib.

### Classes

- class [scripts.utilities.utilities.FilterError](#)

### Namespaces

- [scripts.utilities.utilities](#)

### Functions

- def [scripts.utilities.utilities.export\\_list\\_in\\_tsv\\_as\\_rows](#) (str path, \*rows, file\_mode="w", encoding="UTF-8", list y\_legend=None, list x\_legend=None)  
*Accept a number of list that represent rows of a tab and turn it into a tsv (flat file).*
- def [scripts.utilities.utilities.chart\\_export](#) (list[list[int]] data, bool show=False, str png=None, str tsv=None, str svg=None, list x\_legend=None, list y\_legend=None)  
*Export the current chart.*
- dict[str, str] [scripts.utilities.utilities.parse\\_line](#) (list[str] legend, str line, str separator="\t")  
*Turn a line form a flat File with its legend and turn it into a dictionary.*
- dict[str, any] [scripts.utilities.utilities.extract\\_data\\_from\\_table](#) (str path, str key, str value, str separator="\t", list legend=None, callable filter\_=None)  
*Read a table contained inside a flatFile (e.g.*  
  - def [scripts.utilities.utilities.make\\_bar\\_char](#) (list[int] data, list x\_legend=None, bool x\_legend\_is\_int=True, bool y\_legend\_is\_int=True, str chart\_name=None, str title=None, str xlabel=None, str ylabel=None, bool show=False, str png=None, str tsv=None, str svg=None, bool erase\_last\_plt=True, int y\_max\_value=None)  
*Create a plt.bar using a bunch of argument.*
  - def [scripts.utilities.utilities.make\\_heatmap](#) (list[list[int]] data, list x\_legend=None, list y\_legend=None, str title=None, str xlabel=None, str ylabel=None, bool show=False, str png=None, str tsv=None, str svg=None, bool erase\_last\_plt=True, int contain\_number=None, str uniq\_color=None, str cmap="jet", int y\_max\_value=None)  
*Create a heatmap using a bunch of argument.*

### 8.10.1 Detailed Description

Contain a number of function related to file reading and generation of figure using matplotlib.

It's a toolbox for [scripts/snp\\_analyser.py](#)



# Index

- [\\_\\_author\\_\\_](#)
    - [scripts.getopts\\_parser.getopts\\_parser, 22](#)
    - [scripts.getopts\\_parser.test\\_getopts\\_parser, 23](#)
    - [scripts.snp\\_analyser, 30](#)
  - [\\_\\_credits\\_\\_](#)
    - [scripts.getopts\\_parser.getopts\\_parser, 22](#)
    - [scripts.getopts\\_parser.test\\_getopts\\_parser, 23](#)
    - [scripts.snp\\_analyser, 30](#)
  - [\\_\\_getopts\\_\\_](#)
    - [scripts.snp\\_analyser, 30](#)
  - [\\_\\_init\\_\\_.py, 41](#)
- [chart\\_export](#)
  - [scripts.utilities.utilities, 32](#)
- [compile\\_gene\\_snp](#)
  - [scripts.snp\\_analyser, 24](#)
- [export\\_list\\_in\\_tsv\\_as\\_rows](#)
  - [scripts.utilities.utilities, 33](#)
- [extract\\_data\\_from\\_table](#)
  - [scripts.utilities.utilities, 34](#)
- [filter\\_integer\\_greater\\_or\\_equal\\_to\\_0\\_ignore\\_0](#)
  - [scripts.snp\\_analyser, 25](#)
- [generate\\_cumulative\\_list](#)
  - [scripts.snp\\_analyser, 26](#)
- [getopts](#)
  - [scripts.getopts\\_parser.getopts\\_parser, 16](#)
- [getopts\\_digest\\_available\\_options](#)
  - [scripts.getopts\\_parser.getopts\\_parser, 18](#)
- [getopts\\_digester\\_check\\_item\\_endings](#)
  - [scripts.getopts\\_parser.getopts\\_parser, 19](#)
- [getopts\\_parser](#)
  - [scripts.getopts\\_parser.getopts\\_parser, 20](#)
- [getopts\\_parser.py, 41](#)
- [getopts\\_parser\\_boolean\\_option](#)
  - [scripts.getopts\\_parser.getopts\\_parser, 20](#)
- [getopts\\_parser\\_complex\\_option](#)
  - [scripts.getopts\\_parser.getopts\\_parser, 21](#)
- [getopts\\_retrieve\\_options](#)
  - [scripts.getopts\\_parser.getopts\\_parser, 21](#)
- [help\\_usage](#)
  - [scripts.snp\\_analyser, 26](#)
- [main, 15](#)
  - [scripts.snp\\_analyser, 26](#)
- [main.py, 42](#)
- [main\\_using\\_getopts](#)
  - [scripts.snp\\_analyser, 28](#)
- [make\\_bar\\_char](#)
  - [scripts.utilities.utilities, 34](#)
- [make\\_data\\_matrix](#)
  - [scripts.snp\\_analyser, 28](#)
- [make\\_heatmap](#)
  - [scripts.utilities.utilities, 35](#)
- [parse\\_line](#)
  - [scripts.utilities.utilities, 37](#)
- [README.md, 42](#)
- [scripts, 15](#)
- [scripts.getopts\\_parser, 15](#)
- [scripts.getopts\\_parser.getopts\\_parser, 16](#)
  - [\\_\\_author\\_\\_, 22](#)
  - [\\_\\_credits\\_\\_, 22](#)
  - [getopts, 16](#)
  - [getopts\\_digest\\_available\\_options, 18](#)
  - [getopts\\_digester\\_check\\_item\\_endings, 19](#)
  - [getopts\\_parser, 20](#)
  - [getopts\\_parser\\_boolean\\_option, 20](#)
  - [getopts\\_parser\\_complex\\_option, 21](#)
  - [getopts\\_retrieve\\_options, 21](#)
- [scripts.getopts\\_parser.getopts\\_parser.GetoptsDigestionError, 39](#)
- [scripts.getopts\\_parser.getopts\\_parser.GetoptsOptionError, 40](#)
- [scripts.getopts\\_parser.getopts\\_parser.GetoptsParsingError, 40](#)
- [scripts.getopts\\_parser.test\\_getopts\\_parser, 22](#)
  - [\\_\\_author\\_\\_, 23](#)
  - [\\_\\_credits\\_\\_, 23](#)
  - [test\\_aliases, 23](#)
  - [test\\_all, 23](#)
  - [test\\_defaults\\_values, 23](#)
  - [test\\_simple\\_option, 23](#)
- [scripts.snp\\_analyser, 24](#)
  - [\\_\\_author\\_\\_, 30](#)
  - [\\_\\_credits\\_\\_, 30](#)
  - [\\_\\_getopts\\_\\_, 30](#)
  - [compile\\_gene\\_snp, 24](#)
  - [filter\\_integer\\_greater\\_or\\_equal\\_to\\_0\\_ignore\\_0, 25](#)
  - [generate\\_cumulative\\_list, 26](#)
  - [help\\_usage, 26](#)
  - [main, 26](#)
  - [main\\_using\\_getopts, 28](#)
  - [make\\_data\\_matrix, 28](#)
- [scripts.test\\_snp\\_analyser, 30](#)
  - [test\\_all, 30](#)

- test\_compile\_gene\_snp, [31](#)
- test\_generate\_cumulative\_list, [31](#)
- test\_greater\_than\_0\_int\_filter, [31](#)
- test\_make\_data\_matrix, [31](#)
- scripts.utilities, [31](#)
- scripts.utilities.utilities, [32](#)
  - chart\_export, [32](#)
  - export\_list\_in\_tsv\_as\_rows, [33](#)
  - extract\_data\_from\_table, [34](#)
  - make\_bar\_char, [34](#)
  - make\_heatmap, [35](#)
  - parse\_line, [37](#)
- scripts.utilities.utilities.FilterError, [39](#)
- snp\_analyser.py, [42](#)
- test\_aliases
  - scripts.getopts\_parser.test\_getopts\_parser, [23](#)
- test\_all
  - scripts.getopts\_parser.test\_getopts\_parser, [23](#)
  - scripts.test\_snp\_analyser, [30](#)
- test\_compile\_gene\_snp
  - scripts.test\_snp\_analyser, [31](#)
- test\_defaults\_values
  - scripts.getopts\_parser.test\_getopts\_parser, [23](#)
- test\_generate\_cumulative\_list
  - scripts.test\_snp\_analyser, [31](#)
- test\_getopts\_parser.py, [44](#)
- test\_greater\_than\_0\_int\_filter
  - scripts.test\_snp\_analyser, [31](#)
- test\_make\_data\_matrix
  - scripts.test\_snp\_analyser, [31](#)
- test\_simple\_option
  - scripts.getopts\_parser.test\_getopts\_parser, [23](#)
- test\_snp\_analyser.py, [45](#)
- utilities.py, [45](#)