Infos

This package will receive others functions.

advs

Description

Allow to remove/keep all patterns contained in a list from another list.

Usage

```
advs(input_l, spe_char=[], exclude=True)
```

`input_l` is the input list from which the pattern has to be removed/kept

`spe_char` is the list containing all the patterns that has to be removed/kept, can also contain patterns that are not in input_l without causing an error

`exclude` if set to True, will remove the patterns and if set to False will keep the patterns in spe_char from input_l. Defaults to True.

advs_sub

Description

Behaves like `advs` but the patterns that has to be reoved/kept can be detected as sub-pattern, it means that if this pattern "oui" has to be removed from ["oui", "non", "ouinon"] for example, in advs the result will be ["non", "ouinion"], but in advs_sub the result will be ["non"].

Usage

```
advs_sub(input_l, sub_char=[], exclude=True)
```

`input_l` is the input list from which the pattern has to be removed/kept

`spe_sub` is the list containing all the sub-patterns that has to be removed/kept, can also contain sub-patterns that are not in input_l without causing an error

`exclude` if set to True, will remove the patterns containing the sub-patterns and if set to False will keep the patterns containing the sub-patterns in spe_char from input_l. Defaults to True.

file_rec

Description

Allow to get all the files or subdirectories from a directory. You can exclude names by precising it directly or precise sub-patterns contained in the folder or file names yopu want to exclude.

Usage

```
file_rec(path=".", tracker_l=[os.listdir(".")], cur_depth=0,
depth="max", rtn_l=[], type_rtn="file", excl=[], sub_excl=[],
frst_path=".")
```

As this fnction is recursive, some parameters should not be altered, on ly the alterable pattrens will be presented.

`path` is the input directory. Defaults to “.”.

`depth` is the depth of search sub-directories and files. “max” means the maximum depth. The type of value other than “max” is int. Defaults to “max”.

`type_rtn` is the type of element returned “file” or “folder”

`excl` is a list containing all the filenames or foldernames excluded in the return list

`sub_excl` is a list containing all the sub-patterns that excludes the filenames or foldernames from the return list if those are composed of these sub-patterns.

distance

Description

Allow to return the distances between multiple geographical coordinates and another geographical point.

Usage

```
distance(lat1, long1, lat_l, long_l, alt_l=None, alt1=None)
```

`lat1` is the latiitude of the established point

`long1` is the longitude of the established point

`alt1` is the altitude of the established point, if not given the distances calculated won’t take in count this parameter

`lat_l` is a list containing the latitudes of the geographical points to be compared to the established point

`long_l` is a list containing the longitudes of the geographical points to be compared to the established point

`alt_l` is a list containing the altitudes of the geographical points to be compared to the established point, if not given the distances calculated won’t take in count this parameter

Class composed of functions for list manipulation.

nestfind

Description

Allow to access to the list or the element of the n list in depth from the main list.

Usage

```
nestfind(input_l, dim_search)
```

`input_l` is the input list

`dim_search` is the dimension of the list or the element to find

Example

```
>l = [[1, 3, 2, [5, 6], 5], 4, [7, "ee"]]

>dim_search = [0, 3, 1]

>nestfind(input_l=ldim_search=dim_search)

6

>dim_search = [0, 3]

[5, 6]
```

ns

Description

Function whose goal is to manipulate nested list.

Usage

```
>ns(input_l, dim_end=1, strt_l=[], rtn_l=[], id_rec_main=0,
wrk_l=None, flag_l=[])
```

The two parameters you need to know are input_l and dim_end. The fact that it is a recursive function requires the presence of the others paramters that are used each iteration of the recursive function.

`input_l` The nested list you want to unnest to a certain point. `dim_end` The dimension from which you want to keep.

Example

```
>ns(input_l=[1, [5, [[2], 4, [23, 3, 3]]], 2, 3334, [4, [55, 56],
7, [77, [66, 67], 78], 2, [33, 5]], 3, [5, 6], 4], dim_end=3,
strt_l=[], rtn_l=[], flag_l=[])

[1, 5, [2], 4, [23, 3, 3], 2, 3334, 4, 55, 56, 7, 77, [66, 67],
78, 2, 33, 5, 3, 5, 6, 4]

>ns([1, [2], 3], dim_end=1, strt_l=[], rtn_l=[], flag_l=[])

[1, [2], 3]

>ns([1, [2], 3], dim_end=2, strt_l=[], rtn_l=[], flag_l=[])

[1, 2, 3]
```

Here, we are forced to declare the list parameters in the function call because if not declared, it will take their last value. This is the case for python 3.11.6.

inter_min

Description

Takes as input a list of lists composed of ints or floats ascendly ordered (intervals) that can have a different step to one of another element ex: `[[0, 2, 4], [0, 4], [1, 2, 2.3]]` This function will return the list of lists with the same steps preserving the begin and end value of each interval.

output from ex:

```
[[0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2,
1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2.0, 2.1, 2.2, 2 .3, 2.4, 2.5,
2.6, 2.7, 2.8, 2.9, 3.0, 3.1, 3.2, 3.3, 3.4, 3.5, 3.6, 3.7, 3.8,
3.9, 4.0], [0, 0.1, 0.2, 0.3, 0.4, 0. 5, 0.6, 0.7, 0.8, 0.9, 1.0,
1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2.0, 2.1, 2.2, 2.3,
2.4, 2.5, 2.6, 2.7, 2.8 , 2.9, 3.0, 3.1, 3.2, 3.3, 3.4, 3.5, 3.6,
```

`3.7, 3.8, 3.9, 4.0], [1, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2.0,  2.1, 2.2, 2.3]]`

The way the algorythmn searches the common step of all the sub-lists is also given by the user as a parameter, see `how_to` paramaters.

Usage

```
inter_min(inpt_l, min_=1000, sensi=3,  sensi2=3, how_to_op=deque(["divide"]),
how_to_val=deque([3]))
```

`inpt_l` is the input list containing all the intervals

`min_` is a value you are sure is superior to the maximum step value in all the intervals

`sensi` is the decimal accuracy of how the difference between each value n to n+1 in an interval is calculated

`sensi2` is the decimal accuracy of how the value with the common step is calculated in all the intervals

`how_to_op` is a deque containing the operations to perform to the pre-common step value, defaults to only "divide". The operations can be "divide", "substract", "multiply" or "add". All type of operations can be in this parameter.

`how_to_val` is a deque containing the value relatives to the operations in `hot_to_op`, defaults to 3

inter__max

Description

Takes as input a list of lists composed of ints or floats ascendly ordered (intervals) that can have a different step to one of another element ex: `[[0, 2, 4], [0, 4], [1, 2, 2.3]]` The function will return the list of lists altered according to the maximum step found in the input list.

output from ex:

`[[0, 4, 6], [0, 4, 5], [1, 2.59]]`

or, if you choose to not keep the last value…

`[[0, 4], [0, 4], [1]]`

Usage

```
def inter_max(inpt_l, max_=-1000, get_lst=True)
```

`inpt_l` is the input list of lists

`max_` is a value you are sure is the minimum step value of all the sub-lists

`get_lst` is the parameter that, if set to True, will keep the last values of sub-lists in the return value if the last step exceed the end value of the sub-list.