Index

Super-module

Functions

Classes

Module **general**

▸ EXPAND SOURCE CODE

Functions

`def AllJstreeJson(pathDict)`

Description:

Loop through the PATH_DICT and collate all jsTree json strings into single array

Args:

```
pathDict : dict
    #: The global path dict
```

Outputs:

```
allJson : [str]
    #: String array with all jsTree json strings
```

▸ EXPAND SOURCE CODE

`def BuildDataframes(tree: Tree)`

Description:

Builds dataframes that contain all File and Folder objects within the tree (separate File and Folder dataframes). The dataframes are indexed using the absolute (posix) path string. The specified dataframe columns are built from the fobject attributes using the fobject.__dict__ object. Called on tree construction once all files and folders built

Args:

```
tree : Tree
    #: The tree object containing all files and folders to parse into th
```

Outputs:

```
None
    #: Dataframes are directly set on the Tree instance passed (this met
```

▸ EXPAND SOURCE CODE

`def IndividualJstreeJson(object, isTree=False)`

Description:

Generate the jsTree JSON string for an individual fobject (including size info string). Sets required attributes in the JSONDict and dumps the result to a JSON string. Specifies icon based on input fobject type. Called on fobject construction once attributes set, string stored as fobject attribute.

Args:

```
object : File or Folder
    #: The fobject to generate JSON for
```

Kwargs:

```
isTree : bool : default=False
    #: Only true if this fobject is the absolute tree root (to set parer
```

Outputs:

```
JSONstring : str
    #: JSON string required for jsTree
```

▸ EXPAND SOURCE CODE

`def JsonFromPaths(dfIndices: list)`

▸ EXPAND SOURCE CODE

`def ListAllToplevelFiles(folders: list)`

Description:

Produces a list all files contained in the top level of all folders passed.

Args:

```
folders : list[Folder]
    #: List of folder objects to find files within (toplevel only)
```

Outputs:

```
allFiles : list[File]
    #: List of all file objects contained in all passed folders
```

▸ EXPAND SOURCE CODE

`def ListFolderContents(path, allSubdirs=True, noFiles=False, ignoreProgramFolders=True)`

Description:

Produces a list of all filenames and separate list of all subdirectory names within the folder passed.

Args:

```
path : str
    #: Pathlike string of folder to search within
```

Kwargs:

```
allSubdirs : bool : default=True
    #: Specify whether the list should penetrate into all subdirectories
noFiles : bool : default=False
    #: If true, output list of subdirectories only
ignoreProgramFolders : bool : default=True
    #: Specify whether to exclude any folders whose name matches those t
```

Outputs:

```
filenames : list[str]
    #: List of all file path strings contained in all relevant folders (
allSubfoldersList : list[str]
    #: List of all subfolder path strings (max depth)
subfolderList : list[str]
    #: If allSubdirs=False this returns toplevel subdirs only
```

▸ EXPAND SOURCE CODE

`def ListSubdirs(root)`

▸ EXPAND SOURCE CODE

`def ListSubdirsAtDepth(root, maxDepth)`

Description:

Produces a list all subdirectories at the specified depth within the root folder passed.

Args:

```
root : Pathlike str or Folder object
    #: List of folder objects to find files within (toplevel only)
maxDepth : int
    #: Furthest depth to penetrate into subdirs - 0 = toplevel, 1 = firs
```

Outputs:

```
allSubdirsList : list[Folder]
    #: List of all folder objects found within the maxDepth
```

▸ EXPAND SOURCE CODE

`def ParseFobject(object)`

Description:

Given either a pathlike string, path, or fobject this returns the associated fobject (using the global PATH_DICT). If no corresponding object exists this returns None.

Args:

```
object : str, Path, File or Folder
    #: Accepts any of the above types that may represent the desired fot
```

Outputs:

```
output : File, Folder or None
    #: The file or folder instance returned from the PATH_DICT lookup -
```

Notes:

Possible optimisation could remove this function in places where the object is guaranteed to already be of fobject type.

▸ EXPAND SOURCE CODE

`def ParsePath(path)`

Description:

Given either a pathlike string, path, or fobject this returns (or generates given a pathlike string) the associated Path object.

Args:

```
object : str, Path, File or Folder
    #: Accepts any of the above types that may represent the desired pat
```

Outputs:

```
output : Path
    #: The Path object associated with the passed object
```

Notes:

Possible optimisation could remove this function in places where the object is guaranteed to already be of Path type.

▸ EXPAND SOURCE CODE

`def SortItemValueList(itemValueList, desc=False)`

Description:

Sort a list of pairs of the form (item, value) based on the value.

Args:

```
itemValueList : list[tuple]
    #: List of pairs (item, value) to be sorted
```

Kwargs:

```
desc : bool : default=False
    #: If true, sort in descending order
```

Outputs:

```
sortedItemsList : list[tuple]
    #: The sorted list of (item, value) pairs
```

▸ EXPAND SOURCE CODE

`def WriteStorageInfo(object)`

Description:

Given a fobject this writes the string with top and net size information. Uses size from hurry.filesize to convert to readable size

Args:

```
object : File or Folder
    #: File or Folder object (sizes stored in top and net size attribute
```

Outputs:

```
info : str
    #: The string with size information
```

Notes:

File size is stored in an attribute called "netSize" to match the netSize attribute of folders - to allow methods to reference this property that allow both file and folder object inputs

▸ EXPAND SOURCE CODE

`def calculate_storage(self: Folder)`

▸ EXPAND SOURCE CODE

`def makeNewConsecutiveFolder(root, folderName)`

Description:

Make a new folder within the specified root with consecutively increasing numbering of the form folderName_#

Args:

```
root : pathlike object or fobject
    #: Folder to create new folders within
folderName : str
    #: Name of the new folders to create, eg 'run' would create folders
```

Outputs:

```
#: Path to newly created folder
```

▸ EXPAND SOURCE CODE

`def parseCSVstring(csvString: str)`

Description:

Parse a comma separated value string into a list of values.

Args:

```
csvString : str
    #: String containing comma separated values
```

Outputs:

```
splitList : list[str]
    #: List of values
```

▸ EXPAND SOURCE CODE

`def sortbySizeDesc(unsortedObjects: list, topLevel=False)`

Description:

Sort a list of fobjects in descending size order. Kwarg topLevel only relevant for folders (sort by topLevel as opposed to netSize).

Args:

```
unsortedObjects : list[File or Folder]
    #: List of fobjects to be sorted
```

Kwargs:

```
topLevel : bool : default=False
    #: If true and fobjects are Folders, sort by topLevel not netSizes
```

Outputs:

```
sortedList : list[File or Folder]
    #: The sorted (descending order) list of files
```

▸ EXPAND SOURCE CODE

`def validPath(path)`

Description:

Checks whether the given path is valid (exists on the filesystem)

Args:

```
path : pathlike object
    #: Uses ParsePath to convert any pathlike input to the underlying Pa
```

Outputs:

```
output : bool
    #: True if the path exists on the filesystem
```

▸ EXPAND SOURCE CODE

Classes

`class Config (sizeThreshold=0, depth=-1, included=None, excluded=None)`

Class:

Class that stores the configuration of the duplicate search. Contains preset size (1000 bytes) for the size clustering threshold.

Description:

Constructor for the Config class. Takes no input, kwargs specify default settings and included/excluded folders if specified.

Inputs:

Kwargs:

```
sizeThreshold : int : default=0
    #: Threshold (bytes) below which to ignore files
depth : int : default=-1
    #: Specifies the search depth (-1 = all)
included : list[path]
    #: List of paths to include (only those listed are used)
excluded : list[path]
    #: List of paths to exclude
```

▸ EXPAND SOURCE CODE

`class File (path, tree=None)`

Class:

Class that represents an individual file and holds various attributes derived from the path. netSize is the file size. If the file cannot be accessed (FileNotFoundError exception) this is logged in the tree object.

Description:

Constructor for the File class. Takes the file pathlike object as input.

Inputs:

Args:

```
path : pathlike object
    #: The pathlike object of the file path
```

Kwargs:

```
tree : Tree : default=None
    #: Reference to the tree this file is within
```

▸ EXPAND SOURCE CODE

```
class Folder (path, makeAllSubdirs=True, isTree=False, tree=None,
             ignoreProgramFolders=True)
```

Class:

Class that represents an individual folder. Contains references to all subfolder objects and the files within this folder.

Description:

Constructor for the Folder class. Takes the folder pathlike object as input. Constructing the folder recursively builds all subfolders and file objects.

Inputs:

Args:

```
path : pathlike object
    #: The pathlike object of the folder path
```

Kwargs:

```
makeAllSubdirs : bool : default=True
    #: If true, recursively construct all subdirectories
isTree : bool : default=False
    #: Signifies if this folder is also a tree root
tree : Tree : default=None
    #: Reference to the tree this folder is within
ignoreProgramFolders : bool : default=True
    #: If true, exclude folders whose name matches those in the EXCLUDEI
```

▸ EXPAND SOURCE CODE

`class Log`

Class:

Class that stores the results of a single duplicate finding run. Contains lists of duplicate fingerprints, and dicts that store scores of similar folders.

Description:

Constructor for the Log class. Takes no input

▸ EXPAND SOURCE CODE

`class Tree (root, ignoreProgramFolders=True)`

Class:

Class that represents the entire file system tree. Contains a reference to the tree root folder (that recursively contains all subfolder and file objects) Contains the file and folder dataframes that represent the entire file system

Description:

Constructor for the Tree class. Takes the tree root pathlike object as input. Constructing the tree root folder recursively builds the tree with all subfolders and files Builds the file and folder dataframes once the file/folders are built

Inputs:

Args:

```
root : pathlike object
    #: The pathlike object of the tree root folder
```

Kwargs:

```
ignoreProgramFolders : bool : default=True
    #: If true, exclude folders whose name matches those in the EXCLUDEI
```

▸ EXPAND SOURCE CODE