Dependenpy

Release 3.2.0

Jun 27, 2017

Contents

1 Dependenpy **1**
- 1.1 License . . . 1
- 1.2 Installation . . . 1
- 1.3 Usage . . . 1
- 1.4 Documentation . . . 3
- 1.5 Development . . . 3

2 Usage **5**
- 2.1 Importing classes . . . 5
- 2.2 Creation of objects . . . 6
- 2.3 Accessing elements . . . 8
- 2.4 Print contents . . . 9

3 Reference **11**
- 3.1 `dependenpy` . . . 11
- 3.2 `dependenpy.cli` . . . 11
- 3.3 `dependenpy.dsm` . . . 12
- 3.4 `dependenpy.finder` . . . 14
- 3.5 `dependenpy.helpers` . . . 16
- 3.6 `dependenpy.node` . . . 17
- 3.7 `dependenpy.structures` . . . 19

4 Contributing **23**
- 4.1 Bug reports . . . 23
- 4.2 Documentation improvements . . . 23
- 4.3 Feature requests and feedback . . . 23
- 4.4 Development . . . 24

5 Authors **27**

6 Changelog **29**
- 6.1 3.2.0 (2017-06-27) . . . 29
- 6.2 3.1.0 (2017-06-02) . . . 29
- 6.3 3.0.0 (2017-05-23) . . . 29
- 6.4 2.0.3 (2017-04-20) . . . 30
- 6.5 0.1.0 to 2.0.2 (2016-10-06) . . . 30
- 6.6 0.1.0 (2016-10-06) . . . 30

7 Indices and tables 31

Python Module Index 33

CHAPTER 1

Dependenpy

Dependenpy allows you to build a dependency matrix for a set of Python packages. To do this, it reads and searches the source code for import statements.

License

Software licensed under ISC license.

Installation

```
pip install dependenpy
```

Usage

Version 3 introduces a command-line tool:

Example:

```
dependenpy -h
```

Result:

```
usage: dependenpy [-d DEPTH] [-f {csv,json,text}] [-g] [-G] [-h] [-i INDENT] [-l] [-m]
                  [-o OUTPUT] [-t] [-v]
                  PACKAGES [PACKAGES ...]

Command line tool for dependenpy Python package.

positional arguments:
```

```
  PACKAGES              The package list. Can be a comma-separated list. Each package
                        must be either a valid path or a package in PYTHONPATH.

optional arguments:
  -d DEPTH, --depth DEPTH
                        Specify matrix or graph depth. Default: best guess.
  -f {csv,json,text}, --format {csv,json,text}
                        Output format. Default: text.
  -g, --show-graph      Show the graph (no text format). Default: false.
  -G, --greedy          Explore subdirectories even if they do not contain an
                        __init__.py file. Can make execution slower. Default: false.
  -h, --help            Show this help message and exit.
  -i INDENT, --indent INDENT
                        Specify output indentation. CSV will never be indented. Text
                        will always have new-lines. JSON can be minified with a
                        negative value. Default: best guess.
  -l, --show-dependencies-list
                        Show the dependencies list. Default: false.
  -m, --show-matrix     Show the matrix. Default: true unless -g, -l or -t.
  -o OUTPUT, --output OUTPUT
                        Output to given file. Default: stdout.
  -t, --show-treemap    Show the treemap (work in progress). Default: false.
  -v, --version         Show the current version of the program and exit.
```

Example:

```
dependenpy dependenpy
dependenpy dependenpy --depth=2
```

Result:

```
               Module | Id ||0|1|2|3|4|5|6|7|8|
----------------------+----++-+-+-+-+-+-+-+-+-+
  dependenpy.__init__ |  0 ||0|0|0|4|0|0|0|0|2|
  dependenpy.__main__ |  1 ||0|0|1|0|0|0|0|0|0|
       dependenpy.cli |  2 ||1|0|0|1|0|4|0|0|0|
      dependenpy.dsm  |  3 ||0|0|0|0|2|1|3|0|0|
    dependenpy.finder |  4 ||0|0|0|0|0|0|0|0|0|
   dependenpy.helpers |  5 ||0|0|0|0|0|0|0|0|0|
      dependenpy.node |  6 ||0|0|0|0|0|0|0|0|3|
   dependenpy.plugins |  7 ||0|0|0|1|0|1|0|0|0|
dependenpy.structures |  8 ||0|0|0|0|0|1|0|0|0|
```

You can also use dependenpy programmatically:

```
from dependenpy import DSM

# create DSM
dsm = DSM('django')

# transform as matrix
matrix = dsm.as_matrix(depth=2)

# initialize with many packages
dsm = DSM('django', 'meerkat', 'appsettings', 'dependenpy', 'archan')
with open('output', 'w') as output:
    dsm.print(format='json', indent=2, output=output)
```

```
# access packages and modules
meerkat = dsm['meerkat']  # or dsm.get('meerkat')
finder = dsm['dependenpy.finder']  # or even dsm['dependenpy']['finder']

# instances of DSM and Package all have print, as_matrix, etc. methods
meerkat.print_matrix(depth=2)
```

This package was originally design to work in a Django project. The Django package django-meerkat uses it to display the matrices with Highcharts.

Documentation

On ReadTheDocs

Development

To run all the tests: `tox`

CHAPTER 2

Usage

- *Importing classes*
- *Creation of objects*
  - *Create a DSM*
  - *Create a Package*
  - *Create a Module*
  - *Create a Dependency*
  - *Create a Matrix*
  - *Create a TreeMap*
  - *Create a Graph*
- *Accessing elements*
- *Print contents*

Importing classes

You can directly import the following classes from `dependenpy`: `DSM`, `Package`, `Module`, `Dependency`, `Matrix` and `TreeMap`.

If you need to import other classes, please take a look at the structure of the code.

Example:

```
from dependenpy import DSM, Matrix
```

Creation of objects

For basic usage, you only have to instantiate a `DSM` object, and sometimes `Matrix` and `TreeMap`. But if you need to do more complicated stuff, you might also want to build instances of `Package`, `Module` or `Dependency`.

Create a DSM

To create a `DSM` object, just pass it a list of packages that can be either found on the disk (absolute or relative paths), or in the Python path (like in `sys.path`).

```
from dependenpy import DSM
django = DSM('django')
flask = DSM('flask')
both = DSM('django', 'flask')
```

Three keyword arguments can be given to `DSM`:

- `build_tree`: Boolean
- `build_dependencies`: Boolean
- `enforce_init`: Boolean

The three of them defaults to true.

Turning `build_tree` to false will delay the build of the Python package tree (the exploration of files on the file system). You can later call `dsm.build_tree()` to build the tree.

Turning `build_dependencies` to false will delay the build of the dependencies (the parsing of the source code to determine the inter-dependencies). You can later call `dsm.build_dependencies()` to build the dependencies. Note that you won't be able to build the dependencies before the tree has been built.

Using true for both `build_tree` and `build_dependencies` is recommended since it is done pretty quickly, even for big projects like Django.

Turning `enforce_init` to false will make the exploration of sub-directories complete: by default, a sub-directory is not explored if it does not contain an `__init__.py` file. It makes the building of the tree faster. But in some cases, you might want to still explore the sub-directory even without `__init__.py`. In that case, use `enforce_init=False`. Note that depending on the tree, the build might take longer.

Create a Package

To create a `Package` object, initialize it with a name and a path. These two arguments are the only one required. Name should be the name of the Python package (the name of the directory), and path should be the path to the directory on the file system.

Example:

```
from dependenpy import Package
absolute_package = Package('django', '/my/virtualenv/lib/python3.5/site-packages/
↪django')
relative_package = Package('program', 'src/program')
```

Additionally, you can pass 6 more keyword arguments: the same three from `DSM` (`build_tree`, `build_dependencies` and `enforce_init`), and the three following:

- `dsm`: parent DSM (instance of DSM).

- `package`: parent package (instance of Package).
- `limit_to`: list of strings to limit the exploration to a subset of directories.

These three arguments default to `None`. Both `dsm` and `package` arguments are useful to build a tree.

Argument `limit_to` can be used this way:

```
from dependenpy import Package
django_auth = Package('django', 'path/to/django',
                      limit_to=['contrib.auth'])
```

Of course, you could also have build a the `django_auth` Package by directly specify the name and path of the sub-directory, but using limit_to allows you to build the full tree, starting at the root (Django's directory).

```
from dependenpy import Package
django_auth = Package('auth', 'path/to/django/contrib/auth')
```

Create a Module

To create a `Module` object, initialize it with a name and a path. These two arguments are the only one required. Name should be the name of the Python module (the file without the `.py` extension), and path should be the path to the file on the file system.

As for `Package`, `dsm` and `package` arguments can be passed when creating a module.

Example:

```
from dependenpy import Module
dsm_module = Module('dsm', 'path/to/dependenpy/dsm.py')
```

Create a Dependency

A dependency is a simple object that require:

- `source`: the `Module` instance importing the item,
- `lineno`: the line number at which the import occurred,
- `target`: the `Package` or `Module` instance from which the item is imported
- and an optional `what` argument which defaults to None: the name of the imported item.

Create a Matrix

From an instance of `DSM` or `Package` called `node`:

```
matrix = node.as_matrix(depth=2)
```

From a list of nodes (DSMs, packages or modules):

```
matrix = Matrix(*node_list, depth=2)
```

An instance of `Matrix` has a `data` attribute, which is a two-dimensions array of integers, and a `keys` attribute which is the list of names, in the same order as rows in data.

Create a TreeMap

From an instance of `DSM` or `Package` called `node`:

```
treemap = node.as_treemap(depth=2)
```

From a list of nodes (DSMs, packages or modules):

```
matrix = TreeMap(*node_list, depth=2)
```

An instance of `TreeMap` has a `data` attribute, which is a two-dimensions array of integers or treemaps, a `keys` attribute which is the list of names in the same order as rows in data, and a `value` attribute which is the total number of dependencies in the treemap.

Create a Graph

From an instance of `DSM` or `Package` called `node`:

```
graph = node.as_graph(depth=2)
```

From a list of nodes (DSMs, packages or modules):

```
graph = Graph(*node_list, depth=2)
```

An instance of `Graph` has a `vertices` attribute, which is a list of `Vertex` instances, and a `edges` attribute which is list of `Edge` instances. See the documentation of `Vertex` and `Edge` for more information.

Accessing elements

Accessing elements in a DSM or a Package is very easy. Just like for a dictionary, you can use the `[]` notation to search for a sub-package or a sub-module. You can also use the `get` method, which is equivalent to the brackets accessor, but will return `None` if the element is not found whereas brackets accessor will raise a `KeyError`.

Example:

```
from dependenpy import DSM

dsm = DSM('django')  # full DSM object, containing Django
django = dsm['django']  # Django Package object
```

You can use dots in the element name to go further in just one instruction:

```
django_auth = django['contrib.auth']
django_forms_models = dsm.get('django.forms.models')
```

Of course, accesses can be chained:

```
django_db_models_utils = dsm['django'].get('db')['models']['utils']
```

Print contents

Contents of DSMs, packages, modules, matrices, treemaps and graphs can be printed with their `print` method. The contents printed are the dependencies. With some exception, each one of them can output contents in three different formats:

- text (by default)
- CSV
- JSON

(Currently, treemaps are not implemented, and graphs can only be printed in JSON or CSV.)

To choose one of these format, just pass the `format` argument, which accepts values `'text'`, `'csv'` and `'json'`. Please note that these values can be replaced by constants imported from `dependenpy.helpers` module:

```
from dependenpy import DSM
from dependenpy.helpers import TEXT, CSV, JSON

dsm = DSM('django')
dsm.print(format=JSON)
```

Depending on the chosen format, additional keyword arguments can be passed to the print method:

- text format: `indent`, indentation value (integer)
- CSV format: `header`, True or False, to display the headers (columns names)
- JSON format: every arguments accepted by `json.dumps`, and in the case of a `Module` instance, `absolute` Boolean to switch between output of absolute and relative paths.

For `DSM` and `Package` instances, shortcuts to print a matrix, a treemap or a graph are available with `print_matrix`, `print_treemap` and `print_graph` methods. These methods will first create the related object and then call the object's own `print` method.

CHAPTER 3

Reference

dependenpy

Dependenpy package.

With dependenpy you will be able to analyze the internal dependencies in your Python code, i.e. which module needs which other module. You will then be able to build a dependency matrix and use it for other purposes.

dependenpy.cli

Module that contains the command line application.

Why does this file exist, and why not put this in __main__?

> You might be tempted to import things from __main__ later, but that will cause problems: the code will get executed twice:
>
> - When you run *python -mdependenpy* python will execute `__main__.py` as a script. That means there won't be any `dependenpy.__main__` in `sys.modules`.
> - When you import __main__ it will get executed again (as a module) because there's no `dependenpy.__main__` in `sys.modules`.
>
> Also see (1) from http://click.pocoo.org/5/setuptools/#setuptools-integration

`dependenpy.cli.`**`get_parser`**`()`
Return a parser for the command-line arguments.

`dependenpy.cli.`**`main`**`(`*`args=None`*`)`
Main function.

This function is the command line entry point.

> **Parameters** **args** (*list of str*) – the arguments passed to the program.
>
> **Returns** *int* – return code being 0 (OK), 1 (dsm empty) or 2 (error).

dependenpy.dsm

dependenpy dsm module.

This is the core module of dependenpy. It contains the following classes:

- *DSM*: to create a DSM-capable object for a list of packages,
- *Package*: which represents a Python package,
- *Module*: which represents a Python module,
- *Dependency*: which represents a dependency between two modules.

class dependenpy.dsm.**DSM**(**packages*, *build_tree=True*, *build_dependencies=True*, *enforce_init=True*)
Bases: *dependenpy.node.RootNode*, *dependenpy.node.NodeMixin*, *dependenpy.helpers.PrintMixin*

DSM-capable class.

Technically speaking, a DSM instance is not a real DSM but more a tree representing the Python packages structure. However, it has the necessary methods to build a real DSM in the form of a square matrix, a dictionary or a tree-map.

__init__(**packages*, *build_tree=True*, *build_dependencies=True*, *enforce_init=True*)
Initialization method.

Parameters

- ***packages** (*args*) – list of packages to search for.
- **build_tree** (*bool*) – auto-build the tree or not.
- **build_dependencies** (*bool*) – auto-build the dependencies or not.
- **enforce_init** (*bool*) – if True, only treat directories if they contain an __init__.py file.

build_tree()
Build the Python packages tree.

isdsm
Inherited from NodeMixin. Always True.

class dependenpy.dsm.**Dependency**(*source*, *lineno*, *target*, *what=None*)
Bases: object

Dependency class.

Represent a dependency from a module to another.

__init__(*source*, *lineno*, *target*, *what=None*)
Initialization method.

Parameters

- **source** (*Module*) – source Module.
- **lineno** (*int*) – number of line at which import statement occurs.
- **target** (*str/Module/Package*) – the target node.
- **what** (*str*) – what is imported (optional).

__weakref__
list of weak references to the object (if defined)

external
Property to tell if the dependency's target is a valid node.

class dependenpy.dsm.**Module**(*name*, *path*, *dsm=None*, *package=None*)
Bases: *dependenpy.node.LeafNode*, *dependenpy.node.NodeMixin*, *dependenpy.helpers.PrintMixin*

Module class.

This class represents a Python module (a Python file).

__contains__(*item*)
Whether given item is contained inside this module.

Parameters **item** (*Package/Module*) – a package or module.

Returns *bool* – True if self is item or item is self's package and self if an __init__ module.

__init__(*name*, *path*, *dsm=None*, *package=None*)
Initialization method.

Parameters

- **name** (*str*) – name of the module.
- **path** (*str*) – path to the module.
- **dsm** (*DSM*) – parent DSM.
- **package** (*Package*) – parent Package.

as_dict(*absolute=False*)
Return the dependencies as a dictionary.

Returns *dict* – dictionary of dependencies.

build_dependencies()
Build the dependencies for this module.

Parse the code with ast, find all the import statements, convert them into Dependency objects.

cardinal(*to*)
Return the number of dependencies of this module to the given node.

Parameters **to** (*Package/Module*) – the target node.

Returns *int* – number of dependencies.

get_imports(*ast_body*)
Return all the import statements given an AST body (AST nodes).

Parameters **ast_body** (*compiled code's body*) – the body to filter.

Returns *list of dict* – the import statements.

ismodule
Inherited from NodeMixin. Always True.

parse_code()
Read the source code and return all the import statements.

Returns *list of dict* – the import statements.

class dependenpy.dsm.**Package**(*name*, *path*, *dsm=None*, *package=None*, *limit_to=None*, *build_tree=True*, *build_dependencies=True*, *enforce_init=True*)
Bases: *dependenpy.node.RootNode*, *dependenpy.node.LeafNode*, *dependenpy.node.NodeMixin*, *dependenpy.helpers.PrintMixin*

Package class.

This class represent Python packages as nodes in a tree.

__init__(*name*, *path*, *dsm=None*, *package=None*, *limit_to=None*, *build_tree=True*, *build_dependencies=True*, *enforce_init=True*)
Initialization method.

Parameters

- **name** (*str*) – name of the package.
- **path** (*str*) – path to the package.
- **dsm** (*DSM*) – parent DSM.
- **package** (*Package*) – parent package.
- **limit_to** (*list of str*) – list of string to limit the recursive tree-building to what is specified.
- **build_tree** (*bool*) – auto-build the tree or not.
- **build_dependencies** (*bool*) – auto-build the dependencies or not.
- **enforce_init** (*bool*) – if True, only treat directories if they contain an `__init__.py` file.

build_tree()
Build the tree for this package.

cardinal(*to*)
Return the number of dependencies of this package to the given node.

Parameters **to** (*Package/Module*) – target node.

Returns *int* – number of dependencies.

ispackage
Inherited from NodeMixin. Always True.

isroot
Property to tell if this node is a root node.

Returns *bool* – this package has no parent.

issubpackage
Property to tell if this node is a sub-package.

Returns *bool* – this package has a parent.

split_limits_heads()
Return first parts of dot-separated strings, and rest of strings.

Returns *(list of str, list of str)* – the heads and rest of the strings.

dependenpy.finder

dependenpy finder module.

class dependenpy.finder.**Finder**(*finders=None*)
Bases: object

Main package finder class.

Initialize it with a list of package finder classes (not instances).

__init__ (*finders=None*)
Initialization method.

Parameters **finders** (*list of classes*) – list of package finder classes (not instances) in a specific order. Default: [LocalPackageFinder, InstalledPackageFinder].

__weakref__
list of weak references to the object (if defined)

find (*package*, ***kwargs*)
Find a package using package finders.

Return the first package found.

Parameters

- **package** (*str*) – package to find.
- ****kwargs ()** – additional keyword arguments used by finders.

Returns *PackageSpec* – if package found, else None

class dependenpy.finder.**InstalledPackageFinder**
Bases: *dependenpy.finder.PackageFinder*

Finder to find installed Python packages using importlib.

find (*package*, ***kwargs*)
Find method.

Parameters

- **package** (*str*) – package to find.
- ****kwargs ()** – additional keyword arguments.

Returns *PackageSpec* – the PackageSpec corresponding to the package, or None.

class dependenpy.finder.**LocalPackageFinder**
Bases: *dependenpy.finder.PackageFinder*

Finder to find local packages (directories on the disk).

find (*package*, ***kwargs*)
Find method.

Parameters

- **package** (*str*) – package to find.
- ****kwargs ()** – additional keyword arguments.

Returns *PackageSpec* – the PackageSpec corresponding to the package, or None.

class dependenpy.finder.**PackageFinder**
Bases: object

Abstract package finder class.

__weakref__
list of weak references to the object (if defined)

find (*package*, ***kwargs*)
Find method.

Parameters

- **package** (*str*) – package to find.

- ****kwargs ()** – additional keyword arguments.

Returns *PackageSpec* – the PackageSpec corresponding to the package, or None.

class dependenpy.finder.**PackageSpec**(*name*, *path*, *limit_to=None*)
Bases: object

Holder for a package specification (given as argument to DSM).

__init__(*name*, *path*, *limit_to=None*)
Initialization method.

Parameters

- **name** (*str*) – name of the package.
- **path** (*str*) – path to the package.
- **limit_to** (*list of str*) – limitations.

__weakref__
list of weak references to the object (if defined)

add(*spec*)
Add limitations of given spec to self's.

Parameters **spec** (*PackageSpec*) – another spec.

static combine(*specs*)
Combine package specifications' limitations.

Parameters **specs** (*list of PackageSpec*) – the package specifications.

Returns *list of PackageSpec* – the new, merged list of PackageSpec.

ismodule
Property to tell if the package is in fact a module (a file).

dependenpy.helpers

dependenpy printer module.

class dependenpy.helpers.**PrintMixin**
Bases: object

Print mixin class.

__weakref__
list of weak references to the object (if defined)

print(*format='text'*, *output=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>*, ***kwargs*)
Print the object in a file or on standard output by default.

Parameters

- **format** (*str*) – output format (csv, json or text).
- **output** (*file*) – descriptor to an opened file (default to standard output).
- ****kwargs ()** – additional arguments.

dependenpy.helpers.**guess_depth**(*packages*)
Guess the optimal depth to use for the given list of arguments.

Parameters **packages** (*list of str*) – list of packages.

Returns *int* – guessed depth to use.

dependenpy.node

dependenpy node module.

class dependenpy.node.**LeafNode**
Bases: object

Shared code between Package and Module.

__init__()
Initialization method.

__str__()
String method.

__weakref__
list of weak references to the object (if defined)

absolute_name(*depth=0*)
Return the absolute name of the node.

Concatenate names from root to self within depth.

Parameters **depth** (*int*) – maximum depth to go to.

Returns *str* – absolute name of the node (until given depth is reached).

depth
Property to tell the depth of the node in the tree.

Returns *int* – the node's depth in the tree.

root
Property to return the root of this node.

Returns *Package* – this node's root package.

class dependenpy.node.**NodeMixin**
Bases: object

Shared code between DSM, Package and Module.

__weakref__
list of weak references to the object (if defined)

isdsm
Property to check if object is instance of DSM.

ismodule
Property to check if object is instance of Module.

ispackage
Property to check if object is instance of Package.

class dependenpy.node.**RootNode**(*build_tree=True*)
Bases: object

Shared code between DSM and Package.

__bool__()
Node as Boolean.

Returns *bool* – result of node.empty.

__contains__(*item*)
Get result of _contains, cache it and return it.

Parameters **item** (*Package/Module*) – a package or module.

Returns *bool* – True if self contains item, False otherwise.

__getitem__(*item*)
Return the corresponding Package or Module object.

Parameters **item** (*str*) – name of the package/module, dot-separated.

Returns *Package/Module* – corresponding object.

__init__(*build_tree=True*)
Initialization method.

Parameters **build_tree** (*bool*) – whether to immediately build the tree or not.

__weakref__
list of weak references to the object (if defined)

as_dict()
Return the dependencies as a dictionary.

Returns *dict* – dictionary of dependencies.

as_graph(*depth=0*)
Create a graph with self as node, cache it, return it.

Parameters **depth** (*int*) – depth of the graph.

Returns *Graph* – an instance of Graph.

as_matrix(*depth=0*)
Create a matrix with self as node, cache it, return it.

Parameters **depth** (*int*) – depth of the matrix.

Returns *Matrix* – an instance of Matrix.

as_treemap()
Return the dependencies as a TreeMap.

Returns *TreeMap* – instance of TreeMap.

build_dependencies()
Recursively build the dependencies for sub-modules and sub-packages.

Iterate on node's modules then packages and call their build_dependencies methods.

build_tree()
To be overridden.

empty
Whether the node has neither modules nor packages.

Returns *bool* – True if empty, False otherwise.

get(*item*)
Get item through `__getitem__` and cache the result.

Parameters **item** (*str*) – name of package or module.

Returns *Package/Module* – the corresponding object.

get_target(*target*)
Get the result of _get_target, cache it and return it.

Parameters **target** (*str*) – target to find.

Returns *Package/Module* – package containing target or corresponding module.

print_graph(*format=None*, *output=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>*, *depth=0*, ***kwargs*)
Print the graph for self's nodes.

Parameters

- **format** (*str*) – output format (csv, json or text).
- **output** (*file*) – file descriptor on which to write.
- **depth** (*int*) – depth of the graph.

print_matrix(*format=None*, *output=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>*, *depth=0*, ***kwargs*)
Print the matrix for self's nodes.

Parameters

- **format** (*str*) – output format (csv, json or text).
- **output** (*file*) – file descriptor on which to write.
- **depth** (*int*) – depth of the matrix.

print_treemap(*format=None*, *output=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>*, ***kwargs*)
Print the matrix for self's nodes.

Parameters

- **format** (*str*) – output format (csv, json or text).
- **output** (*file*) – file descriptor on which to write.

submodules
Property to return all sub-modules of the node, recursively.

Returns *list of Module* – the sub-modules.

dependenpy.structures

dependenpy structures module.

class dependenpy.structures.**Edge**(*vertex_out*, *vertex_in*, *weight=1*)
Bases: object

Edge class. Used in Graph class.

__init__(*vertex_out*, *vertex_in*, *weight=1*)
Initialization method.

Parameters

- **vertex_out** (*Vertex*) – source vertex (edge going out).

- **vertex_in** (*Vertex*) – target vertex (edge going in).
- **weight** (*int*) – weight of the edge.

__weakref__
list of weak references to the object (if defined)

go_from(*vertex*)
Tell the edge to go out from this vertex.

Parameters **vertex** (*Vertex*) – vertex to go from.

go_in(*vertex*)
Tell the edge to go into this vertex.

Parameters **vertex** (*Vertex*) – vertex to go into.

class dependenpy.structures.**Graph**(**nodes*, *depth=0*)
Bases: *dependenpy.helpers.PrintMixin*

Graph class.

A class to build a graph given a list of nodes. After instantiation, it has two attributes: vertices, the set of nodes, and edges, the set of edges.

__init__(**nodes*, *depth=0*)
Initialization method.

An intermediary matrix is built to ease the creation of the graph.

Parameters

- ***nodes** (*list of DSM/Package/Module*) – the nodes on which to build the graph.
- **depth** (*int*) – the depth of the intermediary matrix. See the documentation for Matrix class.

class dependenpy.structures.**Matrix**(**nodes*, *depth=0*)
Bases: *dependenpy.helpers.PrintMixin*

Matrix class.

A class to build a matrix given a list of nodes. After instantiation, it has two attributes: data, a 2-dimensions array, and keys, the names of the entities in the corresponding order.

__init__(**nodes*, *depth=0*)
Initialization method.

Parameters

- ***nodes** (*list of DSM/Package/Module*) – the nodes on which to build the matrix.
- **depth** (*int*) – the depth of the matrix. This depth is always absolute, meaning that building a matrix with a sub-package "A.B.C" and a depth of 1 will return a matrix of size 1, containing A only. To see the matrix for the sub-modules and sub-packages in C, you will have to give depth=4.

static cast(*keys*, *data*)
Cast a set of keys and an array to a Matrix object.

total
Return the total number of dependencies within this matrix.

class dependenpy.structures.**TreeMap**(**nodes*, *value=-1*, *data=None*, *keys=None*)
Bases: *dependenpy.helpers.PrintMixin*

TreeMap class.

__init__(**nodes*, *value=-1*, *data=None*, *keys=None*)
Initialization method.

Parameters ***nodes** (*list of Node*) – the nodes from which to build the treemap.

class dependenpy.structures.**Vertex**(*name*)
Bases: object

Vertex class. Used in Graph class.

__init__(*name*)
Initialization method.

Parameters **name** (*str*) – name of the vertex.

__weakref__
list of weak references to the object (if defined)

connect_from(*vertex*, *weight=1*)
Connect another vertex to this one.

Parameters

- **vertex** (*Vertex*) – vertex to connect from.
- **weight** (*int*) – weight of the edge.

Returns *Edge* – the newly created edge.

connect_to(*vertex*, *weight=1*)
Connect this vertex to another one.

Parameters

- **vertex** (*Vertex*) – vertex to connect to.
- **weight** (*int*) – weight of the edge.

Returns *Edge* – the newly created edge.

CHAPTER 4

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

Bug reports

When reporting a bug please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Documentation improvements

Dependenpy could always use more documentation, whether as part of the official Dependenpy docs, in docstrings, or even on the web in blog posts, articles, and such.

Feature requests and feedback

The best way to send feedback is to file an issue at https://github.com/Pawamoy/dependenpy/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that code contributions are welcome :)

Development

To set up *dependenpy* for local development:

1. Fork dependenpy (look for the "Fork" button).
2. Clone your fork locally:

```
git clone git@github.com:your_name_here/dependenpy.git
```

3. Create a branch for local development:

```
git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

4. When you're done making changes, run all the tests with one command:

```
tox
```

5. Commit your changes and push your branch to GitHub:

```
git add .
git commit -m "Your detailed description of your changes."
git push origin name-of-your-bugfix-or-feature
```

6. Submit a pull request through the GitHub website.

Pull Request Guidelines

If you need some code review or feedback while you're developing the code just make the pull request.

For merging, you should:

1. Include passing tests (run `tox`)[1].
2. Update documentation when there's new API, functionality etc.
3. Add a note to `CHANGELOG.rst` about the changes.
4. Add yourself to `AUTHORS.rst`.

[1] If you don't have all the necessary python versions available locally you can rely on...

- **Travis**: it will run the tests for each change you add in the pull request. It will be slower though...
- **pyenv**:

```
# important libraries to compile Python
sudo apt install -y libssl-dev openssl zlib1g-dev sqlite3 libsqlite3-dev libbz2-dev bzip2

git clone https://github.com/pyenv/pyenv.git ~/.pyenv
export PATH="${HOME}/.pyenv/bin:${PATH}"
eval "$(pyenv init -)"

pyenv install 3.5.3
pyenv install 3.6.0  # etc.
pyenv global system 3.5.3 3.6.0
```

Tips

To run a subset of tests:

```
tox -e envname -- py.test -k test_myfeature
```

To run all the test environments in *parallel* (you need to `pip install detox`):

```
detox
```

CHAPTER 5

Authors

- Pierre Parrend
- Timothée Mazzucotelli
- Florent Colin
- Jean-Louis Mandel

CHAPTER 6

Changelog

3.2.0 (2017-06-27)

- Change `-g` short option for `--greedy` to `-G`.
- Add `-g, --show-graph` option with related graph class and capabilities.
- Add a provider for Archan (`dependenpy.plugins.InternalDependencies`).
- Update documentation accordingly.

3.1.0 (2017-06-02)

- Change `-i, --enforce-init` option to its contrary `-g, --greedy`.
- Add `-i, --indent` option to specify indentation level.
- Options `-l`, `-m` and `-t` are now mutually exclusive.
- Fix matrix build for depth 0.
- Print methods have been improved.
- Update documentation.

3.0.0 (2017-05-23)

This version is a big refactoring. The code is way more object oriented, cleaner, shorter, simpler, smarter, more user friendly- in short: better.

Additional features:

- command line entry point,

- runtime static imports are now caught (in functions or classes), as well as import statements (previously only from import).

2.0.3 (2017-04-20)

- Fix occasional UnicodeEncode when reading UTF-8 file.
- Handle bad characters in files when parsing with `ast`.

0.1.0 to 2.0.2 (2016-10-06)

- Development (alpha then beta version).

0.1.0 (2016-10-06)

- Alpha release on PyPI.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

d

dependenpy, 11
dependenpy.cli, 11
dependenpy.dsm, 12
dependenpy.finder, 14
dependenpy.helpers, 16
dependenpy.node, 17
dependenpy.structures, 19

Index

Symbols

__bool__() (dependenpy.node.RootNode method), 17
__contains__() (dependenpy.dsm.Module method), 13
__contains__() (dependenpy.node.RootNode method), 18
__getitem__() (dependenpy.node.RootNode method), 18
__init__() (dependenpy.dsm.DSM method), 12
__init__() (dependenpy.dsm.Dependency method), 12
__init__() (dependenpy.dsm.Module method), 13
__init__() (dependenpy.dsm.Package method), 14
__init__() (dependenpy.finder.Finder method), 14
__init__() (dependenpy.finder.PackageSpec method), 16
__init__() (dependenpy.node.LeafNode method), 17
__init__() (dependenpy.node.RootNode method), 18
__init__() (dependenpy.structures.Edge method), 19
__init__() (dependenpy.structures.Graph method), 20
__init__() (dependenpy.structures.Matrix method), 20
__init__() (dependenpy.structures.TreeMap method), 20
__init__() (dependenpy.structures.Vertex method), 21
__str__() (dependenpy.node.LeafNode method), 17
__weakref__ (dependenpy.dsm.Dependency attribute), 12
__weakref__ (dependenpy.finder.Finder attribute), 15
__weakref__ (dependenpy.finder.PackageFinder attribute), 15
__weakref__ (dependenpy.finder.PackageSpec attribute), 16
__weakref__ (dependenpy.helpers.PrintMixin attribute), 16
__weakref__ (dependenpy.node.LeafNode attribute), 17
__weakref__ (dependenpy.node.NodeMixin attribute), 17
__weakref__ (dependenpy.node.RootNode attribute), 18
__weakref__ (dependenpy.structures.Edge attribute), 20
__weakref__ (dependenpy.structures.Vertex attribute), 21

A

absolute_name() (dependenpy.node.LeafNode method), 17
add() (dependenpy.finder.PackageSpec method), 16
as_dict() (dependenpy.dsm.Module method), 13
as_dict() (dependenpy.node.RootNode method), 18
as_graph() (dependenpy.node.RootNode method), 18
as_matrix() (dependenpy.node.RootNode method), 18
as_treemap() (dependenpy.node.RootNode method), 18

B

build_dependencies() (dependenpy.dsm.Module method), 13
build_dependencies() (dependenpy.node.RootNode method), 18
build_tree() (dependenpy.dsm.DSM method), 12
build_tree() (dependenpy.dsm.Package method), 14
build_tree() (dependenpy.node.RootNode method), 18

C

cardinal() (dependenpy.dsm.Module method), 13
cardinal() (dependenpy.dsm.Package method), 14
cast() (dependenpy.structures.Matrix static method), 20
combine() (dependenpy.finder.PackageSpec static method), 16
connect_from() (dependenpy.structures.Vertex method), 21
connect_to() (dependenpy.structures.Vertex method), 21

D

Dependency (class in dependenpy.dsm), 12
dependenpy (module), 11
dependenpy.cli (module), 11
dependenpy.dsm (module), 12
dependenpy.finder (module), 14
dependenpy.helpers (module), 16
dependenpy.node (module), 17
dependenpy.structures (module), 19
depth (dependenpy.node.LeafNode attribute), 17
DSM (class in dependenpy.dsm), 12

E

Edge (class in dependenpy.structures), 19
empty (dependenpy.node.RootNode attribute), 18
external (dependenpy.dsm.Dependency attribute), 12

F

find() (dependenpy.finder.Finder method), 15
find() (dependenpy.finder.InstalledPackageFinder method), 15
find() (dependenpy.finder.LocalPackageFinder method), 15
find() (dependenpy.finder.PackageFinder method), 15
Finder (class in dependenpy.finder), 14

G

get() (dependenpy.node.RootNode method), 18
get_imports() (dependenpy.dsm.Module method), 13
get_parser() (in module dependenpy.cli), 11
get_target() (dependenpy.node.RootNode method), 19
go_from() (dependenpy.structures.Edge method), 20
go_in() (dependenpy.structures.Edge method), 20
Graph (class in dependenpy.structures), 20
guess_depth() (in module dependenpy.helpers), 16

I

InstalledPackageFinder (class in dependenpy.finder), 15
isdsm (dependenpy.dsm.DSM attribute), 12
isdsm (dependenpy.node.NodeMixin attribute), 17
ismodule (dependenpy.dsm.Module attribute), 13
ismodule (dependenpy.finder.PackageSpec attribute), 16
ismodule (dependenpy.node.NodeMixin attribute), 17
ispackage (dependenpy.dsm.Package attribute), 14
ispackage (dependenpy.node.NodeMixin attribute), 17
isroot (dependenpy.dsm.Package attribute), 14
issubpackage (dependenpy.dsm.Package attribute), 14

L

LeafNode (class in dependenpy.node), 17
LocalPackageFinder (class in dependenpy.finder), 15

M

main() (in module dependenpy.cli), 11
Matrix (class in dependenpy.structures), 20
Module (class in dependenpy.dsm), 13

N

NodeMixin (class in dependenpy.node), 17

P

Package (class in dependenpy.dsm), 13
PackageFinder (class in dependenpy.finder), 15
PackageSpec (class in dependenpy.finder), 16
parse_code() (dependenpy.dsm.Module method), 13
print() (dependenpy.helpers.PrintMixin method), 16
print_graph() (dependenpy.node.RootNode method), 19
print_matrix() (dependenpy.node.RootNode method), 19
print_treemap() (dependenpy.node.RootNode method), 19
PrintMixin (class in dependenpy.helpers), 16

R

root (dependenpy.node.LeafNode attribute), 17
RootNode (class in dependenpy.node), 17

S

split_limits_heads() (dependenpy.dsm.Package method), 14
submodules (dependenpy.node.RootNode attribute), 19

T

total (dependenpy.structures.Matrix attribute), 20
TreeMap (class in dependenpy.structures), 20

V

Vertex (class in dependenpy.structures), 21