knittingpattern Documentation

Release 19

AllYarnsAreBeautiful
FOSSASIA

Feb 26, 2017

Contents

Contents:

CHAPTER 1

knittingpattern Installation Instructions

Package installation from Pypi

The knittingpattern library requires Python 3. It can be installed form the Python Package Index.

Windows

Install it with a specific python version under windows:

```
py -3 -m pip --no-cache-dir install --upgrade knittingpattern
```

Test the installed version:

```
py -3 -m pytest --pyargs knittingpattern
```

Linux

To install the version from the python package index, you can use your terminal and execute this under Linux:

```
sudo python3 -m pip --no-cache-dir install --upgrade knittingpattern
```

test the installed version:

```
python3 -m pytest --pyargs knittingpattern
```

Installation from Repository

You can setup the development version under Windows and Linux.

Linux

If you wish to get latest source version running, you can check out the repository and install it manually.

```
git clone https://github.com/fossasia/knittingpattern.git
cd knittingpattern
sudo python3 -m pip install --upgrade pip
sudo python3 -m pip install -r requirements.txt
sudo python3 -m pip install -r test-requirements.txt
py.test
```

To also make it importable for other libraries, you can link it into the site-packages folder this way:

```
sudo python3 setup.py link
```

Windows

Same as under *Linux* but you need to replace `sudo python3` with `py -3`. This also counts for the following documentation.

CHAPTER 2

Knitting Pattern File Format Specification

For the words see the glossary.

Design Decisions

Concerns:

- We can never implement everything that is possible with knitting. We must therefore allow instructions to be arbitrary.
- We can not use a grid as a basis. This does not reflect if you split the work and make i.e. two big legs
- Knitting can be done on the right and on the wrong side. The same result can be achived when knitting in both directions.

Assumptions

- we start from bottom right
- default instruction (see)

```
{
  "type" : "knit",
}
{
  "type" : "ktog tbl", # identifier
  "count" : 2
}
```

- default connection

```
{
  "start" : 0,
}
```

- `"id"` can point to an object.

CHAPTER 3

Development Setup

Make sure that you have the *repository installed*.

Install Requirements

To install all requirements for the development setup, execute

```
pip install --upgrade -r requirements.txt -r test-requirements.txt -r dev-
↪requirements.txt
```

Sphinx Documentation Setup

Sphinx was setup using the tutorial from readthedocs. It should be already setup if you completed *the previous step*.

Further reading:

- domains

With Notepad++ under Windows, you can run the make_html.bat file in the `docs` directory to create the documentation and show undocumented code.

Code Climate

To install the code climate command line interface (cli), read about it in their github repository You need docker to be installed. Under Linux you can execute this in the Terminal to install docker:

```
wget -qO- https://get.docker.com/ | sh
sudo usermod -aG docker $USER
```

Then, log in and out. Then, you can install the command line interface:

```
wget -qO- https://github.com/codeclimate/codeclimate/archive/master.tar.gz | tar xvz
cd codeclimate-* && sudo make install
```

Then, go to the knittingpattern repository and analyze it.

```
codeclimate analyze
```

Version Pinning

We use version pinning, described in this blog post (outdated). Also read the current version for how to set up.

After installation you can run

```
pip install -r requirements.in -r test-requirements.in -r dev-requirements.in
pip-compile --output-file requirements.txt requirements.in
pip-compile --output-file test-requirements.txt test-requirements.in
pip-compile --output-file dev-requirements.txt dev-requirements.in
pip-sync requirements.txt dev-requirements.txt test-requirements.txt
pip install --upgrade -r requirements.txt -r test-requirements.txt -r dev-
↪requirements.txt
```

`pip-sync` uninstalls every package you do not need and writes the fix package versions to the requirements files.

Continuous Integration to Pypi

Before you put something on Pypi, ensure the following:

1. The version is in the master branch on github.
2. The tests run by travis-ci run successfully.

Pypi is automatically deployed by travis. See here. To upload new versions, tag them with git and push them.

```
setup.py tag_and_deploy
```

The tag shows up as a travis build. If the build succeeds, it is automatically deployed to Pypi.

Manual Upload to the Python Package Index

However, here you can see how to upload this package manually.

Version

Throughout this chapter, `<new_version>` refers to a a string of the form `[0-9]+\.[0-9]+\.[0-9]+[ab]?` or `<MAYOR>.<MINOR>.<STEP>[<MATURITY>]` where `<MAYOR>`, `<MINOR>` and, `<STEP>` represent numbers and `<MATURITY>` can be a letter to indicate how mature the release is.

1. Create a new branch for the version.

```
git checkout -b <new_version>
```

2. Increase the `__version__` in __init__.py
   - no letter at the end means release
   - `b` in the end means Beta
   - `a` in the end means Alpha
3. Commit and upload this version.

```
git add knittingpattern/__init__.py
git commit -m "version <new_version>"
git push origin <new_version>
```

4. Create a pull-request.
5. Wait for travis-ci to pass the tests.
6. Merge the pull-request.
7. Checkout the master branch and pull the changes from the *commit*.

```
git checkout master
git pull
```

8. Tag the version at the master branch with a `v` in the beginning and push it to github.

```
git tag v<new_version>
git push origin v<new_version>
```

9. *Upload* the code to Pypi.

Upload

First ensure all tests are running:

```
setup.py pep8
```

From docs.python.org:

```
setup.py sdist bdist_wininst upload register
```

Classifiers

You can find all Pypi classifiers here.

CHAPTER 4

Reference

The `knittingpattern` Module Reference

`knittingpattern` Module

The knitting pattern module.

Load and convert knitting patterns using the convenience functions listed below.

knittingpattern. **load_from_object** (*object_*)
Load a knitting pattern from an object.

Return type *knittingpattern.KnittingPatternSet.KnittingPatternSet*

knittingpattern. **load_from_string** (*string*)
Load a knitting pattern from a string.

Return type *knittingpattern.KnittingPatternSet.KnittingPatternSet*

knittingpattern. **load_from_file** (*file*)
Load a knitting pattern from a file-like object.

Return type *knittingpattern.KnittingPatternSet.KnittingPatternSet*

knittingpattern. **load_from_path** (*path*)
Load a knitting pattern from a file behind located at *path*.

Return type *knittingpattern.KnittingPatternSet.KnittingPatternSet*

knittingpattern. **load_from_url** (*url*)
Load a knitting pattern from a url.

Return type *knittingpattern.KnittingPatternSet.KnittingPatternSet*

knittingpattern. **load_from_relative_file** (*module*, *path_relative_to*)
Load a knitting pattern from a path relative to a module.

Parameters

- **module** (*str*) – can be a module's file, a module's name or a module's path.
- **path_relative_to** (*str*) – is the path relative to the modules location. The result is loaded from this.

Return type *knittingpattern.KnittingPatternSet.KnittingPatternSet*

knittingpattern. **convert_from_image** (*colors=('white', 'black')*)
Convert and image to a knitting pattern.

Returns a loader

Return type *knittingpattern.Loader.PathLoader*

Parameters **colors** (*tuple*) – the colors to convert to

```
convert_from_image().path("pattern.png").path("pattern.json")
convert_from_image().path("pattern.png").knitting_pattern()
```

See also:

`knittingoattern.convert.image_to_knitting_pattern`

knittingpattern. **load_from** ( )

Create a loader to load knitting patterns with.

Returns the loader to load objects with

Return type *knittingpattern.Loader.JSONLoader*

Example:

```
import knittingpattern, webbrowser
k = knittingpattern.load_from().example("Cafe.json")
webbrowser.open(k.to_svg(25).temporary_path(".svg"))
```

knittingpattern. **new_knitting_pattern** (*id_*, *name=None*)

Create a new knitting pattern.

Returns a new empty knitting pattern.

Parameters

- **id_** – the id of the knitting pattern
- **name** – the name of the knitting pattern or `None` if the `id_` should be used

Return type *knittingpattern.KnittingPattern.KnittingPattern*

See also:

KnittingPatternSet.add_new_pattern()

knittingpattern. **new_knitting_pattern_set** ( )

Create a new, empty knitting pattern set.

Return type *knittingpattern.KnittingPatternSet.KnittingPatternSet*

Returns a new, empty knitting pattern set

`IdCollection` Module

See this module if you like to store object s that have an `id` attribute.

class knittingpattern.IdCollection. **IdCollection**

Bases: `object`

This is a collections of object that have an `id` attribute.

__bool__ ( )

Returns whether there is anything in the collection.

Return type bool

__getitem__ (*id_*)
Get the object with the `id`

```
ic = IdCollection()
ic.append(object_1)
ic.append(object_2)
assert ic[object_1.id] == object_1
assert ic[object_2.id] == object_1
```

Parameters **id_** – the id of an object

Returns the object with the `id`

Raises **KeyError** – if no object with `id` was found

__init__ ( )
Create a new *IdCollection* with no arguments.

You can add objects later using the method *append()* .

__iter__ ( )
allows you to iterate and use for-loops

The objects in the iterator have the order in which they were appended.

__len__ ( )

Returns the number of objects in this collection

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

append (*item*)
Add an object to the end of the *IdCollection* .

Parameters **item** – an object that has an id

at (*index*)
Get the object at an `index` .

Parameters **index** (*int*) – the index of the object

Returns the object at `index`

first
The first element in this collection.

Returns the first element in this collection

Raises **IndexError** – if this collection is empty

`Instruction` Module

Knitting patterns consist of instructions.

The *instructions* . that are used in the `knitting patterns` can be foudn in this module. They have certain attributes in common.

class `knittingpattern.Instruction.` **Instruction** (*specification*, *inherited_values=()*)
Bases: *knittingpattern.Prototype.Prototype*

Instructions specify what should be done during knitting.

This class represents the basic interface for instructions.

It is based on the *Prototype* which allows creating instructions based on other instructions so they can inherit their attributes.

You can create new instructions by passing a specification to them which can consist of a `dictionary` or an other *prototype*. For such specifications see the *InstructionLibrary*.

color
The color of the instruction.

Returns the *color* of the instruction or `None` if none is specified.

colors
All the colors that an instruction has.

Returns a list of colors of the instruction. If the instruction has no color, this is `[None]`.

Return type list

consumes_meshes ( )
Whether this instruction consumes meshes.

Returns whether this instruction consumes any meshes

Return type bool

See also:

number_of_consumed_meshes

description
The description of the instruction.

Returns the `description` of the instruction or `None` if none is specified.

does_knit ( )
Whether this instruction is a knit instruction.

Returns whether this instruction is a knit instruction

Return type bool

does_purl ( )
Whether this instruction is a purl instruction.

Returns whether this instruction is a purl instruction

Return type bool

has_color ( )
Whether this instruction has a color.

Returns whether a *color* is specified

Return type bool

hex_color
The color in "#RRGGBB" format.

Returns the *color* in "#RRGGBB" format or none if no color is given

id
The id of the instruction.

Returns the *id* of the instruction or `None` if none is specified.

number_of_consumed_meshes
The number of meshes that this instruction consumes.

Returns the *number of consumed meshes* of the instruction or *DEFAULT_NUMBER_OF_CONSUMED_MESHES* if none is specified.

number_of_produced_meshes
The number of meshes that this instruction produces.

Returns the *number of produced meshes* of the instruction or *DEFAULT_NUMBER_OF_PRODUCED_MESHES* if none is specified.

produces_meshes ( )
Whether this institution produces meshes.

Returns whether this instruction produces any meshes

Return type bool

See also:

number_of_produced_meshes

render_z
The z-index of the instruction when rendered.

Returns the z-index of the instruction. Instructions with a higher z-index are displayed in front of instructions with lower z-index.

Return type float

to_svg (*converter=None*)
Return a SVGDumper for this instruction.

Parameters **converter** – a :class:‘ knittingpattern.convert.InstructionSVGCache.InstructionSVGCache‘ or None . If None is given, the :func:‘ knittingpattern.convert.InstructionSVGCache.default_svg_cache‘ is used.

Return type *knittingpattern.Dumper.SVGDumper*

type
The type of the instruction.

Returns the *type* of the instruction or *DEFAULT_TYPE* if none is specified.

Return type str

The type should be a string. Depending on the type, the instruction can receive additional attributes.

See also:

knittingpattern.InstructionLibrary

class knittingpattern.Instruction. **InstructionInRow** (*row*, *spec*)
Bases: *knittingpattern.Instruction.Instruction*

Instructions can be placed in rows.

Then, they have additional attributes and properties.

__init__ (*row*, *spec*)
Create a new instruction in a row with a specification.

Parameters

- **row** (knittingpattern.Row.Row) – the row the instruction is placed in

- **spec** – specification of the instruction

__repr__ ( )
repr(instruction) used for print() .

Returns the string representation of this object

Return type str

color
The color of the instruction.

Returns the *color* of the instruction or None if none is specified.

If no color is specified in the instruction, it is inherited form the row.

consumed_meshes
The meshes consumed by this instruction

Returns a list of *meshes* that this instruction consumes

Return type list

```
assert len(inst.consumed_meshes) == inst.number_of_consumed_meshes
assert all(mesh.is_consumed() for mesh in inst.consumed_meshes)
```

See also:

produced_meshes, *producing_instructions*

consuming_instructions
Instructions that consume the meshes that this instruction produces.

Returns a list of *instructions*

Return type list

See also:

producing_instructions, *produced_meshes*

first_consumed_mesh
The first consumed mesh.

Returns the first consumed mesh

Return type *knittingpattern.Mesh.Mesh*

Raises **IndexError** – if no mesh is consumed

See also:

Instruction.number_of_consumed_meshes

first_produced_mesh
The first produced mesh.

Returns the first produced mesh

Return type *knittingpattern.Mesh.Mesh*

Raises **IndexError** – if no mesh is produced

See also:

Instruction.number_of_produced_meshes

get_index_in_row ( )
Index of the instruction in the instructions of the row or None.

Returns index in the *row* 's instructions or None, if the instruction is not in the row

Return type int

See also:

row_instructions , *index_in_row* , *is_in_row()*

index_in_row
Index of the instruction in the instructions of the row.

Returns index in the *row* 's instructions

Return type int

Raises ***knittingpattern.Instruction.InstructionNotFoundInRow*** – if the instruction is not found at the index

```
index = instruction.index_in_row
assert instruction.row.instructions[index] == instruction
```

See also:

row_instructions , *get_index_in_row()* , *is_in_row()*

index_of_first_consumed_mesh_in_row
The index of the first consumed mesh of this instruction in its row.

Same as *index_of_first_produced_mesh_in_row* but for consumed meshes.

index_of_first_produced_mesh_in_row
Index of the first produced mesh in the row that consumes it.

Returns an index of the first produced mesh of rows produced meshes

Return type int

Note: If the instruction *produces meshes* , this is the index of the first mesh the instruction produces in all the meshes of the row. If the instruction does not produce meshes, the index of the mesh is returned as if the instruction had produced a mesh.

```
if instruction.produces_meshes():
    index = instruction.index_of_first_produced_mesh_in_row
```

index_of_last_consumed_mesh_in_row
The index of the last consumed mesh of this instruction in its row.

Same as *index_of_last_produced_mesh_in_row* but for the last consumed mesh.

index_of_last_produced_mesh_in_row
Index of the last mesh produced by this instruction in its row.

Returns an index of the last produced mesh of rows produced meshes

Return type int

Note: If this instruction *produces meshes*, this is the index of its last produces mesh in the row. However, if this instruction does not produce meshes, this is the index **before** the first mesh of the instruction if it produced meshes.

See also:

index_of_first_produced_mesh_in_row

is_in_row()
Whether the instruction can be found in its row.

Returns whether the instruction is in its row

Return type bool

Use this to avoid raising and *InstructionNotFoundInRow*.

last_consumed_mesh
The last consumed mesh.

Returns the last consumed mesh

Return type *knittingpattern.Mesh.Mesh*

Raises **IndexError** – if no mesh is consumed

See also:

Instruction.number_of_consumed_meshes

last_produced_mesh
The last produced mesh.

Returns the last produced mesh

Return type *knittingpattern.Mesh.Mesh*

Raises **IndexError** – if no mesh is produced

See also:

Instruction.number_of_produced_meshes

next_instruction_in_row
The instruction after this one or None.

Returns the instruction in *row_instructions* after this or `None` if this is the last

Return type *knittingpattern.Instruction.InstructionInRow*

This can be used to traverse the instructions.

See also:

previous_instruction_in_row

previous_instruction_in_row
The instruction before this one or None.

Returns the instruction in *row_instructions* before this or `None` if this is the first

Return type *knittingpattern.Instruction.InstructionInRow*

This can be used to traverse the instructions.

See also:

next_instruction_in_row

produced_meshes
The meshes produced by this instruction

Returns a `list` of *meshes* that this instruction produces

Return type list

```
assert len(inst.produced_meshes) == inst.number_of_produced_meshes
assert all(mesh.is_produced() for mesh in inst.produced_meshes)
```

See also:

consumed_meshes, *consuming_instructions*

producing_instructions
Instructions that produce the meshes that this instruction consumes.

Returns a list of *instructions*

Return type list

See also:

consuming_instructions, *consumed_meshes*

row
The row this instruction is in.

Returns the row the instruction is placed in

Return type *knittingpattern.Row.Row*

row_instructions
Shortcut for `instruction.row.instructions`.

Returns the instructions of the *row* the instruction is in

See also:

index_in_row

transfer_to_row (*new_row*)
Transfer this instruction to a new row.

Parameters **new_row** (knittingpattern.Row.Row) – the new row the instruction is in.

exception knittingpattern.Instruction. **InstructionNotFoundInRow**
Bases: ValueError

This exception is raised if an instruction was not found in its row.

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

knittingpattern.Instruction. **ID = 'id'**
the id key in the specification

knittingpattern.Instruction. **TYPE = 'type'**
the type key in the specification

knittingpattern.Instruction. **KNIT_TYPE = 'knit'**
the type of the knit instruction

knittingpattern.Instruction. **PURL_TYPE = 'purl'**
the type of the purl instruction

knittingpattern.Instruction. **DEFAULT_TYPE = 'knit'**
the type of the instruction without a specified type

knittingpattern.Instruction. **COLOR = 'color'**
the color key in the specification

knittingpattern.Instruction. **NUMBER_OF_CONSUMED_MESHES = 'number of consumed meshes'**
the key for the number of meshes that a instruction consumes

knittingpattern.Instruction. **DEFAULT_NUMBER_OF_CONSUMED_MESHES = 1**
the default number of meshes that a instruction consumes

knittingpattern.Instruction. **NUMBER_OF_PRODUCED_MESHES = 'number of produced meshes'**
the key for the number of meshes that a instruction produces

knittingpattern.Instruction. **DEFAULT_NUMBER_OF_PRODUCED_MESHES = 1**
the default number of meshes that a instruction produces

knittingpattern.Instruction. **RENDER_Z = 'z'**
The key to look for the z-index inside the `render` specification. .. seealso:: `get_z()` , *DEFAULT_Z*

knittingpattern.Instruction. **RENDER = 'render'**
Instructions have a default specification. In this specification the key in *RENDER* points to configuration for rendering.

knittingpattern.Instruction. **DEFAULT_Z = 0**
The default z-index, see `get_z()` .

`InstructionLibrary` Module

Instructions have many attributes that do not need to be specified in each `knitting pattern set` .

This module provides the functionality to load default values for instructions from various locations.

class knittingpattern.InstructionLibrary. **InstructionLibrary**
Bases: `object`

This library can be used to look up default specification of instructions.

The specification is searched for by the type of the instruction.

__getitem__ (*instruction_type*)

Returns the specification for `instruction_type`

See also:

as_instruction()

__init__ ( )
Create a new *InstructionLibrary* without arguments.

Use *load* to load specifications.

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

add_instruction (*specification*)
Add an instruction specification

Parameters **specification** – a specification with a key *knittingpattern.Instruction.TYPE*

See also:

as_instruction()

as_instruction (*specification*)
Convert the specification into an instruction

Parameters **specification** – a specification with a key *knittingpattern.Instruction.TYPE*

The instruction is not added.

See also:

add_instruction()

load

Returns a loader that can be used to load specifications

Return type *knittingpattern.Loader.JSONLoader*

A file to load is a list of instructions in JSON format.

```
[
    {
        "type" : "knit",
        "another" : "attribute"
    },
    {
        "type" : "purl"
    }
]
```

loaded_types
The types loaded in this library.

Returns a list of types, preferably as `string`

Return type list

class knittingpattern.InstructionLibrary. **DefaultInstructions**
Bases: *knittingpattern.InstructionLibrary.InstructionLibrary*

The default specifications for instructions ported with this package

INSTRUCTIONS_FOLDER = 'instructions'
the folder relative to this module where the instructions are located

__init__ ( )
Create the default instruction library without arguments.

The default specifications are loaded automatically form this package.

knittingpattern.InstructionLibrary. **default_instructions** ( )

Returns a default instruction library

Return type *DefaultInstructions*

Warning: The return value is mutable and you should not add new instructions to it. If you would like to add instructions to it, create a new *DefaultInstructions* instance.

knittingpattern Module

Here you can find the set of knit instructions in rows.

A *knitting pattern set* consists of several *KnittingPatterns*. Their functionality can be found in this module.

class knittingpattern.KnittingPattern. **KnittingPattern** (*id_*, *name*, *rows*, *parser*)
Bases: object

Knitting patterns contain a set of instructions that form a pattern.

Usually you do not create instances of this but rather load a *knitting pattern set*.

__init__ (*id_*, *name*, *rows*, *parser*)
Create a new instance.

Parameters

- **id_** – the id of this pattern
- **name** – the human readable name of this pattern
- **rows** – a collection of rows of instructions
- **parser** (knittingpattern.Parser.Parser) – the parser to use to new content

See also:

knittingpattern.new_knitting_pattern()

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

add_row (*id_*)
Add a new row to the pattern.

Parameters **id_** – the id of the row

id
the identifier within a *set of knitting patterns*

instruction_colors
The colors of the instructions.

Returns the colors of the instructions listed in first appearance in knit order

Return type list

name
a human readable name

rows
a collection of rows that this pattern is made of

Usually this should be a *knittingpattern.IdCollection.IdCollection* of *knittingpattern.Row.Row*.

rows_in_knit_order ( )
Return the rows in the order that they should be knit.

Return type list

Returns the *rows* in the order that they should be knit

See also:

knittingpattern.walk

KnittingPatternSet Module

A set of knitting patterns that can be dumped and loaded.

class knittingpattern.KnittingPatternSet. **KnittingPatternSet** (*type_*, *version*, *patterns*, *parser*, *comment=None*)

Bases: object

This is the class for a set of knitting patterns.

The *knitting patterns* all have an id and can be accessed from here. It is possible to load this set of knitting patterns from various locations, see the *knittingpattern* module. You rarely need to create such a pattern yourself. It is easier to create the pattern by loading it from a file.

__init__ (*type_*, *version*, *patterns*, *parser*, *comment=None*)

Create a new knitting pattern set.

This is the class for a set of *knitting patterns* .

Parameters

- **type** (*str*) – the type of the knitting pattern set, see the *specification*.
- **version** (*str*) – the version of the knitting pattern set. This is not the version of the library but the version of the *specification*.
- **patterns** – a collection of patterns. This should be a *IdCollection* of *KnittingPatterns* .
- **comment** – a comment about the knitting pattern

__weakref__

list of weak references to the object (if defined)

add_new_pattern (*id_*, *name=None*)

Add a new, empty knitting pattern to the set.

Parameters

- **id_** – the id of the pattern
- **name** – the name of the pattern to add or if None , the id_ is used

Returns a new, empty knitting pattern

Return type *knittingpattern.KnittingPattern.KnittingPattern*

comment

The comment about the knitting pattern.

Returns the comment for the knitting pattern set or None, see *__init__()* .

first

The first element in this set.

Return type *knittingpattern.KnittingPattern.KnittingPattern*

patterns

The pattern contained in this set.

Returns the patterns of the knitting pattern, see *__init__()*

Return type *knittingpattern.IdCollection.IdCollection*

The patterns can be accessed by their id.

to_ayabpng()
Convert the knitting pattern to a png.

Returns a dumper to save this pattern set as png for the AYAB software

Return type *knittingpattern.convert.AYABPNGDumper.AYABPNGDumper*

Example:

```
>>> knitting_pattern_set.to_ayabpng().temporary_path()
"/the/path/to/the/file.png"
```

to_svg(*zoom*)
Create an SVG from the knitting pattern set.

Parameters **zoom** (*float*) – the height and width of a knit instruction

Returns a dumper to save the svg to

Return type *knittingpattern.Dumper.XMLDumper*

Example:

```
>>> knitting_pattern_set.to_svg(25).temporary_path(".svg")
"/the/path/to/the/file.svg"
```

type
The type of the knitting pattern.

Returns the type of the knitting pattern, see *__init__()*

Return type str

See also:

Knitting Pattern File Format Specification

version
The version of the knitting pattern specification.

Returns the version of the knitting pattern, see *__init__()*

Return type str

See also:

Knitting Pattern File Format Specification

Loader Module

One can load objects from different locations. This module provides functionality to load objects from different locations while preserving a simple interface to the consumer.

class knittingpattern.Loader.**JSONLoader**(*process=<function identity>*, *chooses_path=<function true>*)
Bases: *knittingpattern.Loader.ContentLoader*

Load an process JSON from various locations.

The *process* is called with an *object* as first argument: `process(object)`.

object (*object_*)
Processes an already loaded object.

Returns the result of the processing step

Parameters **object** – the object to be loaded

string (*string*)
Load an object from a string and return the processed JSON content

Returns the result of the processing step

Parameters **string** (*str*) – the string to load the JSON from

class knittingpattern.Loader. **ContentLoader** (*process=<function identity>*, *chooses_path=<function true>*)
Bases: *knittingpattern.Loader.PathLoader*

Load contents of files and ressources.

The *process* is called with a `string` as first argument: `process(string)`.

file (*file*)

Returns the processed result of the content of a file-like object.

Parameters **file** – the file-like object to load the content from. It should support the `read` method.

path (*path*)

Returns the processed result of a *path's* content.

Parameters **path** (*str*) – the path where to load the content from. It should exist on the local file system.

string (*string*)

Returns the processed result of a string

Parameters **string** (*str*) – the string to load the ocntent from

url (*url*, *encoding='UTF-8'*)
load and process the content behind a url

Returns the processed result of the *url's* content

Parameters

- **url** (*str*) – the url to retrieve the content from
- **encoding** (*str*) – the encoding of the retrieved content. The default encoding is UTF-8.

class knittingpattern.Loader. **PathLoader** (*process=<function identity>*, *chooses_path=<function true>*)
Bases: `object`

Load paths and folders from the local file system.

The *process* is called with a `path` as first argument: `process(path)`.

__init__ (*process=<function identity>*, *chooses_path=<function true>*)
Create a PathLoader object.

Parameters

- **process** – `process(path)` is called with the *path* to load. The result of `process` is returned to the caller. The default value is *identity()* , so the paths are returned when loaded.
- **chooses_path** – `chooses_path(path)` is called before `process` and returns `True` or `False` depending on whether a specific path should be loaded and passed to `process`.

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

choose_paths (*paths*)

Returns the paths that are chosen by *chooses_path()*

Return type list

chooses_path (*path*)

Returns whether the path should be loaded

Return type bool

Parameters **path** (*str*) – the path to the file to be tested

example (*relative_path*)
Load an example from the knitting pattern examples.

Parameters **relative_path** (*str*) – the path to load

Returns the result of the processing

You can use *knittingpattern.Loader.PathLoader.examples()* to find out the paths of all examples.

examples ( )
Load all examples form the examples folder of this packge.

Returns a list of processed examples

Return type list

Depending on *chooses_path()* some paths may not be loaded. Every loaded path is processed and returned part of the returned list.

folder (*folder*)
Load all files from a folder recursively.

Depending on *chooses_path()* some paths may not be loaded. Every loaded path is processed and returned part of the returned list.

Parameters **folder** (*str*) – the folder to load the files from

Return type list

Returns a list of the results of the processing steps of the loaded files

path (*path*)
load a *path* and return the processed result

Parameters **path** (*str*) – the path to the file to be processed

Returns the result of processing step

relative_file (*module*, *file*)
Load a file relative to a module.

Parameters

- **module** (*str*) – can be
  - a path to a folder
  - a path to a file
  - a module name
- **folder** (*str*) – the path of a folder relative to `module`

Returns the result of the processing

relative_folder (*module*, *folder*)

Load a folder located relative to a module and return the processed result.

Parameters

- **module** (*str*) – can be
  - a path to a folder
  - a path to a file
  - a module name
- **folder** (*str*) – the path of a folder relative to `module`

Returns a list of the results of the processing

Return type list

Depending on *chooses_path()* some paths may not be loaded. Every loaded path is processed and returned part of the returned list. You can use *choose_paths()* to find out which paths are chosen to load.

knittingpattern.Loader. **true** (_)

Returns `True`

Parameters _ – can be ignored

knittingpattern.Loader. **identity** (*object_*)

Returns the argument

Parameters **object_** – the object to be returned

Mesh Module

This module contains the meshes of the knit work.

class knittingpattern.Mesh. **Mesh**

Bases: `object`

A mesh that is either consumed or produced by an instruction.

```
assert mesh.is_produced() or mesh.is_consumed()
```

Since this is an abstract base class you will only get instances of *ProducedMesh* and *ConsumedMesh* .

__repr__ ( )

This mesh as string.

Returns the string representation of this mesh.

Return type str

This is useful for `print()` and class:*str*

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

as_consumed_mesh ( )
The consumed part to this mesh.

as_produced_mesh ( )
The produced part to this mesh.

If meshes are split up, it may be important which row the mesh is connected to afterwards. This method returns the mesh that is connected to the *producing row* .

If you got this mesh from `InstructionInRow.produced_meshes` or `Row.produced_meshes` , this returns the same object.

See also:

as_consumed_mesh() , `knittinpattern.Instruction.InstructionInRow.produced_meshes` , `knittinpattern.Row.Row.produced_meshes`

can_connect_to (*other*)
Whether a connection can be established between those two meshes.

connect_to (*other_mesh*)
Create a connection to an other mesh.

Warning: Both meshes need to be disconnected and one needs to be a consumed and the other a produced mesh. You can check if a connection is possible using *can_connect_to()* .

See also:

is_consumed() , *is_produced()* , *can_connect_to()*

consuming_instruction
Instruction which consumes this mesh.

Returns the instruction that consumes this mesh

Return type *knittingpattern.Instruction.InstructionInRow*

See also:

index_in_consuming_instruction , *consuming_row* , *producing_instruction*

Warning: Check with *is_consumed()* before!

consuming_row
Row which consumes this mesh.

Returns the row that consumes this mesh

Return type *knittingpattern.Row.Row*

See also:

index_in_consuming_row , *consuming_instruction* , *producing_row*

> **Warning:** Check with *is_consumed()* before!

disconnect ( )
Remove the connection between two rows through this mesh.

After disconnecting this mesh, it can be connected anew.

index_in_consuming_instruction
Index in instruction as consumed mesh.

Returns the index of the mesh in the list of meshes that *consuming_instruction* consumes

Return type int

```
instruction = mesh.consuming_instruction
index = mesh.index_in_consuming_instruction
assert instruction.consumed_meshes[index] == mesh
```

See also:

consuming_instruction, *index_in_consuming_instruction*

> **Warning:** Check with *is_consumed()* before!

index_in_consuming_row
Index in row as consumed mesh.

Returns the index of the mesh in the list of meshes that *consuming_row* consumes

Return type int

```
row = mesh.consuming_row
index = mesh.index_in_consuming_row
assert row.consumed_meshes[index] == mesh
```

See also:

consuming_row, *index_in_producing_row*

> **Warning:** Check with *is_consumed()* before!

index_in_producing_instruction
Index in instruction as a produced mesh.

Returns the index of the mesh in the list of meshes that *producing_instruction* produces

Return type int

```
instruction = mesh.producing_instruction
index = mesh.index_in_producing_instruction
assert instruction.produced_meshes[index] == mesh
```

See also:

producing_instruction, *index_in_consuming_instruction*

> **Warning:** Check with *is_produced()* before!

index_in_producing_row
Index in row as produced mesh.

Returns the index of the mesh in the *producing_row*

Return type int

```
row = mesh.producing_row
index = mesh.index_in_producing_row
assert row[index] == mesh
```

See also:

producing_row , *index_in_consuming_row*

> **Warning:** Check with *is_produced()* before!

is_connected ( )
Returns whether this mesh is already connected.

Returns whether this mesh is connected to an other.

Return type bool

is_connected_to (*other_mesh*)
Whether the one mesh is conencted to the other.

is_consumed ( )
Whether the mesh has an instruction that consumed it.

Returns whether the mesh is consumed by an instruction

Return type bool

If you get this mesh from *knittingpattern.Instruction.InstructionInRow.consumed_meshes* or *knittingpattern.Row.Row.consumed_meshes* , this should be `True` .

> **Warning:** Before you use any methods on how the mesh is consumed, you should check with `mesh.is_consumed()` .

is_knit ( )
Whether the mesh is produced by a knit instruction.

Returns whether the mesh is knit by an instruction

Return type bool

See also:

producing_instruction

is_mesh ( )
Whether this object is a mesh.

Returns `True`

Return type bool

is_produced ( )
Whether the mesh has an instruction that produces it.

Returns whether the mesh is produced by an instruction

Return type bool

If you get this mesh from *knittingpattern.Instruction.InstructionInRow.produced_meshes* or *knittingpattern.Row.Row.produced_meshes*, this should be `True`.

Warning: Before you use any methods on how the mesh is produced, you should check with `mesh.is_produced()`.

producing_instruction
Instruction which produces this mesh.

Returns the instruction that produces this mesh

Return type *knittingpattern.Instruction.InstructionInRow*

See also:

index_in_producing_instruction, *producing_row*, *consuming_row*

Warning: Check with *is_produced()* before!

producing_row
Row which produces this mesh.

Returns the row of the instruction that produces this mesh

Return type *knittingpattern.Row.Row*

See also:

index_in_producing_row, *producing_instruction*, *consuming_row*

Warning: Check with *is_produced()* before!

class knittingpattern.Mesh. **ProducedMesh** (*producing_instruction*, *index_in_producing_instruction*)
Bases: *knittingpattern.Mesh.Mesh*

A *Mesh* that has a producing instruction

__init__ (*producing_instruction*, *index_in_producing_instruction*)

Parameters

- **producing_instruction** – the *instruction* that produces the mesh
- **index_in_producing_instruction** (*int*) – the index of the mesh in the list of meshes that `producing_instruction` produces

Note: There should be no necessity to create instances of this directly. You should be able to use `instruction.produced_meshes` or `instruction.consumed_meshes` to access the *meshes*.

class knittingpattern.Mesh. **ConsumedMesh** (*consuming_instruction*, *index_in_consuming_instruction*)

Bases: *knittingpattern.Mesh.Mesh*

A mesh that is only consumed by an instruction

__init__ (*consuming_instruction*, *index_in_consuming_instruction*)

Parameters

- **consuming_instruction** – the *instruction* that consumes the mesh
- **index_in_consuming_instruction** (*int*) – the index of the mesh in the list of meshes that `consuming_instruction` consumes

---

Note: There should be no necessity to create instances of this directly. You should be able to use `instruction.produced_meshes` or `instruction.consumed_meshes` to access the *meshes*.

`Parser` Module

In this module you can find the parsing of knitting pattern structures.

class knittingpattern.Parser. **Parser** (*specification*)

Bases: `object`

Parses a knitting pattern set and anything in it.

__init__ (*specification*)

Create a parser with a specification.

Parameters **specification** – the types and classes to use for the resulting object structure, preferably a *knittingpattern.ParsingSpecification.ParsingSpecification*

__weakref__

list of weak references to the object (if defined)

instruction_in_row (*row*, *specification*)

Parse an instruction.

Parameters

- **row** – the row of the instruction
- **specification** – the specification of the instruction

Returns the instruction in the row

knitting_pattern_set (*values*)

Parse a knitting pattern set.

Parameters **value** (*dict*) – the specification of the knitting pattern set

Return type *knittingpattern.KnittingPatternSet.KnittingPatternSet*

Raises **knittingpattern.KnittingPatternSet.ParsingError** – if `value` does not fulfill the *specification*.

new_pattern (*id_*, *name*, *rows=None*)
Create a new knitting pattern.

If rows is `None` it is replaced with the *new_row_collection()* .

new_row (*id_*)
Create a new row with an id.

Parameters **id_** – the id of the row

Returns a row

Return type *knittingpattern.Row.Row*

new_row_collection ( )
Create a new row collection.

Returns a new specified row collection for the *knitting pattern*

knittingpattern.Parser. **ID = 'id'**
the id of a row, an instruction or a pattern

knittingpattern.Parser. **NAME = 'name'**
the name of a row

knittingpattern.Parser. **TYPE = 'type'**
the type of an instruction or the knitting pattern set

knittingpattern.Parser. **VERSION = 'version'**
the version of a knitting pattern set

knittingpattern.Parser. **INSTRUCTIONS = 'instructions'**
the instructions in a row

knittingpattern.Parser. **SAME_AS = 'same as'**
pointer to a inherit from

knittingpattern.Parser. **PATTERNS = 'patterns'**
the patterns in the knitting pattern set

knittingpattern.Parser. **ROWS = 'rows'**
the rows inside a pattern

knittingpattern.Parser. **CONNECTIONS = 'connections'**
the connections in a pattern

knittingpattern.Parser. **FROM = 'from'**
the position and row a connection comes from

knittingpattern.Parser. **TO = 'to'**
the position and row a connection goes to

knittingpattern.Parser. **START = 'start'**
the mesh index the connection starts at

knittingpattern.Parser. **DEFAULT_START = 0**
the default mesh index the connection starts at if none is given

knittingpattern.Parser. **MESHES = 'meshes'**
the number of meshes of a connection

knittingpattern.Parser. **COMMENT = 'comment'**
a comment of a row, an instruction, anything

exception knittingpattern.Parser. **ParsingError**
Bases: ValueError

Mistake in the provided object to parse.

This Error is raised if there is an error during the parsing for *Parser* .

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

knittingpattern.Parser. **default_parser** ( )
The parser with a default specification.

Returns a parser using a *knittingpattern.ParsingSpecification.DefaultSpecification*

Return type *knittingpattern.Parser.Parser*

ParsingSpecification Module

This modules specifies how to convert JSON to knitting patterns.

When parsing *knitting patterns* a lot of classes can be used.

The *ParsingSpecification* is the one place where to go to change a class that is used throughout the whole structure loaded by e.g. a *knittingpattern.Parser.Parser* . *new_knitting_pattern_set_loader()* is a convinient interface for loading knitting patterns.

These functions should do the same:

```
# (1) load from module
import knittingpattern
kp = knittingpattern.load_from_file("my_pattern")

# (2) load from knitting pattern
from knittingpattern.ParsingSpecification import *
kp = new_knitting_pattern_set_loader().file("my_pattern")
```

class knittingpattern.ParsingSpecification. **ParsingSpecification** (*new_loader=<class 'knittingpattern.Loader.JSONLoader'>, new_parser=<class 'knittingpattern.Parser.Parser'>, new_parsing_error=<class 'knittingpattern.Parser.ParsingError'>, new_pattern_set=<class 'knittingpattern.KnittingPatternSet.KnittingPatternS new_pattern_collection=<class 'knittingpattern.IdCollection.IdCollection'>, new_row_collection=<class 'knittingpattern.IdCollection.IdCollection'>, new_pattern=<class 'knittingpattern.KnittingPattern.KnittingPattern'>, new_row=<class 'knittingpattern.Row.Row'>, new_default_instructions=<class 'knittingpattern.InstructionLibrary.DefaultInstructio new_instruction_in_row=<class 'knittingpattern.Instruction.InstructionInRow'>*)

Bases: object

This is the specification for knitting pattern parsers.

The <knittingpattern.Parser.Parser> uses this specification to parse the knitting patterns. You can change every class in the data structure to add own functionality.

__init__ (*new_loader=<class 'knittingpattern.Loader.JSONLoader'>, new_parser=<class 'knittingpattern.Parser.Parser'>, new_parsing_error=<class 'knittingpattern.Parser.ParsingError'>, new_pattern_set=<class 'knittingpattern.KnittingPatternSet.KnittingPatternSet'>, new_pattern_collection=<class 'knittingpattern.IdCollection.IdCollection'>, new_row_collection=<class 'knittingpattern.IdCollection.IdCollection'>, new_pattern=<class 'knittingpattern.KnittingPattern.KnittingPattern'>, new_row=<class 'knittingpattern.Row.Row'>, new_default_instructions=<class 'knittingpattern.InstructionLibrary.DefaultInstructions'>, new_instruction_in_row=<class 'knittingpattern.Instruction.InstructionInRow'>*)

Create a new parsing specification.

__weakref__

list of weak references to the object (if defined)

knittingpattern.ParsingSpecification. **new_knitting_pattern_set_loader** (*specification=<knittingpatter*

Create a loader for a knitting pattern set.

Parameters **specification** – a *specification* for the knitting pattern set, default *DefaultSpecification*

class knittingpattern.ParsingSpecification. **DefaultSpecification**
Bases: *knittingpattern.ParsingSpecification.ParsingSpecification*

This is the default specification.

It is created like pasing no arguments to *ParsingSpecification*. The idea is to make the default specification easy to spot and create.

__init__ ( )
Initialize the default specification with no arguments.

classmethod __repr__ ( )
The string representation of the object.

Returns the string representation

Return type str

Prototype Module

This module contains the *Prototype* that can be used to create inheritance on object level instead of class level.

class knittingpattern.Prototype. **Prototype** (*specification*, *inherited_values=()*)
Bases: object

This class provides inheritance of its specifications on object level. Throughout this class *specification key* refers to a hashable object to look up a value in the specification.

__contains__ (*key*)
`key in prototype`

Parameters **key** – a *specification key*

Returns whether the key was found in the specification

Return type bool

__getitem__ (*key*)
`prototype[key]`

Parameters **key** – a *specification key*

Returns the value behind `key` in the specification

Raises **KeyError** – if no value was found

__init__ (*specification*, *inherited_values=()*)
create a new prototype

Parameters **specification** – the specification of the prototype. This specification can be inherited by other prototypes. It can be a `dict` or an other *knittingpattern.Prototype.Prototype* or anything else that supports *__contains__()* and *__getitem__()*

To look up a key in the specification it will be walked through

1. `specification`
2. `inherited_values` in order

However, new lookups can be inserted at before `inherited_values`, by calling *inherit_from()*.

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

get (*key*, *default=None*)

Returns the value behind `key` in the specification. If no value was found, `default` is returned.

Parameters **key** – a *specification key*

inherit_from (*new_specification*)

Inherit from a `new_specification`

Parameters **new_specification** – a specification as passed to *__init__()*

The `new_specification` is inserted before the first *inherited value*.

If the order is

1. *specification*
2. *inherited_values*

after calling `prototype.inherit_from(new_specification)` the lookup order is

1. *specification*
2. `new_specification`
3. *inherited_values*

Row Module

This module contains the rows of instructions of knitting patterns.

The *rows* are part of *knitting patterns*. They contain *instructions* and can be connected to other rows.

class knittingpattern.Row. **Row** (*row_id*, *values*, *parser*)

Bases: *knittingpattern.Prototype.Prototype*

This class contains the functionality for rows.

This class is used by *knitting patterns*.

__init__ (*row_id*, *values*, *parser*)

Create a new row.

Parameters

- **row_id** – an identifier for the row
- **values** – the values from the specification
- **inheriting_from** (*list*) – a list of specifications to inherit values from, see *knittingpattern.Prototype.Prototype*

Note: Seldomly, you need to create this row on your own. You can load it with the *knittingpattern* or the *knittingpattern.Parser.Parser*.

__repr__ ( )

The string representation of this row.

Returns a string representation of this row

Return type str

color
The color of the row.

Returns the color of the row as specified or `None`

consumed_meshes
Same as *produced_meshes* but for consumed meshes.

first_consumed_mesh
The first consumed mesh.

Returns the first consumed mesh

Return type *knittingpattern.Mesh.Mesh*

Raises **IndexError** – if no mesh is consumed

See also:

number_of_consumed_meshes

first_instruction
The first instruction of the rows instructions.

Return type *knittingpattern.Instruction.InstructionInRow*

Returns the first instruction in this row's *instructions*

first_produced_mesh
The first produced mesh.

Returns the first produced mesh

Return type *knittingpattern.Mesh.Mesh*

Raises **IndexError** – if no mesh is produced

See also:

number_of_produced_meshes

id
The id of the row.

Returns the id of the row

instruction_colors
The colors of the instructions in the row in the order tehy appear.

Returns a list of colors of the knitting pattern in the order that they appear in

Return type list

instructions
The instructions in this row.

Returns a collection of *instructions inside the row*

Return type ObservableList.ObservableList

last_consumed_mesh
The last consumed mesh.

Returns the last consumed mesh

Return type *knittingpattern.Mesh.Mesh*

Raises **IndexError** – if no mesh is consumed

See also:

number_of_consumed_meshes

last_instruction
The last instruction of the rows instructions.

Return type *knittingpattern.Instruction.InstructionInRow*

Returns the last instruction in this row's *instructions*

last_produced_mesh
The last produced mesh.

Returns the last produced mesh

Return type *knittingpattern.Mesh.Mesh*

Raises **IndexError** – if no mesh is produced

See also:

number_of_produced_meshes

number_of_consumed_meshes
The number of meshes that this row consumes.

Returns the number of meshes that this row consumes

Return type int

See also:

Instruction.number_of_consumed_meshes(), *number_of_produced_meshes()*

number_of_produced_meshes
The number of meshes that this row produces.

Returns the number of meshes that this row produces

Return type int

See also:

Instruction.number_of_produced_meshes(), *number_of_consumed_meshes()*

produced_meshes
The meshes that this row produces with its instructions.

Returns a collection of *meshes* that this instruction produces

rows_after
The rows that consume meshes from this row.

Return type list

Returns a list of rows that consume meshes from this row. Each row occurs only once. They are sorted by the first occurrence in the instructions.

rows_before
The rows that produce meshes for this row.

Return type list

Returns a list of rows that produce meshes for this row. Each row occurs only once. They are sorted by the first occurrence in the instructions.

knittingpattern.Row. **COLOR = 'color'**
the color of the row

utils Module

This module contains some useful functions.

The functions work on the standart library or are not specific to a certain existing module.

knittingpattern.utils. **unique** (*iterables*)
Create an iterable from the iterables that contains each element once.

Returns an iterable over the iterables. Each element of the result appeared only once in the result. They are oredered by the first occurrence in the iterables.

walk Module

Walk the knitting pattern.

knittingpattern.walk. **walk** (*knitting_pattern*)
Walk the knitting pattern in a right-to-left fashion.

Returns an iterable to walk the rows

Return type list

Parameters **knitting_pattern** (knittingpattern.KnittingPattern.KnittingPattern) – a knitting pattern to take the rows from

The knittingpattern.convert Module Reference

convert Module

Convert knitting patterns.

Usually you do not need to import this. Convenience functions should be available in the *knittingpattern* module.

color Module

Functions for color conversion.

knittingpattern.convert.color. **convert_color_to_rrggbb** (*color*)
The color in "#RRGGBB" format.

Returns the color in "#RRGGBB" format

AYABPNGBuilder Module

Convert knitting patterns to png files.

These png files are used to be fed into the ayab-desktop software. They only contain which meshes will be knit with a contrast color. They just contain colors.

class knittingpattern.convert.AYABPNGBuilder. **AYABPNGBuilder** (*min_x*, *min_y*, *max_x*, *max_y*, *default_color='white'*)

Bases: object

Convert knitting patterns to png files that only contain the color information and (x,y) coordinates. Througout this class the term *color* refers to either

- a valid html5 color name such as "black" , "white"
- colors of the form "#RGB" , "#RRGGBB" and "#RRRGGGBBB"

__init__ (*min_x*, *min_y*, *max_x*, *max_y*, *default_color='white'*)

Initialize the builder with the bounding box and a default color. min_x <= x < max_x and min_y <= y < max_y are the bounds of the instructions. Instructions outside the bounds are not rendered. Any Pixel that is not set has the *default_color* .

Parameters

- **min_x** (*int*) – the lower bound of the x coordinates
- **max_x** (*int*) – the upper bound of the x coordinates
- **min_y** (*int*) – the lower bound of the y coordinates
- **max_y** (*int*) – the upper bound of the y coordinates
- **default_color** – a valid *color*

__weakref__

list of weak references to the object (if defined)

default_color

Returns the *color* of the pixels that are not set

You can set this color by passing it to the *constructor* .

is_in_bounds (*x*, *y*)

Returns whether (x,y) is inside the *bounds*

Return type bool

set_color_in_grid (*color_in_grid*)

Set the pixel at the position of the color_in_grid to its color.

Parameters **color_in_grid** – must have the following attributes:

- color is the *color* to set the pixel to
- x is the x position of the pixel
- y is the y position of the pixel

See also:

set_pixel() , *set_colors_in_grid()*

set_colors_in_grid (*some_colors_in_grid*)

Same as *set_color_in_grid()* but with a collection of colors in grid.

Parameters **some_colors_in_grid** (*iterable*) – a collection of colors in grid for *set_color_in_grid()*

set_pixel (*x*, *y*, *color*)
set the pixel at `(x,y)` position to `color`

If `(x,y)` is out of the *bounds* this does not change the image.

See also:

set_color_in_grid()

write_to_file (*file*)
write the png to the file

Parameters **file** – a file-like object

AYABPNGDumper Module

Dump knitting patterns to PNG files compatible with the AYAB software.

class knittingpattern.convert.AYABPNGDumper. **AYABPNGDumper** (*function_that_returns_a_knitting_pattern_set*)
Bases: *knittingpattern.Dumper.file.ContentDumper*

This class converts knitting patterns into PNG files.

__init__ (*function_that_returns_a_knitting_pattern_set*)
Initialize the Dumper with a `function_that_returns_a_knitting_pattern_set`.

Parameters **function_that_returns_a_knitting_pattern_set** – a function that takes no arguments but returns a `knittinpattern.KnittingPatternSet.KnittingPatternSet`

When a dump is requested, the `function_that_returns_a_knitting_pattern_set` is called and the knitting pattern set is converted and saved to the specified location.

temporary_path (*extension='.png'*)
Saves the dump in a temporary file and returns its path.

> **Warning:** The user of this method is responsible for deleting this file to save space on the hard drive. If you only need a file object for a short period of time you can use the method `temporary_file()`.

Parameters **extension** (*str*) – the ending ot the file name e.g. `".png"`

Returns a path to the temporary file

Return type str

image_to_knittingpattern Module

This file lets you convert image files to knitting patterns.

knittingpattern.convert.image_to_knittingpattern. **convert_image_to_knitting_pattern** (*path*, *colors=('white', 'black')*)

Load a image file such as a png bitmap of jpeg file and convert it to a *knitting pattern file*.

Parameters

- **colors** (*list*) – a list of strings that should be used as *colors*.

- **path** (*str*) – ignore this. It is fulfilled by the loeder.

Example:

```
convert_image_to_knitting_pattern().path("image.png").path("image.json")
```

InstructionToSVG Module

This module maps instructions to SVG.

Use *default_instructions_to_svg()* to load the svg files provided by this package.

class knittingpattern.convert.InstructionToSVG. **InstructionToSVG**
Bases: object

This class maps instructions to SVGs.

__init__ ( )
create a InstructionToSVG object without arguments.

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

default_instruction_to_svg (*instruction*)
As *instruction_to_svg()* but it only takes the `default.svg` file into account.

In case no file is found for an instruction in *instruction_to_svg()* , this method is used to determine the default svg for it.

The content is created by replacing the text `{instruction.type}` in the whole svg file named `default.svg` .

If no file `default.svg` was loaded, an empty string is returned.

default_instruction_to_svg_dict (*instruction*)
Returns an xml-dictionary with the same content as *default_instruction_to_svg()*

If no file `default.svg` was loaded, an empty svg-dict is returned.

has_svg_for_instruction (*instruction*)

Returns whether there is an image for the instruction

Return type bool

This can be used before *instruction_to_svg()* as it determines whether

•the default value is used (`False`)

•or there is a dedicated svg representation (`True`).

instruction_to_svg (*instruction*)

Returns an SVG representing the instruction.

The SVG file is determined by the type attribute of the instruction. An instruction of type `"knit"` is looked for in a file named `"knit.svg"` .

Every element inside a group labeled `"color"` of mode `"layer"` that has a `"fill"` style gets this fill replaced by the color of the instruction. Example of a recangle that gets filled like the instruction:

```
<g inkscape:label="color" inkscape:groupmode="layer">
    <rect style="fill:#ff0000;fill-opacity:1;fill-rule:nonzero"
          id="rectangle1" width="10" height="10" x="0" y="0" />
</g>
```

If nothing was loaded to display this instruction, a default image is be generated by *default_instruction_to_svg()*.

instruction_to_svg_dict (*instruction*)

Returns an xml-dictionary with the same content as *instruction_to_svg()*.

load

Returns a loader object that allows loading SVG files from various sources such as files and folders.

Return type *knittingpattern.Loader.PathLoader*

Examples:

- `instruction_to_svg.load.path(path)` loads an SVG from a file named path
- `instruction_to_svg.load.folder(path)` loads all SVG files for instructions in the folder recursively. If multiple files have the same name, the last occurrence is used.

knittingpattern.convert.InstructionToSVG. **default_instructions_to_svg** ( )
load the default set of svg files for instructions

Returns the default svg files for the instructions in this package

Return type knittingpattern.InstructionToSVG.InstructionToSVG

knittingpattern.convert.InstructionToSVG. **DEFAULT_SVG_FOLDER = 'instruction-svgs'**
The name of the folder containing the svg files for the default instructions.

InstructionSVGCache Module

This module provides functionality to cache instruction SVGs.

class knittingpattern.convert.InstructionSVGCache. **InstructionSVGCache** (*instruction_to_svg=None*)
Bases: object

This class is a cache for SVG instructions.

If you plan too use only *instruction_to_svg_dict()*, you are save to replace a knittingpsttern.convert.InstructionToSVG.InstructionToSVG with this cache to get faster results.

__init__ (*instruction_to_svg=None*)
Create the InstructionSVGCache.

Parameters **instruction_to_svg** – an *InstructionToSVG* object. If None is given, the *default_instructions_to_svg* is used.

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

get_instruction_id (*instruction_or_id*)
The id that identifies the instruction in this cache.

Parameters **instruction_or_id** – an *instruction* or an instruction id

Returns a `hashable` object

Return type tuple

instruction_to_svg_dict (*instruction_or_id*, *copy_result=True*)
Return the SVG dict for the SVGBuilder.

Parameters

- **instruction_or_id** – the instruction or id, see *get_instruction_id()*
- **copy_result** (*bool*) – whether to copy the result

Return type dict

The result is cached.

to_svg (*instruction_or_id*, *i_promise_not_to_change_the_result=False*)
Return the SVG for an instruction.

Parameters

- **instruction_or_id** – either an *Instruction* or an id returned by *get_instruction_id()*
- **i_promise_not_to_change_the_result** (*bool*) –
  - `False` : the result is copied, you can alter it.
  - `True` : the result is directly from the cache. If you change the result, other calls of this function get the changed result.

Returns an SVGDumper

Return type *knittingpattern.Dumper.SVGDumper*

knittingpattern.convert.InstructionSVGCache. **default_instruction_svg_cache** ( )
Return the default InstructionSVGCache.

Return type *knittingpattern.convert.InstructionSVGCache.InstructionSVGCache*

knittingpattern.convert.InstructionSVGCache. **default_svg_cache** ( )
Return the default InstructionSVGCache.

Return type *knittingpattern.convert.InstructionSVGCache.InstructionSVGCache*

KnittingPatternToSVG Module

This module provides functionality to convert knitting patterns to SVG.

class knittingpattern.convert.KnittingPatternToSVG. **KnittingPatternToSVG** (*knittingpattern*, *layout*, *instruction_to_svg*, *builder*, *zoom*)

Bases: `object`

Converts a KnittingPattern to SVG.

This is inspired by the method object pattern, since building an SVG requires several steps.

__init__ (*knittingpattern*, *layout*, *instruction_to_svg*, *builder*, *zoom*)

Parameters

- **knittingpattern** (knittingpattern.KnittingPattern.KnittingPattern) – a knitting pattern
- **layout** (knittingpattern.convert.Layout.GridLayout) –
- **instruction_to_svg** – an *InstructionToSVG* :class:‘ ~knittingpattern.convert.InstructionToSVGCache.InstructionSVGCache‘, both with instructions already loaded.
- **builder** (knittingpattern.convert.SVGBuilder.SVGBuilder) –
- **zoom** (*float*) – the height and width of a knit instruction

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

build_SVG_dict ( )
Go through the layout and build the SVG.

Returns an xml dict that can be exported using a *XMLDumper*

Return type dict

knittingpattern.convert.KnittingPatternToSVG. **DEFINITION_HOLDER = 'g'**
Inside the svg, the instructions are put into definitions. The svg tag is renamed to the tag given in *DEFINITION_HOLDER*.

Layout Module

Map (x,y) coordinates to instructions

class knittingpattern.convert.Layout. **GridLayout** (*pattern*)
Bases: object

This class places the instructions at (x,y) positions.

__init__ (*pattern*)

Parameters **pattern** (knittingpattern.KnittingPattern.KnittingPattern) – the pattern to layout

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

bounding_box
The minimum and maximum bounds of this layout.

Returns (min_x,min_y,max_x,max_y) the bounding box of this layout

Return type tuple

row_in_grid (*row*)
The a RowInGrid for the row with position information.

Returns a row in the grid

Return type *RowInGrid*

walk_connections (*mapping=<function identity>*)
Iterate over connections between instructions.

Returns an iterator over *connections* between *instructions in grid*

Parameters **mapping** – funcion to map the result, see *walk_instructions()* for an example usage

walk_instructions (*mapping=<function identity>*)
Iterate over instructions.

Returns an iterator over *instructions in grid*

Parameters **mapping** – funcion to map the result

```
for pos, c in layout.walk_instructions(lambda i: (i.xy, i.color)):
    print("color {} at {}".format(c, pos))
```

walk_rows (*mapping=<function identity>*)
Iterate over rows.

Returns an iterator over `rows`

Parameters **mapping** – funcion to map the result, see *walk_instructions()* for an example usage

class knittingpattern.convert.Layout. **InstructionInGrid** (*instruction*, *position*)
Bases: *knittingpattern.convert.Layout.InGrid*

Holder of an instruction in the GridLayout.

__init__ (*instruction*, *position*)

Parameters

- **instruction** – an *instruction*
- **position** (Point) – the position of the *instruction*

color
The color of the instruction.

Returns the color of the *instruction*

instruction
The instruction.

Returns instruction that is placed on the grid

Return type *knittingpattern.Instruction.InstructionInRow*

class knittingpattern.convert.Layout. **Connection** (*start*, *stop*)
Bases: object

a connection between two *InstructionInGrid* objects

__init__ (*start*, *stop*)

Parameters

- **start** (InstructionInGrid) – the start of the connection
- **stop** (InstructionInGrid) – the end of the connection

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

is_visible ( )

Returns is this connection is visible

Return type bool

A connection is visible if it is longer that 0.

start

Returns the start of the connection

Return type *InstructionInGrid*

stop

Returns the end of the connection

Return type *InstructionInGrid*

knittingpattern.convert.Layout. **identity** (*object_*)

Returns the argument

class knittingpattern.convert.Layout. **Point** (*x*, *y*)

Bases: tuple

__getnewargs__ ( )

Return self as a plain tuple. Used by copy and pickle.

static __new__ (*_cls*, *x*, *y*)

Create new instance of Point(x, y)

__repr__ ( )

Return a nicely formatted representation string

x

Alias for field number 0

y

Alias for field number 1

knittingpattern.convert.Layout. **INSTRUCTION_HEIGHT = 1**

the default height of an instruction in the grid

class knittingpattern.convert.Layout. **InGrid** (*position*)

Bases: object

Base class for things in a grid

__init__ (*position*)

Create a new InGrid object.

__weakref__

list of weak references to the object (if defined)

bounding_box

The bounding box of this object.

Returns (min x, min y, max x, max y)

Return type tuple

height

Returns height of the object on the grid

Return type float

id

The id of this object.

row

Returns row of the object on the grid

Return type *knittingpattern.Row.Row*

width

Returns width of the object on the grid

Return type float

x

Returns x coordinate in the grid

Return type float

xy

Returns `(x,y)` coordinate in the grid

Return type tuple

y

Returns y coordinate in the grid

Return type float

yx

Returns `(y,x)` coordinate in the grid

Return type tuple

class knittingpattern.convert.Layout. **RowInGrid** (*row*, *position*)

Bases: *knittingpattern.convert.Layout.InGrid*

Assign x and y coordinates to rows.

__init__ (*row*, *position*)

Create a new row in the grid.

instructions

The instructions in a grid.

Returns the *instructions in a grid* of this row

Return type list

load_and_dump Module

convinience methods for conversion

Best to use *decorate_load_and_dump()* .

knittingpattern.convert.load_and_dump. **load_and_dump** (*create_loader*, *create_dumper*, *load_and_dump_*)

Returns a function that has the doc string of `load_and_dump_` additional arguments to this function are passed on to `load_and_dump_` .

Parameters

- **create_loader** – a loader, e.g. *knittingpattern.Loader.PathLoader*
- **create_dumper** – a dumper, e.g. *knittingpattern.Dumper.ContentDumper*

- **load_and_dump_** – a function to call with the loaded content. The arguments to both, `create_dumper` and, `create_loader` will be passed to `load_and_dump_` . Any additional arguments to the return value are also passed to `load_and_dump_` . The return value of `load_and_dump_` is passed back to the `Dumper` .

See also:

decorate_load_and_dump()

knittingpattern.convert.load_and_dump. **decorate_load_and_dump** (*create_loader*, *create_dumper*)

Same as *load_and_dump()* but returns a function to enable decorator syntax.

Examples:

```
@decorate_load_and_dump(ContentLoader, JSONDumper)
def convert_from_loader_to_dumper(loaded_stuff, other="arguments"):
    # convert
    return converted_stuff

@decorate_load_and_dump(PathLoader, lambda dump: ContentDumper(dump,
    encoding=None))
def convert_from_loader_to_dumper(loaded_stuff, to_file):
    # convert
    to_file.write(converted_stuff)
```

SVGBuilder Module

build SVG files

class knittingpattern.convert.SVGBuilder. **SVGBuilder**

Bases: object

This class builds an SVG to a file.

The class itself does not know what the objects look like. It offers a more convinient interface to build SVG files.

__init__ ( )

Initialize this object without arguments.

__weakref__

list of weak references to the object (if defined)

bounding_box

the bounding box of this SVG `(min_x,min_y,max_x,max_y)` .

```
svg_builder10x10.bounding_box = (0, 0, 10, 10)
assert svg_builder10x10.bounding_box == (0, 0, 10, 10)
```

`viewBox` , `width` and `height` are computed from this.

If the bounding box was never set, the result is a tuple of four `None` .

get_svg_dict ( )

Return the SVG structure generated.

insert_defs (*defs*)

Adds the defs to the SVG structure.

Parameters **defs** – a list of SVG dictionaries, which contain the defs, which should be added to the SVG structure.

place (*x*, *y*, *svg*, *layer_id*)
Place the `svg` content at `(x,y)` position in the SVG, in a layer with the id `layer_id` .

Parameters

- **x** (*float*) – the x position of the svg
- **y** (*float*) – the y position of the svg
- **svg** (*str*) – the SVG to place at `(x,y)`
- **layer_id** (*str*) – the id of the layer that this `svg` should be placed inside

place_svg_dict (*x*, *y*, *svg_dict*, *layer_id*, *group=None*)
Same as *place()* but with a dictionary as `svg_dict` .

Parameters

- **svg_dict** (*dict*) – a dictionary returned by xmltodict.parse()
- **group** (*dict*) – a dictionary of values to add to the group the `svg_dict` will be added to or `None` if nothing should be added

place_svg_use (*symbol_id*, *layer_id*, *group=None*)
Same as *place_svg_use_coords()* .

With implicit *x* and *y* which are set to *0* in this method and then *place_svg_use_coords()* is called.

place_svg_use_coords (*x*, *y*, *symbol_id*, *layer_id*, *group=None*)
Similar to *place()* but with an id as `symbol_id` .

Parameters

- **symbol_id** (*str*) – an id which identifies an svg object defined in the defs
- **group** (*dict*) – a dictionary of values to add to the group the use statement will be added to or `None` if nothing should be added

write_to_file (*file*)
Writes the current SVG to the `file` .

Parameters **file** – a file-like object

knittingpattern.convert.SVGBuilder. **SVG_FILE = '\n<svg\n xmlns:ns="http://PURL.org/dc/elements/1.1/"\n xmlr**
an empty svg file as a basis

The knittingpattern.Dumper Module Reference

Dumper Module

Writing objects to files

This module offers a unified interface to serialize objects to strings and save them to files.

class knittingpattern.Dumper. **ContentDumper** (*on_dump*, *text_is_expected=True*, *encoding='UTF-8'*)

Bases: `object`

This class is a unified interface for saving objects.

The idea is to decouple the place to save to from the process used to dump the content. We are saving several objects such as patterns and SVGs. They should all have the same convenient interface.

The process of saving something usually requires writing to some file. However, users may want to have the result as a string, an open file, a file on the hard drive on a fixed or temporary location, posted to some url or in a zip file. This class should provide for all those needs while providing a uniform interface for the dumping.

__init__ (*on_dump*, *text_is_expected=True*, *encoding='UTF-8'*)
Create a new dumper object with a function `on_dump`

Parameters

- **on_dump** – a function that takes a file-like object as argument and writes content to it.
- **text_is_expected** (*bool*) – whether to use text mode (`True` , default) or binary mode (`False`) for `on_dump` .

The dumper calls `on_dump` with a file-like object every time one of its save methods, e.g. *`string()`* or *`file()`* is called. The file-like object in the *`file`* argument supports the method `write()` to which the content should be written.

`text_is_expected` should be

- `True` to pass a file to `on_dump` that you can write strings to
- `False` to pass a file to `on_dump` that you can write bytes to

__repr__ ( )
the string representation for people to read

Returns the string representation of this object

Return type str

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

binary_file (*file=None*)
Same as *`file()`* but for binary content.

binary_temporary_file (*delete_when_closed=True*)
Same as *`temporary_file()`* but for binary mode.

bytes ( )

Returns the dump as bytes.

encoding

Returns the encoding for byte to string conversion

Return type str

file (*file=None*)
Saves the dump in a file-like object in text mode.

Parameters **file** – `None` or a file-like object.

Returns a file-like object

If *`file`* is `None` , a new `io.StringIO` is returned. If *`file`* is not `None` it should be a file-like object.

The content is written to the file. After writing, the file's read/write position points behind the dumped content.

path (*path*)
Saves the dump in a file named *path* .

Parameters **path** (*str*) – a valid path to a file location. The file can exist.

string ( )

Returns the dump as a string

temporary_binary_file (*delete_when_closed=True*)
Same as *temporary_file()* but for binary mode.

temporary_file (*delete_when_closed=True*)
Saves the dump in a temporary file and returns the open file object.

Parameters **delete_when_closed** (*bool*) – whether to delete the temporary file when it is closed.

Returns a file-like object

If `delete_when_closed` is `True` (default) the file on the hard drive will be deleted if it is closed or not referenced any more.

If `delete_when_closed` is `False` the returned temporary file is not deleted when closed or unreferenced. The user of this method has then the responsibility to free the space on the host system.

The returned file-like object has an attribute `name` that holds the location of the file.

temporary_path (*extension=''*)
Saves the dump in a temporary file and returns its path.

> **Warning:** The user of this method is responsible for deleting this file to save space on the hard drive. If you only need a file object for a short period of time you can use the method *temporary_file()* .

Parameters **extension** (*str*) – the ending ot the file name e.g. `".png"`

Returns a path to the temporary file

Return type str

class knittingpattern.Dumper. **JSONDumper** (*on_dump*)
Bases: *knittingpattern.Dumper.file.ContentDumper*

This class can be used to dump object s as JSON.

__init__ (*on_dump*)
Create a new JSONDumper object with the callable *on_dump*.

on_dump takes no arguments and returns the object that should be serialized to JSON.

knitting_pattern (*specification=None*)
loads a *knitting pattern* from the dumped content

Parameters **specification** – a *ParsingSpecification* or `None` to use the default specification

object ( )
Return the object that should be dumped.

class knittingpattern.Dumper. **XMLDumper** (*on_dump*)
Bases: *knittingpattern.Dumper.file.ContentDumper*

Used to dump objects as XML.

__init__ (*on_dump*)
Create a new XMLDumper object with the callable *on_dump*.

on_dump takes no aguments and returns the object that should be serialized to XML.

object ( )
Return the object that should be dumped.

class knittingpattern.Dumper. **SVGDumper** (*on_dump*)
Bases: *knittingpattern.Dumper.xml.XMLDumper*

This class dumps objects to SVG.

kivy_svg ( )
An SVG object.

Returns an SVG object

Return type kivy.graphics.svg.Svg

Raises **ImportError** – if the module was not found

file Module

Save strings to files.

class knittingpattern.Dumper.file. **ContentDumper** (*on_dump*, *text_is_expected=True*, *encoding='UTF-8'*)
Bases: object

This class is a unified interface for saving objects.

The idea is to decouple the place to save to from the process used to dump the content. We are saving several objects such as patterns and SVGs. They should all have the same convenient interface.

The process of saving something usually requires writing to some file. However, users may want to have the result as a string, an open file, a file on the hard drive on a fixed or temporary location, posted to some url or in a zip file. This class should provide for all those needs while providing a uniform interface for the dumping.

__init__ (*on_dump*, *text_is_expected=True*, *encoding='UTF-8'*)
Create a new dumper object with a function on_dump

Parameters

- **on_dump** – a function that takes a file-like object as argument and writes content to it.
- **text_is_expected** (*bool*) – whether to use text mode (True , default) or binary mode (False) for on_dump .

The dumper calls on_dump with a file-like object every time one of its save methods, e.g. *string()* or *file()* is called. The file-like object in the *file* argument supports the method write() to which the content should be written.

text_is_expected should be

- True to pass a file to on_dump that you can write strings to
- False to pass a file to on_dump that you can write bytes to

__repr__ ( )
the string representation for people to read

Returns the string representation of this object

Return type str

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

binary_file (*file=None*)
Same as *file()* but for binary content.

binary_temporary_file (*delete_when_closed=True*)
Same as *temporary_file()* but for binary mode.

bytes ( )

Returns the dump as bytes.

encoding

Returns the encoding for byte to string conversion

Return type str

file (*file=None*)
Saves the dump in a file-like object in text mode.

Parameters **file** – None or a file-like object.

Returns a file-like object

If *file* is None , a new io.StringIO is returned. If *file* is not None it should be a file-like object.

The content is written to the file. After writing, the file's read/write position points behind the dumped content.

path (*path*)
Saves the dump in a file named *path* .

Parameters **path** (*str*) – a valid path to a file location. The file can exist.

string ( )

Returns the dump as a string

temporary_binary_file (*delete_when_closed=True*)
Same as *temporary_file()* but for binary mode.

temporary_file (*delete_when_closed=True*)
Saves the dump in a temporary file and returns the open file object.

Parameters **delete_when_closed** (*bool*) – whether to delete the temporary file when it is closed.

Returns a file-like object

If delete_when_closed is True (default) the file on the hard drive will be deleted if it is closed or not referenced any more.

If delete_when_closed is False the returned temporary file is not deleted when closed or unreferenced. The user of this method has then the responsibility to free the space on the host system.

The returned file-like object has an attribute name that holds the location of the file.

temporary_path (*extension=''*)
Saves the dump in a temporary file and returns its path.

> **Warning:** The user of this method is responsible for deleting this file to save space on the hard drive. If you only need a file object for a short period of time you can use the method *temporary_file()* .

Parameters **extension** (*str*) – the ending ot the file name e.g. ".png"

Returns a path to the temporary file

Return type str

FileWrapper Module

This module provides wrappers for file-like objects for encoding and decoding.

class knittingpattern.Dumper.FileWrapper. **TextWrapper** (*binary_file*, *encoding*)
Bases: object

Use this class if you have a binary-file but you want to write strings to it.

__init__ (*binary_file*, *encoding*)
Create a wrapper around binary_file that encodes strings to bytes using encoding and writes them to binary_file .

Parameters

- **encoding** (*str*) – The encoding to use to transfer the written string to bytes so they can be written to binary_file
- **binary_file** – a file-like object open in binary mode

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

write (*string*)
Write a string to the file.

class knittingpattern.Dumper.FileWrapper. **BytesWrapper** (*text_file*, *encoding*)
Bases: object

Use this class if you have a text-file but you want to write bytes to it.

__init__ (*text_file*, *encoding*)
Create a wrapper around text_file that decodes bytes to string using encoding and writes them to text_file .

Parameters

- **encoding** (*str*) – The encoding to use to transfer the written bytes to string so they can be written to text_file
- **text_file** – a file-like object open in text mode

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

write (*bytes_*)
Write bytes to the file.

json Module

Dump objects to JSON.

class knittingpattern.Dumper.json. **JSONDumper** (*on_dump*)
Bases: *knittingpattern.Dumper.file.ContentDumper*

This class can be used to dump object s as JSON.

__init__ (*on_dump*)
Create a new JSONDumper object with the callable *on_dump*.

on_dump takes no arguments and returns the object that should be serialized to JSON.

knitting_pattern (*specification=None*)
loads a *knitting pattern* from the dumped content

Parameters **specification** – a *ParsingSpecification* or None to use the default specification

object ( )
Return the object that should be dumped.

svg Module

Dump objects to SVG.

class knittingpattern.Dumper.svg. **SVGDumper** (*on_dump*)
Bases: *knittingpattern.Dumper.xml.XMLDumper*

This class dumps objects to SVG.

kivy_svg ( )
An SVG object.

Returns an SVG object

Return type kivy.graphics.svg.Svg

Raises **ImportError** – if the module was not found

xml Module

Dump objects to XML.

class knittingpattern.Dumper.xml. **XMLDumper** (*on_dump*)
Bases: *knittingpattern.Dumper.file.ContentDumper*

Used to dump objects as XML.

__init__ (*on_dump*)
Create a new XMLDumper object with the callable *on_dump*.

on_dump takes no aguments and returns the object that should be serialized to XML.

object ( )
Return the object that should be dumped.

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

k

Symbols

A

B

C

D

J

K

L

M

N

O

P

R

S

T

U

V

W

X

Y