pyorg Documentation

Release 0.1

Jared Lumpe

Jan 23, 2022

Contents:

pyorg is a Python library for working with Org mode files and interacting with Org mode through Emacs itself.

This project, and especially the documentation, are a work in progress.

CHAPTER 1

Installation

1.1 Emacs dependencies

pyorg requires the ox-json package be installed in Emacs in order to be able to extract syntax trees from files.

1.2 Installing the package

Just clone the repository and run the setup script:

```
git clone https://github.com/jlumpe/pyorg
cd pyorg
python setup.py install
```

CHAPTER 2

Quick start

2.1 Getting the data from Emacs to Python

Create the following example file in Emacs:

```
#+title: Example file


* Header 1
Section 1

** Header 2
Section 2

*** Header 3
Section 3

**** Header 4
Section 4


* Markup
A paragraph with *bold*, /italic/, _underline_, +strike+, =verbatim=, and ~code~
objects.


* TODO [#A] A headline with a TODO and tags                          :tag1:tag2:
DEADLINE: <2019-06-29 Sat>
```

Use the `ox-json-export-to-json` command to export it as `example.json`. Now, read the JSON file with pyorg:

```
import json
from pyorg.io import org_doc_from_json
```

(continues on next page)

(continued from previous page)

```
with open('example.json') as f:
   data = json.load(f)

doc = org_doc_from_json(data)
```

2.2 Explore the AST structure

`doc` is an *`OrgDocument`* which contains all data read from the file. Its `root` attribute the root node of the AST:

```
>>> doc.root
OrgDataNode(type='org-data')
```

Its has the type `org-data`, which is always the root node of the buffer. Its contents are a `section` node and some more `headline` nodes:

```
>>> doc.root.contents
[OrgNode(type='section'),
 OrgOutlineNode(type='headline'),
 OrgOutlineNode(type='headline'),
 OrgOutlineNode(type='headline')]
```

We can print a simple representation of the outline tree with the *`dump_outline()`* method:

```
>>> doc.root.dump_outline()
Root
  0. Header 1
    0. Header 2
      0. Header 3
        0. Header 4
  1. Markup
  2. A headline with a TODO and tags
```

Get the 2nd headline (3rd item in root node's contents) and print the full AST subtree, along with each node's properties:

```
>>> hl2 = doc.root[2]
>>> hl2.dump(properties=True)
headline
  :archivedp       = False
  :commentedp      = False
  :footnote-section-p = False
  :level           = 1
  :post-affiliated = 120
  :post-blank      = 2
  :pre-blank       = 0
  :priority        = None
  :raw-value       = 'Markup'
  :tags            = []
  :title           = ['Markup']
  :todo-keyword    = None
  :todo-type       = None
  0 section
    :post-affiliated = 129
```

(continues on next page)

(continued from previous page)

```
    :post-blank      = 2
    0 paragraph
      :post-affiliated = 129
      :post-blank      = 0
      0 'A paragraph with '
      1 bold
        :post-blank      = 0
        0 'bold'
      2 ', '
      3 italic
        :post-blank      = 0
        0 'italic'
      4 ', '
      5 underline
        :post-blank      = 0
        0 'underline'
      6 ', '
      7 strike-through
        :post-blank      = 0
        0 'strike'
      8 ', '
      9 verbatim
        :post-blank      = 0
        :value           = 'verbatim'
      10 ', and '
      11 code
        :post-blank      = 0
        :value           = 'code'
      12 '\nobjects.\n'
```

Check third headline's properties to get the TODO information and tags:

```
>>> hl3 = doc.root[3]
>>> hl3.properties
{'title': ['A headline with a TODO and tags'],
 'deadline': OrgTimestampNode(type='timestamp'),
 'post-affiliated': 301,
 'commentedp': False,
 'archivedp': False,
 'footnote-section-p': False,
 'post-blank': 0,
 'todo-type': 'todo',
 'todo-keyword': 'TODO',
 'tags': ['tag1', 'tag2'],
 'priority': 65,
 'level': 1,
 'pre-blank': 0,
 'raw-value': 'A headline with a TODO and tags'}
```

CHAPTER 3

Org file structure

The contents of an org file are represented internally in Org mode as an Abstract Syntax Tree (AST). The nodes of this tree are org elements and objects, such as headings, paragraphs, blocks, and text formatting/markup constructs. See Org mode's documentation on the Element API for more detailed information.

3.1 The document

The *OrgDocument* class stores data and metadata for an entire Org document. The *OrgDocument.root* attribute stores the root of the document's AST (see *Outline structure*).

3.2 AST nodes

Nodes are all represented as instances of *OrgNode* or one of its subclasses. They have several key attributes:

type The node's type, such as `paragraph` or `list-item` (see below).

ref A unique string ID assigned by Org mode during the export process. Can be used to look up targets of internal links.

properties A dictionary of named properties that depends on the node's type. See Org mode's documentation on the Element API for a list of all properties by type. Some additional properties are also added by `ox-json` on export.

contents Ordered list of this node's AST children and text contents. Elements of the list are either *OrgNode* instances or strings.

keywords TODO

3.2.1 Node types

The *OrgNode.type* attribute is an instance of *OrgNodeType*. This is a `namedtuple` which stores the type's name as well as its properties as determined by the name's membership in the

`org-element-all-elements`, `org-element-all-objects`, `org-element-greater-elements`, `org-element-object-containers`, and `org-element-recursive-objects` variables in Emacs.

`pyorg.ast.ORG_NODE_TYPES` is a dictionary containing all node types defined by Org mode, keyed by name.

3.2.2 Specialized `OrgNode` subclasses

Outline structure

An org document is structured as an outline tree, which is made of nested headline elements. In Org mode, the root of the parse tree (and therefore the outline tree) is a special element with type `org-data`. All other outline nodes correspond to `headline` elements. In pyorg these are represented with the specialized classes *`OrgDataNode`* and *`OrgHeadlineNode`*, both of which inherit from the abstract base class *`OrgOutlineNode`*.

The contents of an outline node always consist of an optional `section` element followed by zero or more `headline` elements. For convenience these are also stored in the *`OrgOutlineNode.section`* and *`OrgOutlineNode.subheadings`* attributes.

You can use the *`OrgOutlineNode.dump_outline`* method to print a simple representation of an outline node's subtree:

```
>>> mydocument.root.dump_outline()

Root
0. Header for section one
 0. Header for subsection 1.1
   0. Header 1.1.1
 1. Header 1.2
1. These are the header's title text
2. Section three...
```

Timestamps

See *`OrgTimestampNode`*

Tables

See *`OrgTableNode`*

CHAPTER 4

Reading in Org file data

The main function of this package is to read in Org mode documents as Abstract Syntax Trees (ASTs) where they can be processed and converted/exported into other formats. See the documentation for the org element API for more information about the AST structure.

4.1 Reading from JSON export

Rather attempting to parse .org files directly, pyorg is designed to work with the output of the ox-json Emacs package. This simply export the AST generated by the `org` package itself to machine-readable JSON format. This has the advantage of also including all of your personal Org mode setting and customization in Emacs (such as link abbreviations).

4.2 Parsing Org files directly

pyorg has very limited capability to parse .org files without the help of Emacs. See the *pyorg.parse* module.

CHAPTER 5

Interfacing directly with Org mode

5.1 High-level interface to Org mode

pyorg.interface.Org

5.2 Communicating with Emacs

5.2.1 Emacs interface

pyorg.emacs.Emacs

5.2.2 Representing elisp code in Python

pyorg.elisp

CHAPTER 6

Converting org file data to other formats

6.1 Plain text

pyorg.convert.plaintext.to_plaintext()

6.2 HTML

pyorg.convert.html.converter.to_html()

6.3 JSON

pyorg.convert.json.to_json()

6.4 Creating your own converters

Subclass *pyorg.convert.base.OrgConverterBase*.

CHAPTER 7

The agenda

Support for the agenda is a work in progress. See `pyorg.agenda`.

CHAPTER 8

pyorg

8.1 pyorg package

8.1.1 Subpackages

pyorg.convert package

Subpackages

pyorg.convert.html package

Submodules

pyorg.convert.html.converter module

class pyorg.convert.html.converter.**OrgHtmlConverter**(*config=None*, ***kw*)

Bases: *pyorg.convert.base.OrgConverterBase*

DEFAULT_CONFIG = {'date_format': '%Y-%m-%d %a', 'image_extensions': ('.png', '.jpg',

DEFAULT_RESOLVE_LINK = {'http': True, 'https': True}

INLINE_NODES = frozenset({'link', 'line-break', 'footnote-reference', 'macro', 'code',

TAGS = {'babel-call': None, 'bold': 'strong', 'center-block': 'div', 'code': 'code

convert(*node*, *dom=False*, ***kwargs*)

Convert org node to HTML.

Parameters

- **node** (pyorg.ast.OrgNode) – Org node to convert.
- **dom** (*bool*) – Return HTML element instead of string.

Returns

Return type str or *HtmlElement*

default_classes(*type*)

default_tag(*type_*)

make_headline_text(*node*, *ctx=None*, *dom=False*)
Make HTML element for text content of headline node.

resolve_link(*linktype*, *raw*, *path*, *ctx=None*)
Resolve link into a proper URL.

pyorg.convert.html.converter.**to_html**(*node*, *dom=False*, ***kwargs*)
Convert org node to HTML.

Parameters

- **node** (pyorg.ast.OrgNode) – Org node to convert.
- **dom** (*bool*) – Return HTML element instead of string.
- **kwargs** – Keyword arguments to *OrgHtmlConverter* constructor.

Returns

Return type str or *HtmlElement*

pyorg.convert.html.element module

class pyorg.convert.html.element.**HtmlElement**(*tag*, *children=None*, *attrs=None*, *inline=False*, *post_ws=False*)

Bases: object

Lightweight class to represent an HTML element.

tag
HTML tag name (minus angle brackets).

Type str

children
List of child elements (HtmlElement or strings).

Type list

attrs
Mapping from attributes names (strings) to values (strings or bools).

Type dict

inline
Whether to render children in an inline context. If False each child will be rendered on its own line. If True whitespace will only be added before/after children according to the *post_ws* attribute of the child.

Type bool

classes
List of class names present in the "class" attribute. Assignable property.

Type list

post_ws
Whether to add whitespace after the tag when rendering in an inline context.

Type bool

add_class(*classes*)

classes

class pyorg.convert.html.element.**TextNode**(*text*, *post_ws=False*)
Bases: object

Text node to be used within HTML.

text
Wrapped text.

Type str

post_ws
Whether to add whitespace after the tag when rendering in an inline context.

Type bool

pyorg.convert.html.element.**html_to_string**(*elem*, ***kwargs*)

pyorg.convert.html.element.**write_html**(*stream*, *elem*, *indent='\t'*, *inline=False*)

Module contents

Export org mode AST nodes to HTML.

Submodules

pyorg.convert.base module

class pyorg.convert.base.**OrgConverterBase**(*config=None*, ***kw*)
Bases: object

Abstract base class for objects which convert org mode AST to another format.

config

Type dict

DEFAULT_CONFIG = {'date_format': '%Y-%m-%d %a', 'image_extensions': ('.png', '.jpg',

convert(*node*, ***kwargs*)

pyorg.convert.json module

Convert org mode AST nodes to JSON.

class pyorg.convert.json.**OrgJsonConverter**(*config=None*, ***kw*)
Bases: *pyorg.convert.base.OrgConverterBase*

DEFAULT_CONFIG = {'date_format': '%Y-%m-%d %a', 'image_extensions': ('.png', '.jpg',

make_object(*type_*, *data*)

pyorg.convert.json.**to_json**(*node*, ***kwargs*)

pyorg.convert.plaintext module

class pyorg.convert.plaintext.**OrgPlaintextConverter**(*config=None*, ***kw*)
Bases: *pyorg.convert.base.OrgConverterBase*

convert_multi(*items*, *blanks=False*, *sep=None*)

pyorg.convert.plaintext.**to_plaintext**(*arg*, *blanks=False*, *sep=None*, ***kwargs*)

Module contents

Convert org AST to other formats.

8.1.2 Submodules

pyorg.agenda module

pyorg.ast module

Work with org file abstract syntax trees.

See https://orgmode.org/worg/dev/org-syntax.html for a description of the org syntax.

class pyorg.ast.**DispatchNodeType**(*default*, *registry=None*, *doc=None*)
Bases: *pyorg.util.SingleDispatchBase*

Generic function which dispatches on the node type of its first argument.

format_key(*key*)

get_key(*node*)
Get the key to look up the correct implementation for the given argument.

pyorg.ast.**NODE_CLASSES = {'headline': <class 'pyorg.ast.OrgHeadlineNode'>, 'org-data': <**
Mapping from org element/node types to their Python class

pyorg.ast.**ORG_NODE_TYPES = {'babel-call': OrgNodeType('babel-call'), 'bold': OrgNodeType**
Mapping from names of all AST node types to *OrgNodeType* instances.

class pyorg.ast.**OrgDataNode**(*type_*, **args*, ***kw*)
Bases: *pyorg.ast.OrgOutlineNode*

Root node for an org mode parse tree.

Doesn't do anything special, aside from being the outline node at level 0.

class pyorg.ast.**OrgDocument**(*root*, *properties=None*, *meta=None*)
Bases: object

Represents an entire Org mode document.

root
The root of the document's Abstract Syntax Tree.

Type *OrgOutlineNode*

properties
Additional file-level properties attached to the document, such as the author or date. Values may be strings or secondary strings.

Type dict

meta
A dictionary containing arbitrary application-specific metadata.

Type dict

assign_header_ids(*depth=3*)
Assign unique IDs to headers.

class pyorg.ast.**OrgHeadlineNode**(*type_*, **args*, *title=None*, *id=None*, ***kw*)
Bases: *pyorg.ast.OrgOutlineNode*

Org header element.

title
Title of headline as plain text.

Type str

id
Unique ID for TOC tree.

Type str

has_todo
Whether this outline has a TODO keyword.

Type bool

priority_chr
Priority character if headline with priority, otherwise None.

Type str

scheduled
The timestamp in the "scheduled" property of the headline, if present.

Type *OrgTimestamp*

deadline
The timestamp in the "deadline" property of the headline, if present.

Type *OrgTimestamp*

closed
The timestamp in the "closed" property of the headline, if present.

Type *OrgTimestamp*

closed

deadline

has_todo

priority_chr

scheduled

class pyorg.ast.**OrgNode**(*type_*, *properties=None*, *contents=None*, *keywords=None*, *ref=None*, *meta=None*)
Bases: object

A node in an org file abstract syntax tree.

Implements the sequence protocol as a sequence containing its child nodes (identically to *contents*). Also allows accessing property values by indexing with a string key.

type
Node type, obtained from *org-element-type*.

Type *OrgNodeType*

properties
Dictionary of property values, obtained from *org-element-property*.

Type dict

contents
List of contents (org nodes or strings), obtained from *org-element-contents*.

Type list

ref
A unique ID assigned to the node during the export process.

Type str

keywords
Dictionary of keyword values.

Type dict

meta
A dictionary containing arbitrary application-specific metadata.

Type dict

is_outline
Whether this node is an outline node.

Type bool

children
Iterator over all child AST nodes (in contents or keyword/property values.

descendants(*incself=False*, *properties=False*)
Recursively iterate over all of the node's descendants.

Parameters

- **incself** (*bool*) – Include self.
- **properties** (*bool*) – Include children in the node's properties, not just *contents* (see *children*).

Yields .*OrgNode*

dump(*properties=False*, *indent=' '*)
Print a debug representation of the node and its descendants.

Parameters

- **value** (OrgNode) –
- **properties** (*bool*) – Also print node properties.
- **indent** (*str*) –
- **to indent with.** (*Characters*) –

is_outline = False

class pyorg.ast.**OrgNodeType**
Bases: *pyorg.ast.OrgNodeType*

The properties of an org AST node type.

name
The unique name of this node type.

Type str

is_element
Whether this node type is an element. "An element defines syntactical parts that are at the same level as a paragraph, i.e. which cannot contain or be included in a paragraph."

Type bool

is_object
Whether this node type is an object. All nodes which are not elements are objects. "An object is a part that could be included in an element."

Type bool

is_greater_element
Whether this node type is a greater element. "Greater elements are all parts that can contain an element."

Type bool

is_recursive
Whether this node type is a recursive object.

Type bool

is_object_container
Whether this node type is an object container, i.e. can directly contain objects.

Type bool

References

Org Syntax

is_object

class pyorg.ast.**OrgOutlineNode**(*type_*, *properties=None*, *contents=None*, *keywords=None*, *ref=None*, *meta=None*)
Bases: *pyorg.ast.OrgNode*

Abstract base class for org node that is a component of the outline tree.

Corresponds to the root org-data node or a headline node.

level
Outline level. 0 corresponds to the root node of the file.

Type int

section
Org node with type *"section"* that contains the outline node's direct content (not part of any nested outline nodes).

Type *OrgNode*

subheadings
List of nested headings.

Type list

dump_outline(*depth=None*, *indent=' '*)
Print representation of node's outline subtree.

Parameters

- **depth** (*int*) – Maximum depth to print.
- **indent** (*str*) – String to indent with.

is_outline = True

outline_tree()
Create a list of `(child, child_tree)` pairs.

section

subheadings

class pyorg.ast.**OrgTableNode**(*type_*, *properties=None*, *contents=None*, *keywords=None*, *ref=None*, *meta=None*)
Bases: *pyorg.ast.OrgNode*

An org node with type "table".

rows
List of standard rows.

Type list of OrgNode

nrows
Number of (non-rule) rows in table. This includes the header.

Type int

ncols
Number of columns in table.

Type int

blocks()
Standard rows divided into "blocks", which were separated by rule rows.

Returns

Return type list of list of OrgNode

cells()

ncols

nrows

rows

class pyorg.ast.**OrgTimestamp**(*tstype*, *start*, *end=None*, *repeater=None*, *warning=None*)
Bases: object

Stores Org mode timestamp data, without the whole AST node.

tstype

Type str

start

Type datetime.datetime

end

Type datetime.datetime

repeater

Type *OrgTimestampInterval*

warning

Type *OrgTimestampInterval*

interval

is_range

class pyorg.ast.**OrgTimestampInterval**(*type_*, *unit*, *value*)
Bases: object

An interval of time stored in an Org mode time stamp's repeater or warning.

type

Type str

unit

Type str

value

Type float

class pyorg.ast.**OrgTimestampNode**(*type_*, **args*, ***kwargs*)
Bases: *pyorg.ast.OrgNode*, *pyorg.ast.OrgTimestamp*

An org node with type "timestamp".

pyorg.ast.**as_node_type**(*t*)
Convert to node type object, looking up strings by name.

pyorg.ast.**as_secondary_string**(*obj*)
Convert argument to a "secondary string" (list of nodes or strings).

Parameters **obj** (*OrgNode or str or list*) –

Returns

Return type list

Raises TypeError : if obj is not a str or *OrgNode* or iterable of these.

pyorg.ast.**dispatch_node_type**(*parent=None*)
Decorator to create DispatchNodeType instance from default implementation.

pyorg.ast.**dump_ast**(*value*, *properties=False*, *indent=' '*, *_level=0*)
Print a debug representation of an org AST node and its descendants.

Parameters

- **value** (OrgNode) –
- **properties** (*bool*) – Also print node properties.
- **indent** (*str*) – Characters to indent with.

pyorg.ast.**get_node_type**(*obj*, *name=False*)
Get type of AST node, returning None for other Python types.

pyorg.ast.**node_cls**(*type_*)
Register a node class for a particular type in *NODE_CLASSES*.

pyorg.interface module

class pyorg.interface.**Org**(*emacs*, *orgdir=None*, *loader=None*)
Bases: object

Interface to org mode.

emacs

Type pyorg.emacs.Emacs

orgdir
Directory org files are read from.

Type pyorg.files.OrgDirectory

loader
Loader used to read .org file data.

Type pyorg.files.OrgFileLoader

open_org_file(*file*, *focus=False*)
Open an org file in the org directory for editing in Emacs.

Parameters

- **file** (*str or pathlib.Path*) – Path to file to open. If not absolute it is taken to be relative to *orgdir*.
- **focus** (*bool*) – Switch window system focus to the active Emacs frame.

Raises

- emacs.emacs.EmacsException
- FileNotFoundError

read_org_file(*file*, *raw=None*)
Read and parse an org file.

Parameters

- **file** (*str or pathlib.Path*) – Path to file to load (relative paths are interpreted relative to org directory).
- **raw** (*bool*) – Don't parse and just return raw JSON exported from Emacs.

Returns

Return type *pyorg.ast.OrgDocument*

Raises

- emacs.emacs.EmacsException
- FileNotFoundError

read_org_file_direct(*file*, *raw=False*)
Read and parse an org file directly from Emacs.

Always reads the current file and does not use cached data, or perform any additional processing other than parsing.

Parameters

- **file** (*str or pathlib.Path*) – Path to file to load (relative paths are interpreted relative to org directory).
- **raw** (*bool*) – Don't parse and just return raw JSON exported from Emacs.

Returns

Return type *pyorg.ast.OrgDocument* or dict

Raises

- emacs.emacs.EmacsException
- FileNotFoundError

pyorg.io module

Read (and write) org mode data from JSON and other formats.

pyorg.io.**org_doc_from_json**(*data*)
Parse an ORG document from exported JSON data.

Returns

Return type *OrgDocument*

pyorg.io.**org_node_from_json**(*data*)
Parse an org AST node from JSON data.

Returns

Return type *OrgNode*

pyorg.parse module

(Partially) parse org files.

pyorg.parse.**parse_tags**(*string*)
Parse tags from string.

Parameters **string** (*str*) – Tags separated by colons.

Returns List of tags.

Return type list[str]

pyorg.parse.**read_file_keywords**(*file*)
Read file-level keywords from an .org file (without using Emacs).

Limitations: only reads up to the first element in the initial section (excluding comments). If the initial section does contain such an element, any keywords directly preceding it (not separated with a blank line) will be considered affiliated keywords of that element and ignored.

Will not parse org markup in keyword values.

All keys are converted to uppercase.

Keys which appear more than once will have values in a list.

Parameters **file** – String or open file object or stream in text mode.

Returns

Return type dict

pyorg.util module

Misc. utility code.

class pyorg.util.**Namespace**(*_map=None*, ***kwargs*)
Bases: object

A simple collection of attribute values, that supports inheritance.

Meant to be used to pass large sets of arguments down through recursive function calls in a way that they can be overridden easily.

Public attributes and methods start with an underscore so as not to interfere with the namespace.

_map
Stores the underlying data.

Type collections.ChainMap

class pyorg.util.**SingleDispatch**(*default*, *registry=None*, *doc=None*)
Bases: *pyorg.util.SingleDispatchBase*

Generic function which dispatches on the type of its first argument.

iter_keys(*arg*)

validate_key(*key*)
Validate and possibly replace a key before an implementation is registered under it.

Default implementation simply returns the argument. Subclasses may wish to override this. An error should be raised for invalid keys.

Parameters **key** – Key passed to register().

Returns

Return type Key to use for registration, which may be different than argument.

class pyorg.util.**SingleDispatchBase**(*default*, *registry=None*, *doc=None*)
Bases: abc.ABC

ABC for a generic function which dispatches on some trait of its first argument.

May be bound to an object or class as a method.

Concrete subclasses must implement one of the *get_key()* or *iter_keys()* method.

default
Default implementation.

Type callable

registry
Stores the specialized implementation functions by key.

Type dict

bind(*instance*, *owner=None*)
Get a version of the function bound to the given instance as a method.

Parameters

- **instance** – Object instance to bind to.

- **owner** –

copy()

dispatch(*arg*)
Get the actual function implementation for the given argument.

get_key(*arg*)
Get the key to look up the correct implementation for the given argument.

iter_keys(*arg*)

register(*key*, *impl=None*)
Register an implementation for the given key.

Parameters

- **key** – Key to register method under. May also be a list of keys.
- **impl** (*callable*) – Implementation to register under the given key(s). If None will return a decorator function that completes the registration.

Returns None if `method` is given. Otherwise returns a decorator that will register the function it is applied to.

Return type function or None

validate_key(*key*)
Validate and possibly replace a key before an implementation is registered under it.

Default implementation simply returns the argument. Subclasses may wish to override this. An error should be raised for invalid keys.

Parameters **key** – Key passed to *register()*.

Returns

Return type Key to use for registration, which may be different than argument.

class pyorg.util.**SingleDispatchMethod**(*func*, *instance*, *owner=None*)
Bases: object

Version of a *SingleDispatchBase* which acts as a method.

func

Type *SingleDispatchBase*

instance
Instance the function is bound to, or None.

owner

default

dispatch(*arg*)

class pyorg.util.**TreeNamespace**(*_map=None*, *_path=()*, ***kwargs*)
Bases: *pyorg.util.Namespace*

Namespace with a `_path` attribute that marks its location in a tree structure.

_path

Type tuple

pyorg.util.**parse_iso_date**(*string*)
Parse date or datetime from an ISO 8601 date string.

Parameters `string` –

Returns Return time varies based on whether the string includes a time component.

Return type datetime.date or datetime.datetime

8.1.3 Module contents

Root package for pyorg.

Package for working with Emacs org-mode files

CHAPTER 9

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

Symbols

A

B

C

D

E

F

R

S

T

U

V

W