bugyi.lib

unknown

Dec 20, 2021

TABLE OF CONTENTS

Overly general Python library used when no other library is a good fit.

CHAPTER

ONE

LIB

1.1 lib package

Overly general Python library used when no other library is a good fit.

1.1.1 Submodules

lib.core module

Functions/classes that we were unable to match to a category are placed in this module. Thus, you should only add code to this module when you are unable to find ANY other module to add it to.

> **Warning:** This module probably shouldn't be imported from other modules in this package.

lib.dates module

Helper functions/classes related to dates/datetimes.

`parse_date`(*date*)

Parses a date string.

Parameters `date` (`Union`[`str`, `date`, `datetime`]) – The date string to parse.

This function defaults to using dateutil.parser.parse(), but also accepts the following special formats for `date`:

@t | @today Today's date.

Nd N days ago.

Nw N weeks ago.

Return type `date`

`parse_daterange`(*daterange*)

Transforms a date range specifier into a list of sequential dates.

Parameters `daterange` (`str`) – A string of the form START_DATE:END_DATE where both START_DATE and END_DATE are any value that the parse_date() function can parse.

Return type `List`[`date`]

lib.errors module

Custom error handling code lives here.

class BErr(*emsg*, *cause=None*, *up=0*)
Bases: `result.result.Err[lib.types.T, BugyiError]`

Bugyi Err Type.

Parameters

- **emsg** (`str`) –
- **cause** (`Optional[Exception]`) –
- **up** (`int`) –

exception BugyiError(*emsg*, *cause=None*, *up=0*)
Bases: `Exception`

Custom general-purpose exception.

Parameters

- **emsg** (`str`) –
- **cause** (`Optional[Exception]`) –
- **up** (`int`) –

report(*width=80*)
Return an _ErrorReport object formatting the current state of this BugyiError

Parameters **width** (`int`) –

Return type `_ErrorReport`

chain_errors(*e1*, *e2*)
Chain two exceptions together.

This is the functional equivalent to `raise e1 from e2`.

Parameters

- **e1** (*~E*) – An exception.
- **e2** (`Optional[Exception]`) – The exception we want to chain to `e2`.

Return type ~E

Returns `e1` after chaining `e2` to it.

lib.io module

Helper utilities related to IO.

box(*title*)
Wraps @title in a pretty ASCII box.

Parameters **title** (`str`) –

Return type `str`

class colors
Bases: `object`

Namespace for <color>() functions.

black()

Parameters **msg** (str) –

Return type str

blue()

Parameters **msg** (str) –

Return type str

cyan()

Parameters **msg** (str) –

Return type str

green()

Parameters **msg** (str) –

Return type str

magenta()

Parameters **msg** (str) –

Return type str

red()

Parameters **msg** (str) –

Return type str

white()

Parameters **msg** (str) –

Return type str

yellow()

Parameters **msg** (str) –

Return type str

confirm(*prompt*)

Prompt user for 'y' or 'n' answer.

Return type bool

Returns True iff the user responds to the @prompt with 'y'.

Parameters **prompt** (str) –

copy_to_clipboard(*clip*)

Copys a clip to the system clipboard.

Parameters **clip** (str) – The clip that gets copied into the clipboard.

Return type None

create_dir(*directory*)
Create directory if it does not already exist.

Parameters **directory** (str) – The full directory path.

Return type None

efill(*multiline_msg*, *width=80*, *indent=0*)
A better version of textwrap.fill().

Parameters

- **multiline_msg** (str) –
- **width** (int) –
- **indent** (int) –

Return type str

ewrap(*multiline_msg*, *width=80*, *indent=0*)
A better version of textwrap.wrap().

Parameters

- **multiline_msg** (str) –
- **width** (int) –
- **indent** (int) –

Return type Iterator[str]

getch(*prompt=None*)
Reads a single character from stdin.

Parameters **prompt** (Optional[str]) – prompt that is presented to user.

Return type str

Returns The single character that was read.

mkfifo(*fifo_path*)
Creates named pipe if it does not already exist.

Parameters **fifo_path** (str) – The full file path where the named pipe will be created.

Return type None

notify(**args*, *title=None*, *urgency=None*, *up=0*)
Sends desktop notification with calling script's name as the notification title.

Parameters

- ***args** – Arguments to be passed to the notify-send command.
- **title** (Optional[str]) – Notification title.
- **urgency** (Optional[str]) – Notification urgency.
- **up** (int) – How far should we crawl up the stack to get the script's name?
- **args** (str) –

Return type None

xkey(*key*)
Wrapper for *xdotool key*

Parameters **key** (str) –

Return type None

xtype(*keys*, *, *delay=None*)
Wrapper for *xdotool type*

Parameters

- **keys** (*str*) – Keys to type.
- **delay** (*optional*) – Typing delay.

Return type None

lib.meta module

Functions/classes which make use of Python's dynamic nature to inspect a program's internals.

exception BugyiDepreciationWarning
Bases: Warning

DepreciationWarning that doesn't get ignored by default.

class Inspector(*, *up=0*)
Bases: object

Helper class for python introspection (e.g. What line number is this?)

Parameters **up** (int) –

cname(*obj*)
Helper function for getting an object's class name as a string.

Parameters **obj** (object) –

Return type str

deprecated(*func*, *wmsg*)
Used to deprecate @func after renaming it or moving it to a different module/package.

Parameters

- **func** (Callable) –
- **wmsg** (str) –

Return type Callable

scriptname(*, *up=0*)
Returns the name of the current script / module.

Parameters **up** (int) – How far should we crawl up the stack?

Return type str

lib.secrets module

Helper functions/classes related to secrets (e.g. passwords).

get_secret(*key*, **key_parts*, *cmd_list=None*, *folder=None*, *user=None*)

Returns a secret (i.e. a password).

Parameters

- **key** (str) – The key that the secret is associated with.
- **key_parts** (str) – If provided, these are treated as part of the secret key and are joined with the key argument by separating each distinct key part with a period.
- **cmd_list** (Optional[Iterable[str]]) – Optional command list to use for fetching the secret (e.g. ["pass", "show"], which is the default). Note: we will append the key argument to this list before running.
- **folder** (Optional[str]) – If provided, this folder name is prepended to the beginning of the key argument.
- **user** (Optional[str]) – Should we use *sudo -u <user>* to run our secret retriever command as that user?

Return type str

lib.shell module

Helper utilities related to the subprocess module and the shell.

class Process(*popen*, ***, *timeout=15*)

Bases: object

A wrapper around a subprocess.Popen(...) object.

Examples

```
>>> from subprocess import PIPE, Popen
```

```
>>> echo_factory = lambda x: Popen(["echo", x], stdout=PIPE)
```

```
>>> echo_popen = echo_factory("foo")
>>> echo_proc = Process(echo_popen)
>>> echo_proc.out
'foo'
```

```
>>> echo_popen = echo_factory("bar")
>>> out, _err = Process(echo_popen)
>>> out
'bar'
```

Parameters

- **popen** (Popen) –
- **timeout** (float) –

to_error(*, *up=0*)
Converts a Process object into an Err(...) object..

Parameters **up** (int) –

Return type Err[*Process*, *BugyiError*]

exception StillAliveException(*pid*)
Bases: Exception

Raised when Old Instance of Script is Still Running

Parameters **pid** (int) –

command_exists(*cmd*)
Returns True iff the shell command cmd exists.

Parameters **cmd** (str) –

Return type bool

create_pidfile(*, *up=0*)
Writes PID to file, which is created if necessary.

Raises *StillAliveException* – if old instance of script is still alive.

Parameters **up** (int) –

Return type None

safe_popen(*cmd_parts*, *, *up=0*, *timeout=15*, ***kwargs*)
Wrapper for subprocess.Popen(...).

Return type Union[Ok[*Process*, *BugyiError*], Err[*Process*, *BugyiError*]]

Returns

Ok(Process) if the command is successful. OR

Err(BugyiError) otherwise.

Parameters

- **cmd_parts** (Iterable[str]) –
- **up** (int) –
- **timeout** (float) –
- **kwargs** (Any) –

unsafe_popen(*cmd_parts*, *, *timeout=15*, ***kwargs*)
Wrapper for subprocess.Popen(...)

You can use unsafe_popen() instead of safe_popen() when you don't care whether or not the command succeeds.

Return type *Process*

Returns A Process(...) object.

Parameters

- **cmd_parts** (Iterable[str]) –
- **timeout** (float) –
- **kwargs** (Any) –

lib.types module

Helper utilities related to Python types.

assert_never(*value*)

Raises an AssertionError. This function can be used to achieve exhaustiveness checking with mypy.

REFERENCE: https://hakibenita.com/python-mypy-exhaustive-checking

Parameters **value** (NoReturn) –

Return type NoReturn

literal_to_list(*literal*)

Convert a typing.Literal into a list.

Examples

```
>>> from typing import Literal
>>> literal_to_list(Literal['a', 'b', 'c'])
['a', 'b', 'c']
```

```
>>> literal_to_list(Literal['a', 'b', Literal['c', 'd', Literal['e']]])
['a', 'b', 'c', 'd', 'e']
```

```
>>> literal_to_list(Literal['a', 'b', Literal[1, 2, Literal[None]]])
['a', 'b', 1, 2, None]
```

Parameters **literal** (Any) –

Return type List[Union[None, bool, bytes, int, str, Enum]]

lib.xdg module

XDG Utilities

get_base_dir(*xdg_type*)

Return type Path

Returns The base/general XDG user directory.

Parameters **xdg_type** (Literal['cache', 'config', 'data', 'runtime']) –

get_full_dir(*xdg_type*, *, *up=0*)

Return type Path

Returns Full XDG user directory (including scriptname).

Parameters

- **xdg_type** (Literal['cache', 'config', 'data', 'runtime']) –
- **up** (int) –

init_full_dir(*xdg_type*, ***, *up=0*)

Return type Path

Returns Full XDG user directory (including scriptname).

Side Effects: Ensures the full XDG user directory exists before returning it.

Parameters

- **xdg_type** (Literal['cache', 'config', 'data', 'runtime']) –
- **up** (int) –

CHAPTER

TWO

CHANGELOG FOR `BUGYI.LIB`

All notable changes to this project will be documented in this file.

The format is based on Keep a Changelog, and this project adheres to Semantic Versioning.

2.1 Unreleased

No notable changes have been made.

2.2 0.11.0 - 2021-12-20

2.2.1 Removed

- *BREAKING CHANGE*: Sync with cc-python version `v2021.12.20` (drops Python3.7 support).

2.3 0.10.0 - 2021-12-11

2.3.1 Removed

- Remove `bugyi.sh` bash library.

2.4 0.9.0 - 2021-12-11

2.4.1 Added

- Add `bugyi.sh` bash library.

2.5 0.8.0 - 2021-11-24

2.5.1 Changed

- Use python-result library instead of `lib.result` module.

2.6 0.7.0 - 2021-11-23

2.6.1 Changed

- The `shell.*_popen()` functions now return a Process object.

2.7 0.6.1 - 2021-10-04

2.7.1 Fixed

- Fix `_path_to_module()` when `sys.path` contains `Path` objects.

2.8 0.6.0 - 2021-10-01

2.8.1 Added

- Add `io.colors` class.

2.9 0.5.0 - 2021-09-28

2.9.1 Changed

- Improve the `io.get_secret()` function.

2.9.2 Miscellaneous

- Increase test coverage to >=30%.

2.10 0.4.0 - 2021-09-26

2.10.1 Removed

- Removed the `lib.cli` module, which has been migrated to the clap package.

2.11 0.3.0 - 2021-09-25

2.11.1 Changed

- Integrated the `logutils` package into this package.

2.12 0.2.1 - 2021-09-25

2.12.1 Miscellaneous

- Added `py.typed` file so mypy works with this package.

2.13 0.2.0 - 2021-09-25

2.13.1 Added

- Initialized package with initial module files (e.g. `cli.py`, `errors.py`, `meta.py`).

2.14 0.1.0 - 2021-09-23

2.14.1 Miscellaneous

- First release.

CHAPTER

THREE

CONTRIBUTING

3.1 How to submit feedback?

The best way to submit feedback is to file an issue.

If you are reporting a bug, please include:

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

If you are proposing a feature:

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

3.2 Developer's Guide

3.2.1 Badges

Every badge shown below corresponds with a development tool used to maintain this project. Every badge is clickable and links back to that project's github / documentation site:

tools / frameworks used by test suite (i.e. used by ``make test``):

linters used to maintain code quality (i.e. used by ``make lint``):

tools / frameworks used to render documentation (i.e used by ``make build-docs``):

miscellaneous tools used to maintain this project:

3.2.2 Basic Usage

Before making a PR please run the following

- Optional one time setup: run `make use-docker` if you need to build/test this with docker
- `make lint` to check for any format or convention issues
- `make test` to run all tests

3.2.3 How do I . . . ?

3.3 New Releases

This section serves as a reminder to the maintainers of this project on how to release a new version of this package to PyPI.

Make sure all your changes are committed, that you have added a new section to the CHANGELOG.md file, and that you have bumpversion installed. Then run:

```
bumpversion patch  # possible values: major / minor / patch
git push
git push --tags
```

A new version of `bugyi.lib` will then deploy to PyPI if all CI checks pass.

CHAPTER

FOUR

BUGYI.LIB

Overly general Python library used when no other library is a good fit.

project status badges:

version badges:

4.1 Installation

To install `bugyi.lib` using pip, run the following commands in your terminal:

```
python3 -m pip install --user bugyi.lib  # install bugyi.lib
```

If you don't have pip installed, this Python installation guide can guide you through the process.

4.2 Useful Links

- API Reference: A developer's reference of the API exposed by this project.
- cc-python: The cookiecutter that was used to generate this project. Changes made to this cookiecutter are periodically synced with this project using cruft.
- CHANGELOG.md: We use this file to document all notable changes made to this project.
- CONTRIBUTING.md: This document contains guidelines for developers interested in contributing to this project.
- Create a New Issue: Create a new GitHub issue for this project.
- Documentation: This project's full documentation.

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex

PYTHON MODULE INDEX

INDEX

R

S

T

U

W

X

Y