pythonfinder Documentation

Release 1.2.7dev

Dan Ryan <dan@danryan.co>

April 01, 2021

CONTENTS:

CHAPTER

ONE

PYTHONFINDER: CROSS PLATFORM SEARCH TOOL FOR FINDING PYTHONS

1.1 Installation

Install from PyPI:

```
$ pipenv install pythonfinder
```

Install from Github:

```
$ pipenv install -e git+https://github.com/sarugaku/pythonfinder.git#egg=pythonfinder
```

1.2 Usage

Using PythonFinder is easy. Simply import it and ask for a python:

```
>>> from pythonfinder.pythonfinder import PythonFinder
>>> PythonFinder.from_line('python3')
'/home/techalchemy/.pyenv/versions/3.6.5/python3'

>>> from pythonfinder import Finder
>>> f = Finder()
>>> f.find_python_version(3, minor=6)
PathEntry(path=PosixPath('/home/hawk/.pyenv/versions/3.6.5/bin/python'), _children={},
 ↪ is_root=False, only_python=False, py_version=PythonVersion(major=3, minor=6,␣
 ↪patch=5, is_prerelease=False, is_postrelease=False, is_devrelease=False, version=
 ↪<Version('3.6.5')>, architecture='64bit', comes_from=PathEntry(path=PosixPath('/
 ↪home/hawk/.pyenv/versions/3.6.5/bin/python'), _children={}, is_root=True, only_
 ↪python=False, py_version=None, pythons=None), executable=None), pythons=None)
>>> f.find_python_version(2)
```

(continues on next page)

(continued from previous page)

```
PathEntry(path=PosixPath('/home/hawk/.pyenv/shims/python2'), ...py_
↪version=PythonVersion(major=2, minor=7, patch=15, is_prerelease=False, is_
↪postrelease=False, is_devrelease=False, version=<Version('2.7.15')>, architecture=
↪'64bit', comes_from=PathEntry(path=PosixPath('/home/hawk/.pyenv/shims/python2'), _
↪children={}, is_root=True, only_python=False, py_version=None, pythons=None),␣
↪executable=None), pythons=None)
>>> f.find_python_version("anaconda3-5.3.0")
```

Find a named distribution, such as `anaconda3-5.3.0`:

```
PathEntry(path=PosixPath('/home/hawk/.pyenv/versions/anaconda3-5.3.0/bin/python3.7m'),
↪ _children={'/home/hawk/.pyenv/versions/anaconda3-5.3.0/bin/python3.7m': ...}, only_
↪python=False, name='anaconda3-5.3.0', _py_version=PythonVersion(major=3, minor=7,␣
↪patch=0, is_prerelease=False, is_postrelease=False, is_devrelease=False,...))
```

PythonFinder can even find beta releases:

```
>>> f.find_python_version(3, minor=7)
PathEntry(path=PosixPath('/home/hawk/.pyenv/versions/3.7.0b1/bin/python'), _children=
↪{}, is_root=False, only_python=False, py_version=PythonVersion(major=3, minor=7,␣
↪patch=0, is_prerelease=True, is_postrelease=False, is_devrelease=False, version=
↪<Version('3.7.0b1')>, architecture='64bit', comes_from=PathEntry(path=PosixPath('/
↪home/hawk/.pyenv/versions/3.7.0b1/bin/python'), _children={}, is_root=True, only_
↪python=False, py_version=None, pythons=None), executable=None), pythons=None)

>>> f.which('python')
PathEntry(path=PosixPath('/home/hawk/.pyenv/versions/3.6.5/bin/python'), _children={},
↪ is_root=False, only_python=False, py_version=PythonVersion(major=3, minor=6,␣
↪patch=5, is_prerelease=False, is_postrelease=False, is_devrelease=False, version=
↪<Version('3.6.5')>, architecture='64bit', comes_from=PathEntry(path=PosixPath('/
↪home/hawk/.pyenv/versions/3.6.5/bin/python'), _children={}, is_root=True, only_
↪python=False, py_version=None, pythons=None), executable=None), pythons=None)
```

1.3 Windows Support

PythonFinder natively supports windows via both the *PATH* environment variable and PEP-514 compliant finder which comes by default with python 3. Usage on windows becomes:

```
>>> from pythonfinder import Finder
>>> f = Finder()
>>> f.find_python_version(3, minor=6)
PythonVersion(major=3, minor=6, patch=4, is_prerelease=False, is_postrelease=False,␣
↪is_devrelease=False, version=<Version('3.6.4')>, architecture='64bit', comes_
↪from=PathEntry(path=WindowsPath('C:/Program Files/Python36/python.exe'), _children=
↪{}, is_root=False, only_python=True, py_version=None, pythons=None),␣
↪executable=WindowsPath('C:/Program Files/Python36/python.exe'))

>>> f.find_python_version(3, minor=7, pre=True)
PythonVersion(major=3, minor=7, patch=0, is_prerelease=True, is_postrelease=False, is_
↪devrelease=False, version=<Version('3.7.0b5')>, architecture='64bit', comes_
↪from=PathEntry(path=WindowsPath('C:/Program Files/Python37/python.exe'), _children=
↪{}, is_root=False, only_python=True, py_version=None, pythons=None),␣
↪executable=WindowsPath('C:/Program Files/Python37/python.exe'))
```

(continues on next page)

(continued from previous page)

```
>>> f.which('python')
PathEntry(path=WindowsPath('C:/Python27/python.exe'), _children={}, is_root=False,␣
↪only_python=False, py_version=None, pythons=None)
```

1.3.1 Finding Executables

PythonFinder also provides **which** functionality across platforms, and it uses lazy loading and fast-returns to be performant at this task.

```
>>> f.which('cmd')
PathEntry(path=WindowsPath('C:/windows/system32/cmd.exe'), _children={}, is_
↪root=False, only_python=False, py_version=None, pythons=None)

>>> f.which('code')
PathEntry(path=WindowsPath('C:/Program Files/Microsoft VS Code/bin/code'), _children=
↪{}, is_root=False, only_python=False, py_version=None, pythons=None)

 >>> f.which('vim')
PathEntry(path=PosixPath('/usr/bin/vim'), _children={}, is_root=False, only_
↪python=False, py_version=None, pythons=None)

>>> f.which('inv')
PathEntry(path=PosixPath('/home/hawk/.pyenv/versions/3.6.5/bin/inv'), _children={},␣
↪is_root=False, only_python=False, py_version=None, pythons=None)
```

1.3.2 Architecture support

PythonFinder supports architecture specific lookups on all platforms:

```
>>> f.find_python_version(3, minor=6, arch="64")
PathEntry(path=PosixPath('/usr/bin/python3'), _children={'/usr/bin/python3': ...},␣
↪only_python=False, name='python3', _py_version=PythonVersion(major=3, minor=6,␣
↪patch=7, is_prerelease=False, is_postrelease=False, is_devrelease=False, is_
↪debug=False, version=<Version('3.6.7')>, architecture='64bit', comes_from=...,␣
↪executable='/usr/bin/python3', name='python3'), _pythons=defaultdict(None, {}), is_
↪root=False)
```

1.4 Integrations

- Pyenv
- ASDF
- PEP-514
- Virtualenv
- Pipenv

CHAPTER

TWO

PYTHONFINDER PACKAGE

class pythonfinder.**Finder**(*path=None*, *system=False*, *global_search=True*, *ignore_unsupported=True*, *sort_by_path=False*)

Bases: object

A cross-platform Finder for locating python and other executables.

Searches for python and other specified binaries starting in *path*, if supplied, but searching the bin path of `sys.executable` if *system* is `True`, and then searching in the `os.environ['PATH']` if *global_search* is `True`. When *global_search* is `False`, this search operation is restricted to the allowed locations of *path* and *system*.

create_system_path()

find_all_python_versions(*major=None*, *minor=None*, *patch=None*, *pre=None*, *dev=None*, *arch=None*, *name=None*)

find_python_version(*major=None*, *minor=None*, *patch=None*, *pre=None*, *dev=None*, *arch=None*, *name=None*, *sort_by_path=False*)

Find the python version which corresponds most closely to the version requested.

Parameters

- **int] major** (*Union[str,]*) – The major version to look for, or the full version, or the name of the target version.
- **minor** (*Optional[int]*) – The minor version. If provided, disables string-based lookups from the major version field.
- **patch** (*Optional[int]*) – The patch version.
- **pre** (*Optional[bool]*) – If provided, specifies whether to search pre-releases.
- **dev** (*Optional[bool]*) – If provided, whether to search dev-releases.
- **arch** (*Optional[str]*) – If provided, which architecture to search.
- **name** (*Optional[str]*) – *Name* of the target python, e.g. `anaconda3-5.3.0`
- **sort_by_path** (*bool*) – Whether to sort by path – default sort is by version(default: False)

Returns A new *PathEntry* pointer at a matching python version, if one can be located.

Return type *pythonfinder.models.path.PathEntry*

classmethod parse_major(*major*, *minor=None*, *patch=None*, *pre=None*, *dev=None*, *arch=None*)

rehash()

reload_system_path()
Rebuilds the base system path and all of the contained finders within it.

This will re-apply any changes to the environment or any version changes on the system.

property system_path

which(*exe*)

property windows_finder

exception pythonfinder.**InvalidPythonVersion**
Bases: Exception

Raised when parsing an invalid python version

class pythonfinder.**SystemPath**(*global_search=True*, *paths=NOTHING*, *executables=NOTHING*, *python_executables=NOTHING*, *path_order=NOTHING*, *python_version_dict=NOTHING*, *only_python=False*, *pyenv_finder=None*, *asdf_finder=None*, *windows_finder=None*, *system=False*, *version_dict=NOTHING*, *ignore_unsupported=False*, *SystemPath__finders=NOTHING*)
Bases: object

static check_for_asdf()

static check_for_pyenv()

clear_caches()

classmethod create(*path=None*, *system=False*, *only_python=False*, *global_search=True*, *ignore_unsupported=True*)
Create a new pythonfinder.models.SystemPath instance.

Parameters

- **path** – Search path to prepend when searching, defaults to None
- **path** – str, optional
- **system** (*bool*) – Whether to use the running python by default instead of searching, defaults to False
- **only_python** (*bool*) – Whether to search only for python executables, defaults to False
- **ignore_unsupported** (*bool*) – Whether to ignore unsupported python versions, if False, an error is raised, defaults to True

Returns A new pythonfinder.models.SystemPath instance.

Return type pythonfinder.models.SystemPath

create_python_version_dict()

executables

find_all(*executable*)
Search the path for an executable. Return all copies.

Parameters **executable** (*str*) – Name of the executable

Returns List[PathEntry]

find_all_python_versions(*major=None*, *minor=None*, *patch=None*, *pre=None*, *dev=None*, *arch=None*, *name=None*)
Search for a specific python version on the path. Return all copies

Parameters

- **major** (*int*) – Major python version to search for.
- **minor** (*int*) – Minor python version to search for, defaults to None
- **patch** (*int*) – Patch python version to search for, defaults to None
- **pre** (*bool*) – Search for prereleases (default None) - prioritize releases if None
- **dev** (*bool*) – Search for devreleases (default None) - prioritize releases if None
- **arch** (*str*) – Architecture to include, e.g. '64bit', defaults to None
- **name** (*str*) – The name of a python version, e.g. `anaconda3-5.3.0`

Returns A list of `PathEntry` instances matching the version requested.

Return type List[`PathEntry`]

find_python_version(*major=None*, *minor=None*, *patch=None*, *pre=None*, *dev=None*, *arch=None*, *name=None*, *sort_by_path=False*)

Search for a specific python version on the path.

Parameters

- **major** (*int*) – Major python version to search for.
- **minor** (*int*) – Minor python version to search for, defaults to None
- **patch** (*int*) – Patch python version to search for, defaults to None
- **pre** (*bool*) – Search for prereleases (default None) - prioritize releases if None
- **dev** (*bool*) – Search for devreleases (default None) - prioritize releases if None
- **arch** (*str*) – Architecture to include, e.g. '64bit', defaults to None
- **name** (*str*) – The name of a python version, e.g. `anaconda3-5.3.0`
- **sort_by_path** (*bool*) – Whether to sort by path – default sort is by version(default: False)

Returns A `PathEntry` instance matching the version requested.

Return type `PathEntry`

property finders

get_path(*path*)

get_pythons(*finder*)

path_entries

python_executables

reload_finder(*finder_name*)

version_dict

which(*executable*)

Search for an executable on the path.

Parameters **executable** (*str*) – Name of the executable to be located.

Returns `PathEntry` object.

class pythonfinder.**WindowsFinder**(*paths=NOTHING*, *version_list=NOTHING*, *versions=NOTHING*, *pythons=NOTHING*)

Bases: *pythonfinder.models.mixins.BaseFinder*

classmethod create(**args*, ***kwargs*)

find_all_python_versions(*major=None*, *minor=None*, *patch=None*, *pre=None*, *dev=None*, *arch=None*, *name=None*)

find_python_version(*major=None*, *minor=None*, *patch=None*, *pre=None*, *dev=None*, *arch=None*, *name=None*)

get_pythons()

get_versions()

Return the available versions from the finder

property pythons

property versions

2.1 Subpackages

2.1.1 pythonfinder.models package

2.1.1.1 Submodules

pythonfinder.models.mixins module

class pythonfinder.models.mixins.**BaseFinder**

Bases: object

classmethod create(**args*, ***kwargs*)

property expanded_paths

get_versions()

Return the available versions from the finder

property pythons

property version_paths

class pythonfinder.models.mixins.**BasePath**(*path=None*, *children=NOTHING*, *only_python=False*, *name=NOTHING*, *py_version=None*, *pythons=NOTHING*, *is_dir=None*, *is_executable=None*, *is_python=None*)

Bases: object

property as_python

property children

find_all_python_versions(*major=None*, *minor=None*, *patch=None*, *pre=None*, *dev=None*, *arch=None*, *name=None*)

Search for a specific python version on the path. Return all copies

Parameters

- **major** (*int*) – Major python version to search for.

- **minor** (*int*) – Minor python version to search for, defaults to None
- **patch** (*int*) – Patch python version to search for, defaults to None
- **pre** (*bool*) – Search for prereleases (default None) - prioritize releases if None
- **dev** (*bool*) – Search for devreleases (default None) - prioritize releases if None
- **arch** (*str*) – Architecture to include, e.g. '64bit', defaults to None
- **name** (*str*) – The name of a python version, e.g. `anaconda3-5.3.0`

Returns A list of `PathEntry` instances matching the version requested.

Return type List[`PathEntry`]

find_python_version(*major=None*, *minor=None*, *patch=None*, *pre=None*, *dev=None*, *arch=None*, *name=None*)

Search or self for the specified Python version and return the first match.

Parameters

- **major** (*int*) – Major version number.
- **minor** (*int*) – Minor python version to search for, defaults to None
- **patch** (*int*) – Patch python version to search for, defaults to None
- **pre** (*bool*) – Search for prereleases (default None) - prioritize releases if None
- **dev** (*bool*) – Search for devreleases (default None) - prioritize releases if None
- **arch** (*str*) – Architecture to include, e.g. '64bit', defaults to None
- **name** (*str*) – The name of a python version, e.g. `anaconda3-5.3.0`

Returns A `PathEntry` instance matching the version requested.

get_name()

get_py_version()

property is_dir

property is_executable

property is_python

name

next()

only_python

path

property py_version

property pythons

which(*name*)

Search in this path for an executable.

Parameters **executable** (*str*) – The name of an executable to search for.

Returns `PathEntry` instance.

pythonfinder.models.path module

class pythonfinder.models.path.**PathEntry**(*path=None*, *children=NOTHING*, *only_python=False*, *name=NOTHING*, *py_version=None*, *pythons=NOTHING*, *is_dir=None*, *is_executable=None*, *is_python=None*, *is_root=True*)

Bases: *pythonfinder.models.mixins.BasePath*

property children

classmethod create(*path*, *is_root=False*, *only_python=False*, *pythons=None*, *name=None*)

Helper method for creating new `pythonfinder.models.PathEntry` instances.

Parameters

- **path** (*str*) – Path to the specified location.
- **is_root** (*bool*) – Whether this is a root from the environment PATH variable, defaults to False
- **only_python** (*bool*) – Whether to search only for python executables, defaults to False
- **pythons** (*dict*) – A dictionary of existing python objects (usually from a finder), defaults to None
- **name** (*str*) – Name of the python version, e.g. `anaconda3-5.3.0`

Returns A new instance of the class.

Return type `pythonfinder.models.PathEntry`

is_root

class pythonfinder.models.path.**SystemPath**(*global_search=True*, *paths=NOTHING*, *executables=NOTHING*, *python_executables=NOTHING*, *path_order=NOTHING*, *python_version_dict=NOTHING*, *only_python=False*, *pyenv_finder=None*, *asdf_finder=None*, *windows_finder=None*, *system=False*, *version_dict=NOTHING*, *ignore_unsupported=False*, *SystemPath__finders=NOTHING*)

Bases: `object`

static check_for_asdf()

static check_for_pyenv()

clear_caches()

classmethod create(*path=None*, *system=False*, *only_python=False*, *global_search=True*, *ignore_unsupported=True*)

Create a new `pythonfinder.models.SystemPath` instance.

Parameters

- **path** – Search path to prepend when searching, defaults to None
- **path** – str, optional

- **system** (*bool*) – Whether to use the running python by default instead of searching, defaults to False
- **only_python** (*bool*) – Whether to search only for python executables, defaults to False
- **ignore_unsupported** (*bool*) – Whether to ignore unsupported python versions, if False, an error is raised, defaults to True

Returns A new `pythonfinder.models.SystemPath` instance.

Return type `pythonfinder.models.SystemPath`

create_python_version_dict()

executables

find_all(*executable*)

Search the path for an executable. Return all copies.

Parameters **executable** (*str*) – Name of the executable

Returns List[PathEntry]

find_all_python_versions(*major=None*, *minor=None*, *patch=None*, *pre=None*, *dev=None*, *arch=None*, *name=None*)

Search for a specific python version on the path. Return all copies

Parameters

- **major** (*int*) – Major python version to search for.
- **minor** (*int*) – Minor python version to search for, defaults to None
- **patch** (*int*) – Patch python version to search for, defaults to None
- **pre** (*bool*) – Search for prereleases (default None) - prioritize releases if None
- **dev** (*bool*) – Search for devreleases (default None) - prioritize releases if None
- **arch** (*str*) – Architecture to include, e.g. '64bit', defaults to None
- **name** (*str*) – The name of a python version, e.g. `anaconda3-5.3.0`

Returns A list of `PathEntry` instances matching the version requested.

Return type List[`PathEntry`]

find_python_version(*major=None*, *minor=None*, *patch=None*, *pre=None*, *dev=None*, *arch=None*, *name=None*, *sort_by_path=False*)

Search for a specific python version on the path.

Parameters

- **major** (*int*) – Major python version to search for.
- **minor** (*int*) – Minor python version to search for, defaults to None
- **patch** (*int*) – Patch python version to search for, defaults to None
- **pre** (*bool*) – Search for prereleases (default None) - prioritize releases if None
- **dev** (*bool*) – Search for devreleases (default None) - prioritize releases if None
- **arch** (*str*) – Architecture to include, e.g. '64bit', defaults to None
- **name** (*str*) – The name of a python version, e.g. `anaconda3-5.3.0`

- **sort_by_path** (*bool*) – Whether to sort by path – default sort is by version(default: False)

Returns A `PathEntry` instance matching the version requested.

Return type `PathEntry`

property finders

get_path(*path*)

get_pythons(*finder*)

path_entries

python_executables

reload_finder(*finder_name*)

version_dict

which(*executable*)

Search for an executable on the path.

Parameters **executable** (*str*) – Name of the executable to be located.

Returns `PathEntry` object.

class pythonfinder.models.path.**VersionPath**(*global_search=True*, *paths=NOTHING*, *executables=NOTHING*, *python_executables=NOTHING*, *path_order=NOTHING*, *python_version_dict=NOTHING*, *only_python=False*, *pyenv_finder=None*, *asdf_finder=None*, *windows_finder=None*, *system=False*, *version_dict=NOTHING*, *ignore_unsupported=False*, *SystemPath__finders=NOTHING*, *base=None*, *name=None*)

Bases: *pythonfinder.models.path.SystemPath*

classmethod create(*path*, *only_python=True*, *pythons=None*, *name=None*)

Accepts a path to a base python version directory.

Generates the version listings for it

pythonfinder.models.python module

class pythonfinder.models.python.**PythonFinder**(*path=None*, *children=NOTHING*, *only_python=False*, *name=NOTHING*, *py_version=None*, *is_dir=None*, *is_executable=None*, *is_python=None*, *root=None*, *ignore_unsupported=True*, *version_glob_path='versions/*'*, *sort_function=None*, *roots=NOTHING*, *paths=NOTHING*, *shim_dir='shims'*, *versions=NOTHING*, *pythons=NOTHING*)

Bases: *pythonfinder.models.mixins.BaseFinder*, *pythonfinder.models.mixins.BasePath*

classmethod create(*root*, *sort_function*, *version_glob_path=None*, *ignore_unsupported=True*)

property expanded_paths

find_all_python_versions(*major=None*, *minor=None*, *patch=None*, *pre=None*, *dev=None*, *arch=None*, *name=None*)

Search for a specific python version on the path. Return all copies

Parameters

- **major** (*int*) – Major python version to search for.
- **minor** (*int*) – Minor python version to search for, defaults to None
- **patch** (*int*) – Patch python version to search for, defaults to None
- **pre** (*bool*) – Search for prereleases (default None) - prioritize releases if None
- **dev** (*bool*) – Search for devreleases (default None) - prioritize releases if None
- **arch** (*str*) – Architecture to include, e.g. '64bit', defaults to None
- **name** (*str*) – The name of a python version, e.g. `anaconda3-5.3.0`

Returns A list of `PathEntry` instances matching the version requested.

Return type List[`PathEntry`]

find_python_version(*major=None*, *minor=None*, *patch=None*, *pre=None*, *dev=None*, *arch=None*, *name=None*)

Search or self for the specified Python version and return the first match.

Parameters

- **major** (*int*) – Major version number.
- **minor** (*int*) – Minor python version to search for, defaults to None
- **patch** (*int*) – Patch python version to search for, defaults to None
- **pre** (*bool*) – Search for prereleases (default None) - prioritize releases if None
- **dev** (*bool*) – Search for devreleases (default None) - prioritize releases if None
- **arch** (*str*) – Architecture to include, e.g. '64bit', defaults to None
- **name** (*str*) – The name of a python version, e.g. `anaconda3-5.3.0`

Returns A `PathEntry` instance matching the version requested.

get_bin_dir(*base*)

get_paths()

get_pythons()

get_version_order()

ignore_unsupported

Whether to ignore any paths which raise exceptions and are not actually python

property is_asdf

property is_pyenv

paths

List of paths discovered during search

property pythons

root

roots
The root locations used for discovery

shim_dir
shim directory

sort_function
The function to use to sort version order when returning an ordered verion set

classmethod version_from_bin_dir(*entry*)

version_glob_path
Glob path for python versions off of the root directory

property versions

which(*name*)
Search in this path for an executable.

Parameters **executable** (*str*) – The name of an executable to search for.

Returns PathEntry instance.

class pythonfinder.models.python.**PythonVersion**(*major=0*, *minor=None*, *patch=None*, *is_prerelease=False*, *is_postrelease=False*, *is_devrelease=False*, *is_debug=False*, *version=None*, *architecture=None*, *comes_from=None*, *executable=None*, *company=None*, *name=None*)

Bases: object

architecture

as_dict()

as_major()

as_minor()

comes_from

company

classmethod create(***kwargs*)

executable

classmethod from_path(*path*, *name=None*, *ignore_unsupported=True*, *company=None*)
Parses a python version from a system path.

Raises: ValueError – Not a valid python path

Parameters

- **path** (str or *PathEntry* instance) – A string or *PathEntry*
- **name** (*str*) – Name of the python distribution in question
- **ignore_unsupported** (*bool*) – Whether to ignore or error on unsupported paths.
- **company** (*Optional[str]*) – The company or vendor packaging the distribution.

Returns An instance of a PythonVersion.

Return type *PythonVersion*

classmethod from_windows_launcher(*launcher_entry*, *name=None*, *company=None*)
Create a new PythonVersion instance from a Windows Launcher Entry

Parameters

- **launcher_entry** – A python launcher environment object.
- **name** (*Optional[str]*) – The name of the distribution.
- **company** (*Optional[str]*) – The name of the distributing company.

Returns An instance of a PythonVersion.

Return type *PythonVersion*

get_architecture()

is_debug

is_devrelease

is_postrelease

is_prerelease

major

matches(*major=None*, *minor=None*, *patch=None*, *pre=False*, *dev=False*, *arch=None*, *debug=False*, *python_name=None*)

minor

name

classmethod parse(*version*)
Parse a valid version string into a dictionary

Raises: ValueError – Unable to parse version string ValueError – Not a valid python version TypeError – NoneType or unparseable type passed in

Parameters **version** (*str*) – A valid version string

Returns A dictionary with metadata about the specified python version.

Return type dict

classmethod parse_executable(*path*)

patch

update_metadata(*metadata*)
Update the metadata on the current *pythonfinder.models.python.PythonVersion*

Given a parsed version dictionary from *pythonfinder.utils.parse_python_version()*, update the instance variables of the current version instance to reflect the newly supplied values.

version

property version_sort
A tuple for sorting against other instances of the same class.

Returns a tuple of the python version but includes points for core python, non-dev, and non-prerelease versions. So released versions will have 2 points for this value. E.g. `(1, 3, 6, 6, 2)` is a release, `(1, 3, 6, 6, 1)` is a prerelease, `(1, 3, 6, 6, 0)` is a dev release, and `(1, 3, 6, 6, 3)` is a postrelease. `(0, 3, 7, 3, 2)` represents a non-core python release, e.g. by a repackager of python like Continuum.

property version_tuple
Provides a version tuple for using as a dictionary key.

Returns A tuple describing the python version meetadata contained.

Return type tuple

class pythonfinder.models.python.**VersionMap**(*versions=NOTHING*)
Bases: object

add_entry(*entry*)

merge(*target*)

pythonfinder.models.python.**overload**(*f*)

pythonfinder.models.windows module

class pythonfinder.models.windows.**WindowsFinder**(*paths=NOTHING*, *version_list=NOTHING*, *versions=NOTHING*, *pythons=NOTHING*)
Bases: *pythonfinder.models.mixins.BaseFinder*

classmethod create(**args*, ***kwargs*)

find_all_python_versions(*major=None*, *minor=None*, *patch=None*, *pre=None*, *dev=None*, *arch=None*, *name=None*)

find_python_version(*major=None*, *minor=None*, *patch=None*, *pre=None*, *dev=None*, *arch=None*, *name=None*)

get_pythons()

get_versions()
Return the available versions from the finder

property pythons

property versions

2.2 Submodules

2.2.1 pythonfinder.cli module

2.2.2 pythonfinder.environment module

pythonfinder.environment.**SUBPROCESS_TIMEOUT = 5**
The default subprocess timeout for determining python versions

Set to **5** by default.

pythonfinder.environment.**get_shim_paths**()

pythonfinder.environment.**is_type_checking**()

2.2.3 pythonfinder.exceptions module

exception pythonfinder.exceptions.**InvalidPythonVersion**
Bases: Exception

Raised when parsing an invalid python version

2.2.4 pythonfinder.pythonfinder module

class pythonfinder.pythonfinder.**Finder**(*path=None*, *system=False*, *global_search=True*, *ignore_unsupported=True*, *sort_by_path=False*)
Bases: object

A cross-platform Finder for locating python and other executables.

Searches for python and other specified binaries starting in *path*, if supplied, but searching the bin path of `sys.executable` if *system* is `True`, and then searching in the `os.environ['PATH']` if *global_search* is `True`. When *global_search* is `False`, this search operation is restricted to the allowed locations of *path* and *system*.

create_system_path()

find_all_python_versions(*major=None*, *minor=None*, *patch=None*, *pre=None*, *dev=None*, *arch=None*, *name=None*)

find_python_version(*major=None*, *minor=None*, *patch=None*, *pre=None*, *dev=None*, *arch=None*, *name=None*, *sort_by_path=False*)
Find the python version which corresponds most closely to the version requested.

Parameters

- **int] major** (*Union[str,]*) – The major version to look for, or the full version, or the name of the target version.
- **minor** (*Optional[int]*) – The minor version. If provided, disables string-based lookups from the major version field.
- **patch** (*Optional[int]*) – The patch version.
- **pre** (*Optional[bool]*) – If provided, specifies whether to search pre-releases.
- **dev** (*Optional[bool]*) – If provided, whether to search dev-releases.
- **arch** (*Optional[str]*) – If provided, which architecture to search.
- **name** (*Optional[str]*) – *Name* of the target python, e.g. `anaconda3-5.3.0`
- **sort_by_path** (*bool*) – Whether to sort by path – default sort is by version(default: False)

Returns A new *PathEntry* pointer at a matching python version, if one can be located.

Return type *pythonfinder.models.path.PathEntry*

classmethod parse_major(*major*, *minor=None*, *patch=None*, *pre=None*, *dev=None*, *arch=None*)

rehash()

reload_system_path()
Rebuilds the base system path and all of the contained finders within it.

This will re-apply any changes to the environment or any version changes on the system.

property system_path

which(*exe*)

property windows_finder

2.2.5 pythonfinder.utils module

pythonfinder.utils.**dedup**(*iterable*)
Deduplicate an iterable object like iter(set(iterable)) but order-reserved.

pythonfinder.utils.**ensure_path**(*path*)
Given a path (either a string or a Path object), expand variables and return a Path object.

Parameters **path** (str or Path) – A string or a Path object.

Returns A fully expanded Path object.

Return type Path

pythonfinder.utils.**expand_paths**(*path*, *only_python=True*)
Recursively expand a list or *PathEntry* instance

Parameters

- **PathEntry] path** (*Union[Sequence,*) – The path or list of paths to expand
- **only_python** (*bool*) – Whether to filter to include only python paths, default True

Returns An iterator over the expanded set of path entries

Return type Iterator[*PathEntry*]

pythonfinder.utils.**filter_pythons**(*path*)
Return all valid pythons in a given path

pythonfinder.utils.**get_python_version**(*path*)
Get python version string using subprocess from a given path.

pythonfinder.utils.**guess_company**(*path*)
Given a path to python, guess the company who created it

Parameters **path** (*str*) – The path to guess about

Returns The guessed company

Return type Optional[str]

pythonfinder.utils.**is_in_path**(*path*, *parent*)

pythonfinder.utils.**looks_like_python**(*name*)
Determine whether the supplied filename looks like a possible name of python.

Parameters **name** (*str*) – The name of the provided file.

Returns Whether the provided name looks like python.

Return type bool

pythonfinder.utils.**normalize_path**(*path*)

pythonfinder.utils.**optional_instance_of**(*cls*)
Return an validator to determine whether an input is an optional instance of a class.

Returns A validator to determine optional instance membership.

Return type _OptionalValidator

pythonfinder.utils.**parse_asdf_version_order**(*filename='.tool-versions'*)

pythonfinder.utils.**parse_pyenv_version_order**(*filename='version'*)

pythonfinder.utils.**parse_python_version**(*version_str*)

pythonfinder.utils.**path_is_executable**(*path*)
Determine whether the supplied path is executable.

Returns Whether the provided path is executable.

Return type bool

pythonfinder.utils.**path_is_known_executable**(*path*)
Returns whether a given path is a known executable from known executable extensions or has the executable bit toggled.

Parameters **path** (Path) – The path to the target executable.

Returns True if the path has chmod +x, or is a readable, known executable extension.

Return type bool

pythonfinder.utils.**path_is_python**(*path*)
Determine whether the supplied path is executable and looks like a possible path to python.

Parameters **path** (Path) – The path to an executable.

Returns Whether the provided path is an executable path to python.

Return type bool

pythonfinder.utils.**path_is_pythoncore**(*path*)
Given a path, determine whether it appears to be pythoncore.

Does not verify whether the path is in fact a path to python, but simply does an exclusionary check on the possible known python implementations to see if their names are present in the path (fairly dumb check).

Parameters **path** (*str*) – The path to check

Returns Whether that path is a PythonCore path or not

Return type bool

pythonfinder.utils.**split_version_and_name**(*major=None*, *minor=None*, *patch=None*, *name=None*)

pythonfinder.utils.**unnest**(*item*)

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

p

INDEX

F

G

R

S

U

V

W