pythonfinder Documentation

Release 2.0.5

Dan Ryan <dan@danryan.co>

July 23, 2023

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)
PathEntry(path=PosixPath('/home/hawk/.pyenv/shims/python2'), ...py_
```

(continues on next page)

(continued from previous page)

```
↪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={},␣
```

(continues on next page)

(continued from previous page)

```
↪is_root=False, only_python=True, py_version=None, pythons=None),␣
↪executable=WindowsPath('C:/Program Files/Python37/python.exe'))

>>> 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**(***data*)

Bases: FinderBaseModel

create_system_path()

Return type
SystemPath

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

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*)

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

Parameters

- **major** (*str | int | None*) – The major version to look for, or the full version, or the name of the target version.
- **minor** (*int | None*) – The minor version. If provided, disables string-based lookups from the major version field.
- **patch** (*int | None*) – The patch version.
- **pre** (*bool | None*) – If provided, specifies whether to search pre-releases.
- **dev** (*bool | None*) – If provided, whether to search dev-releases.
- **arch** (*str | None*) – If provided, which architecture to search.
- **name** (*str | None*) – *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)

Return type
PathEntry | None

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

global_search: bool

ignore_unsupported: bool

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

Return type
dict[str, Any]

path_prepend: Optional[str]

sort_by_path: bool

system: bool

system_path: Optional[SystemPath]

which(*exe*)

Return type
PathEntry | None

exception pythonfinder.InvalidPythonVersion

Bases: Exception

Raised when parsing an invalid python version

class pythonfinder.SystemPath(*, *global_search: bool = True*, *paths: Dict[str, Union[*PythonFinder, PathEntry*]] = None*, *executables_tracking: List[*PathEntry*] = None*, *python_executables_tracking: Dict[str,* PathEntry*] = None*, *path_order: List[str] = None*, *python_version_dict: Dict[Tuple, Any] = None*, *version_dict_tracking: Dict[Tuple, List[*PathEntry*]] = None*, *only_python: bool = False*, *pyenv_finder: Optional[*PythonFinder*] = None*, *asdf_finder: Optional[*PythonFinder*] = None*, *system: bool = False*, *ignore_unsupported: bool = False*, *finders_dict: Dict[str,* PythonFinder*] = None*)

Bases: FinderBaseModel

class Config

Bases: object

allow_mutation = True

arbitrary_types_allowed = True

include_private_attributes = True

keep_untouched = (<class 'cached_property.cached_property'>,)

validate_assignment = True

asdf_finder: Optional[PythonFinder]

static check_for_asdf()

static check_for_pyenv()

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

Create a new pythonfinder.models.SystemPath instance.

Parameters

- **path** (*str* | *None*) – 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

Return type
SystemPath

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

property executables: list[*PathEntry*]

executables_tracking: List[PathEntry]

find_all(*executable*)

Search the path for an executable. Return all copies.

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

Return type
list[*PathEntry* | *PythonFinder*]

Returns
List[PathEntry]

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

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*)

Return type
PathEntry

property finders: list[str]

finders_dict: Dict[str, PythonFinder]

get_path(*path*)

Return type
PythonFinder | *PathEntry*

get_pythons(*finder*)

Return type
Iterator

global_search: bool

ignore_unsupported: bool

only_python: bool

path_entries

path_order: List[str]

paths: Dict[str, Union[PythonFinder, PathEntry]]

pyenv_finder: Optional[PythonFinder]

python_executables

python_executables_tracking: Dict[str, PathEntry]

python_version_dict: Dict[Tuple, Any]

classmethod set_defaults(*values*)

system: bool

version_dict

version_dict_tracking: Dict[Tuple, List[PathEntry]]

which(*executable*)

Search for an executable on the path.

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

Return type
PathEntry | None

Returns
PathEntry object.

2.1 Subpackages

2.1.1 pythonfinder.models package

2.1.1.1 Submodules

pythonfinder.models.mixins module

class pythonfinder.models.mixins.**PathEntry**(***data*)

Bases: BaseModel

class Config

Bases: object

allow_mutation = True

arbitrary_types_allowed = True

include_private_attributes = True

validate_assignment = True

property as_python: *PythonVersion*

property children: dict[str, *PathEntry*]

children_ref: Optional[Any]

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

Return type
PathEntry

Returns
A new instance of the class.

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

Return type
list[*PathEntry*]

Returns
A list of PathEntry instances matching the version requested.

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`

Return type
PathEntry | None

Returns
A `PathEntry` instance matching the version requested.

get_py_version()

property is_dir: bool

is_dir_ref: Optional[bool]

property is_executable: bool

is_executable_ref: Optional[bool]

property is_python: bool

is_python_ref: Optional[bool]

is_root: bool

name: Optional[str]

next()

Return type
Generator

only_python: Optional[bool]

path: Optional[Path]

property py_version: *PythonVersion* | None

py_version_ref: Optional[Any]

property pythons: dict[str | Path, *PathEntry*]

pythons_ref: Optional[Dict[Any, Any]]

classmethod set_children(*v*, *values*, ***kwargs*)

which(*name*)

Search in this path for an executable.

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

Return type
PathEntry | None

Returns
`PathEntry` instance.

pythonfinder.models.path module

class pythonfinder.models.path.**SystemPath**(*, *global_search: bool = True*, *paths: Dict[str, Union[PythonFinder, PathEntry]] = None*, *executables_tracking: List[PathEntry] = None*, *python_executables_tracking: Dict[str, PathEntry] = None*, *path_order: List[str] = None*, *python_version_dict: Dict[Tuple, Any] = None*, *version_dict_tracking: Dict[Tuple, List[PathEntry]] = None*, *only_python: bool = False*, *pyenv_finder: Optional[PythonFinder] = None*, *asdf_finder: Optional[PythonFinder] = None*, *system: bool = False*, *ignore_unsupported: bool = False*, *finders_dict: Dict[str, PythonFinder] = None*)

Bases: `FinderBaseModel`

class **Config**

Bases: `object`

allow_mutation = True

arbitrary_types_allowed = True

include_private_attributes = True

keep_untouched = (<class 'cached_property.cached_property'>,)

validate_assignment = True

asdf_finder: Optional[PythonFinder]

static **check_for_asdf**()

static **check_for_pyenv**()

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

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

Parameters

- **path** (*str | None*) – 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

Return type
SystemPath

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

property **executables**

executables_tracking: List[PathEntry]

find_all(*executable*)

Search the path for an executable. Return all copies.

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

Return type
list[*PathEntry* | *PythonFinder*]

Returns
List[PathEntry]

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

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*)

Return type
PathEntry

property finders

finders_dict: Dict[str, PythonFinder]

get_path(*path*)

Return type
PythonFinder | *PathEntry*

get_pythons(*finder*)

Return type
Iterator

global_search: bool

ignore_unsupported: bool

only_python: bool

path_entries

path_order: List[str]

paths: Dict[str, Union[PythonFinder, PathEntry]]

pyenv_finder: Optional[PythonFinder]

python_executables

python_executables_tracking: Dict[str, PathEntry]

python_version_dict: Dict[Tuple, Any]

classmethod set_defaults(*values*)

system: bool

version_dict

version_dict_tracking: Dict[Tuple, List[PathEntry]]

which(*executable*)

Search for an executable on the path.

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

Return type
PathEntry | None

Returns
PathEntry object.

pythonfinder.models.path.**exists_and_is_accessible**(*path*)

pythonfinder.models.python module

class pythonfinder.models.python.**PythonFinder**(***data*)

Bases: *PathEntry*

class Config

Bases: object

allow_mutation = True

arbitrary_types_allowed = True

include_private_attributes = True

validate_assignment = True

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

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

Return type
PythonFinder

Returns
A new instance of the class.

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`

Return type
list[*PathEntry*]

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

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`

Return type
PathEntry | None

Returns
A `PathEntry` instance matching the version requested.

get_bin_dir(*base*)

Return type
`Path`

classmethod get_paths(*v*)

get_pythons()

Return type
`DefaultDict`[`str`, *PathEntry*]

get_version_order()

ignore_unsupported: bool

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

property is_asdf: bool

property is_pyenv: bool

paths: List

List of paths discovered during search

property pythons: dict

pythons_ref: Dict

root: Path

roots: Dict

The root locations used for discovery

sort_function: Optional[Callable]

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

classmethod version_from_bin_dir(*entry*)

Return type
PathEntry | None

version_glob_path: str

Glob path for python versions off of the root directory

property version_paths: Any

property versions: DefaultDict[tuple, PathEntry]

which(*name*)

Search in this path for an executable.

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

Return type
PathEntry | None

Returns
PathEntry instance.

class pythonfinder.models.python.**PythonVersion**(***data*)

Bases: FinderBaseModel

class Config

Bases: object

allow_mutation = True

arbitrary_types_allowed = True

include_private_attributes = True

validate_assignment = True

architecture: Optional[str]

as_dict()

Return type
dict[str, int | bool | Version | None]

as_major()

Return type
PythonVersion

as_minor()

Return type
PythonVersion

comes_from: Optional['PathEntry']

company: Optional[str]

classmethod create(***kwargs*)

Return type
PythonVersion

executable: Optional[Union[str, WindowsPath, Path]]

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.

Return type
PythonVersion

Returns
An instance of a 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.

Return type
PythonVersion

Returns
An instance of a PythonVersion.

`get_architecture`()

Return type
str

`is_debug: bool`

`is_devrelease: bool`

`is_postrelease: bool`

`is_prerelease: bool`

`major: int`

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

Return type
bool

`minor: Optional[int]`

`name: Optional[str]`

`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 unparsable type passed in

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

Return type
dict[str, str | int | Version]

Returns
A dictionary with metadata about the specified python version.

`classmethod parse_executable`(*path*)

Return type
dict[str, str | int | Version | None]

`patch: Optional[int]`

`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.

Return type
None

version: Optional[Version]

property version_sort: tuple[int, int, int | None, int, int]

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: tuple[int, int, int, bool, bool, bool]

Provides a version tuple for using as a dictionary key.

Returns
A tuple describing the python version meetadata contained.

class pythonfinder.models.python.**VersionMap**(***data*)

Bases: FinderBaseModel

class Config

Bases: object

allow_mutation = True

arbitrary_types_allowed = True

include_private_attributes = True

validate_assignment = True

add_entry(*entry*)

Return type
None

merge(*target*)

Return type
None

versions: DefaultDict[Tuple[int, Optional[int], Optional[int], bool, bool, bool], List[PathEntry]]

pythonfinder.models.windows module

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.**is_type_checking**()

pythonfinder.environment.**possibly_convert_to_windows_style_path**(*path*)

pythonfinder.environment.**set_asdf_paths**()

pythonfinder.environment.**set_pyenv_paths**()

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**(***data*)

Bases: FinderBaseModel

create_system_path()

Return type
SystemPath

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

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*)

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

Parameters

- **major** (*str | int | None*) – The major version to look for, or the full version, or the name of the target version.
- **minor** (*int | None*) – The minor version. If provided, disables string-based lookups from the major version field.
- **patch** (*int | None*) – The patch version.
- **pre** (*bool | None*) – If provided, specifies whether to search pre-releases.
- **dev** (*bool | None*) – If provided, whether to search dev-releases.
- **arch** (*str | None*) – If provided, which architecture to search.
- **name** (*str | None*) – *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)

Return type
PathEntry | None

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

global_search: bool

ignore_unsupported: bool

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

Return type
dict[str, Any]

path_prepend: Optional[str]

sort_by_path: bool

system: bool

system_path: Optional[SystemPath]

which(*exe*)

Return type
PathEntry | None

2.2.5 pythonfinder.utils module

pythonfinder.utils.**dedup**(*iterable*)

Deduplicate an iterable object like iter(set(iterable)) but order-reserved.

Return type
Iterable

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.

Return type
Path

Returns
A fully expanded Path object.

pythonfinder.utils.**expand_paths**(*path*, *only_python=True*)

Recursively expand a list or PathEntry instance

Parameters

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

Return type
Iterator

Returns
An iterator over the expanded set of path entries

pythonfinder.utils.**filter_pythons**(*path*)

Return all valid pythons in a given path

Return type
Iterable | Path

pythonfinder.utils.**get_python_version**(*path*)

Get python version string using subprocess from a given path.

Return type
str

pythonfinder.utils.**guess_company**(*path*)

Given a path to python, guess the company who created it

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

Return type
str | None

Returns
The guessed company

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.

Return type
bool

Returns
Whether the provided name looks like python.

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

Return type
str

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

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

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

Return type
dict[str, str | int | Version]

pythonfinder.utils.**path_is_executable**(*path*)

Determine whether the supplied path is executable.

Return type
bool

Returns
Whether the provided path is executable.

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.

Return type
bool

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

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.

Return type
bool

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

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

Return type
bool

Returns
Whether that path is a PythonCore path or not

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

Return type
tuple[str | int | None, str | int | None, str | int | None, str | None]

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

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

p

INDEX

D

E

F

G

I

K

L

M

N

O

P

R