

---

# **pythonfinder Documentation**

*Release 1.2.7dev*

**Dan Ryan <dan@danryan.co>**

**April 01, 2021**



## CONTENTS:

<b>1 PythonFinder: Cross Platform Search Tool for Finding Pythons</b>	<b>1</b>
1.1 Installation . . . . .	1
1.2 Usage . . . . .	1
1.3 Windows Support . . . . .	2
1.3.1 Finding Executables . . . . .	3
1.3.2 Architecture support . . . . .	3
1.4 Integrations . . . . .	3
<b>2 pythonfinder package</b>	<b>5</b>
2.1 Subpackages . . . . .	8
2.1.1 pythonfinder.models package . . . . .	8
2.1.1.1 Submodules . . . . .	8
2.2 Submodules . . . . .	16
2.2.1 pythonfinder.cli module . . . . .	16
2.2.2 pythonfinder.environment module . . . . .	16
2.2.3 pythonfinder.exceptions module . . . . .	17
2.2.4 pythonfinder.pythonfinder module . . . . .	17
2.2.5 pythonfinder.utils module . . . . .	18
<b>3 Indices and tables</b>	<b>21</b>
<b>Python Module Index</b>	<b>23</b>
<b>Index</b>	<b>25</b>



---

# 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





## 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)
```

```
refresh()
```

**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, ver-
    sions=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, System-
                                             Path__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, ig-
                    nore_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, System-
Path_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', ver-
sions=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 version 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,* *ver-*  
*sion\_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*)

**refresh** ()

**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*)





## INDICES AND TABLES

- genindex
- modindex
- search



## PYTHON MODULE INDEX

### p

- `pythonfinder`, 5
- `pythonfinder.cli`, 16
- `pythonfinder.environment`, 16
- `pythonfinder.exceptions`, 17
- `pythonfinder.models`, 8
- `pythonfinder.models.mixins`, 8
- `pythonfinder.models.path`, 10
- `pythonfinder.models.python`, 12
- `pythonfinder.models.windows`, 16
- `pythonfinder.pythonfinder`, 17
- `pythonfinder.utils`, 18



## A

add\_entry() (*pythonfinder.models.python.VersionMap method*), 16

architecture (*pythonfinder.models.python.PythonVersion attribute*), 14

as\_dict() (*pythonfinder.models.python.PythonVersion method*), 14

as\_major() (*pythonfinder.models.python.PythonVersion method*), 14

as\_minor() (*pythonfinder.models.python.PythonVersion method*), 14

as\_python() (*pythonfinder.models.mixins.BasePath property*), 8

## B

BaseFinder (*class in pythonfinder.models.mixins*), 8

BasePath (*class in pythonfinder.models.mixins*), 8

## C

check\_for\_asdf() (*pythonfinder.models.path.SystemPath static method*), 10

check\_for\_asdf() (*pythonfinder.SystemPath static method*), 6

check\_for\_pyenv() (*pythonfinder.models.path.SystemPath static method*), 10

check\_for\_pyenv() (*pythonfinder.SystemPath static method*), 6

children() (*pythonfinder.models.mixins.BasePath property*), 8

children() (*pythonfinder.models.path.PathEntry property*), 10

clear\_caches() (*pythonfinder.models.path.SystemPath method*), 10

clear\_caches() (*pythonfinder.SystemPath method*), 6

comes\_from (*pythonfinder.models.python.PythonVersion attribute*), 14

company (*pythonfinder.models.python.PythonVersion attribute*), 14

create() (*pythonfinder.models.mixins.BaseFinder class method*), 8

create() (*pythonfinder.models.path.PathEntry class method*), 10

create() (*pythonfinder.models.path.SystemPath class method*), 10

create() (*pythonfinder.models.path.VersionPath class method*), 12

create() (*pythonfinder.models.python.PythonFinder class method*), 12

create() (*pythonfinder.models.python.PythonVersion class method*), 14

create() (*pythonfinder.models.windows.WindowsFinder class method*), 16

create() (*pythonfinder.SystemPath class method*), 6

create() (*pythonfinder.WindowsFinder class method*), 8

create\_python\_version\_dict() (*pythonfinder.models.path.SystemPath method*), 11

create\_python\_version\_dict() (*pythonfinder.SystemPath method*), 6

create\_system\_path() (*pythonfinder.Finder method*), 5

create\_system\_path() (*pythonfinder.pythonfinder.Finder method*), 17

## D

dedup() (*in module pythonfinder.utils*), 18

## E

ensure\_path() (*in module pythonfinder.utils*), 18

executable (*pythonfinder.models.python.PythonVersion attribute*), 14

executables (*pythonfinder.models.path.SystemPath attribute*), 11

executables (*pythonfinder.SystemPath attribute*), 6

- `expand_paths()` (in module `pythonfinder.utils`), 18  
`expanded_paths()` (`pythonfinder.models.mixins.BaseFinder` property), 8  
`expanded_paths()` (`pythonfinder.models.python.PythonFinder` property), 12
- ## F
- `filter_pythons()` (in module `pythonfinder.utils`), 18  
`find_all()` (`pythonfinder.models.path.SystemPath` method), 11  
`find_all()` (`pythonfinder.SystemPath` method), 6  
`find_all_python_versions()` (`pythonfinder.Finder` method), 5  
`find_all_python_versions()` (`pythonfinder.models.mixins.BasePath` method), 8  
`find_all_python_versions()` (`pythonfinder.models.path.SystemPath` method), 11  
`find_all_python_versions()` (`pythonfinder.models.python.PythonFinder` method), 13  
`find_all_python_versions()` (`pythonfinder.models.windows.WindowsFinder` method), 16  
`find_all_python_versions()` (`pythonfinder.pythonfinder.Finder` method), 17  
`find_all_python_versions()` (`pythonfinder.SystemPath` method), 6  
`find_all_python_versions()` (`pythonfinder.WindowsFinder` method), 8  
`find_python_version()` (`pythonfinder.Finder` method), 5  
`find_python_version()` (`pythonfinder.models.mixins.BasePath` method), 9  
`find_python_version()` (`pythonfinder.models.path.SystemPath` method), 11  
`find_python_version()` (`pythonfinder.models.python.PythonFinder` method), 13  
`find_python_version()` (`pythonfinder.models.windows.WindowsFinder` method), 16  
`find_python_version()` (`pythonfinder.pythonfinder.Finder` method), 17  
`find_python_version()` (`pythonfinder.SystemPath` method), 7  
`find_python_version()` (`pythonfinder.WindowsFinder` method), 8  
`Finder` (class in `pythonfinder`), 5
- `Finder` (class in `pythonfinder.pythonfinder`), 17  
`finders()` (`pythonfinder.models.path.SystemPath` property), 12  
`finders()` (`pythonfinder.SystemPath` property), 7  
`from_path()` (`pythonfinder.models.python.PythonVersion` class method), 14  
`from_windows_launcher()` (`pythonfinder.models.python.PythonVersion` class method), 14
- ## G
- `get_architecture()` (`pythonfinder.models.python.PythonVersion` method), 15  
`get_bin_dir()` (`pythonfinder.models.python.PythonFinder` method), 13  
`get_name()` (`pythonfinder.models.mixins.BasePath` method), 9  
`get_path()` (`pythonfinder.models.path.SystemPath` method), 12  
`get_path()` (`pythonfinder.SystemPath` method), 7  
`get_paths()` (`pythonfinder.models.python.PythonFinder` method), 13  
`get_py_version()` (`pythonfinder.models.mixins.BasePath` method), 9  
`get_python_version()` (in module `pythonfinder.utils`), 18  
`get_pythons()` (`pythonfinder.models.path.SystemPath` method), 12  
`get_pythons()` (`pythonfinder.models.python.PythonFinder` method), 13  
`get_pythons()` (`pythonfinder.models.windows.WindowsFinder` method), 16  
`get_pythons()` (`pythonfinder.SystemPath` method), 7  
`get_pythons()` (`pythonfinder.WindowsFinder` method), 8  
`get_shim_paths()` (in module `pythonfinder.environment`), 16  
`get_version_order()` (`pythonfinder.models.python.PythonFinder` method), 13  
`get_versions()` (`pythonfinder.models.mixins.BaseFinder` method), 8  
`get_versions()` (`pythonfinder.models.windows.WindowsFinder` method), 16

- get\_versions() (*pythonfinder.WindowsFinder method*), 8
- guess\_company() (*in module pythonfinder.utils*), 18
- ## I
- ignore\_unsupported (*pythonfinder.models.python.PythonFinder attribute*), 13
- InvalidPythonVersion, 6, 17
- is\_asdf() (*pythonfinder.models.python.PythonFinder property*), 13
- is\_debug (*pythonfinder.models.python.PythonVersion attribute*), 15
- is\_devrelease (*pythonfinder.models.python.PythonVersion attribute*), 15
- is\_dir() (*pythonfinder.models.mixins.BasePath property*), 9
- is\_executable() (*pythonfinder.models.mixins.BasePath property*), 9
- is\_in\_path() (*in module pythonfinder.utils*), 18
- is\_postrelease (*pythonfinder.models.python.PythonVersion attribute*), 15
- is\_prerelease (*pythonfinder.models.python.PythonVersion attribute*), 15
- is\_pyenv() (*pythonfinder.models.python.PythonFinder property*), 13
- is\_python() (*pythonfinder.models.mixins.BasePath property*), 9
- is\_root (*pythonfinder.models.path.PathEntry attribute*), 10
- is\_type\_checking() (*in module pythonfinder.environment*), 16
- ## L
- looks\_like\_python() (*in module pythonfinder.utils*), 18
- ## M
- major (*pythonfinder.models.python.PythonVersion attribute*), 15
- matches() (*pythonfinder.models.python.PythonVersion method*), 15
- merge() (*pythonfinder.models.python.VersionMap method*), 16
- minor (*pythonfinder.models.python.PythonVersion attribute*), 15
- module
  - pythonfinder, 5
  - pythonfinder.cli, 16
  - pythonfinder.environment, 16
  - pythonfinder.exceptions, 17
  - pythonfinder.models, 8
  - pythonfinder.models.mixins, 8
  - pythonfinder.models.path, 10
  - pythonfinder.models.python, 12
  - pythonfinder.models.windows, 16
  - pythonfinder.pythonfinder, 17
  - pythonfinder.utils, 18
- ## N
- name (*pythonfinder.models.mixins.BasePath attribute*), 9
- name (*pythonfinder.models.python.PythonVersion attribute*), 15
- next() (*pythonfinder.models.mixins.BasePath method*), 9
- normalize\_path() (*in module pythonfinder.utils*), 18
- ## O
- only\_python (*pythonfinder.models.mixins.BasePath attribute*), 9
- optional\_instance\_of() (*in module pythonfinder.utils*), 18
- overload() (*in module pythonfinder.models.python*), 16
- ## P
- parse() (*pythonfinder.models.python.PythonVersion class method*), 15
- parse\_asdf\_version\_order() (*in module pythonfinder.utils*), 18
- parse\_executable() (*pythonfinder.models.python.PythonVersion class method*), 15
- parse\_major() (*pythonfinder.Finder class method*), 5
- parse\_major() (*pythonfinder.pythonfinder.Finder class method*), 17
- parse\_pyenv\_version\_order() (*in module pythonfinder.utils*), 19
- parse\_python\_version() (*in module pythonfinder.utils*), 19
- patch (*pythonfinder.models.python.PythonVersion attribute*), 15
- path (*pythonfinder.models.mixins.BasePath attribute*), 9
- path\_entries (*pythonfinder.models.path.SystemPath attribute*), 12
- path\_entries (*pythonfinder.SystemPath attribute*), 7
- path\_is\_executable() (*in module pythonfinder.utils*), 19
- path\_is\_known\_executable() (*in module pythonfinder.utils*), 19
- path\_is\_python() (*in module pythonfinder.utils*), 19
- path\_is\_pythoncore() (*in module pythonfinder.utils*), 19

- PathEntry (class in *pythonfinder.models.path*), 10
- paths (*pythonfinder.models.python.PythonFinder* attribute), 13
- py\_version() (*pythonfinder.models.mixins.BasePath* property), 9
- python\_executables (*pythonfinder.models.path.SystemPath* attribute), 12
- python\_executables (*pythonfinder.SystemPath* attribute), 7
- pythonfinder  
module, 5
- PythonFinder (class in *pythonfinder.models.python*), 12
- pythonfinder.cli  
module, 16
- pythonfinder.environment  
module, 16
- pythonfinder.exceptions  
module, 17
- pythonfinder.models  
module, 8
- pythonfinder.models.mixins  
module, 8
- pythonfinder.models.path  
module, 10
- pythonfinder.models.python  
module, 12
- pythonfinder.models.windows  
module, 16
- pythonfinder.pythonfinder  
module, 17
- pythonfinder.utils  
module, 18
- pythons() (*pythonfinder.models.mixins.BaseFinder* property), 8
- pythons() (*pythonfinder.models.mixins.BasePath* property), 9
- pythons() (*pythonfinder.models.python.PythonFinder* property), 13
- pythons() (*pythonfinder.models.windows.WindowsFinder* property), 16
- pythons() (*pythonfinder.WindowsFinder* property), 8
- PythonVersion (class in *pythonfinder.models.python*), 14
- ## R
- rehash() (*pythonfinder.Finder* method), 5
- rehash() (*pythonfinder.pythonfinder.Finder* method), 17
- reload\_finder() (*pythonfinder.models.path.SystemPath* method), 12
- reload\_finder() (*pythonfinder.SystemPath* method), 7
- reload\_system\_path() (*pythonfinder.Finder* method), 5
- reload\_system\_path() (*pythonfinder.pythonfinder.Finder* method), 17
- root (*pythonfinder.models.python.PythonFinder* attribute), 13
- roots (*pythonfinder.models.python.PythonFinder* attribute), 13
- ## S
- shim\_dir (*pythonfinder.models.python.PythonFinder* attribute), 14
- sort\_function (*pythonfinder.models.python.PythonFinder* attribute), 14
- split\_version\_and\_name() (in module *pythonfinder.utils*), 19
- SUBPROCESS\_TIMEOUT (in module *pythonfinder.environment*), 16
- system\_path() (*pythonfinder.Finder* property), 6
- system\_path() (*pythonfinder.pythonfinder.Finder* property), 17
- SystemPath (class in *pythonfinder*), 6
- SystemPath (class in *pythonfinder.models.path*), 10
- ## U
- unnest() (in module *pythonfinder.utils*), 19
- update\_metadata() (*pythonfinder.models.python.PythonVersion* method), 15
- ## V
- version (*pythonfinder.models.python.PythonVersion* attribute), 15
- version\_dict (*pythonfinder.models.path.SystemPath* attribute), 12
- version\_dict (*pythonfinder.SystemPath* attribute), 7
- version\_from\_bin\_dir() (*pythonfinder.models.python.PythonFinder* class method), 14
- version\_glob\_path (*pythonfinder.models.python.PythonFinder* attribute), 14
- version\_paths() (*pythonfinder.models.mixins.BaseFinder* property), 8
- version\_sort() (*pythonfinder.models.python.PythonVersion* property), 15
- version\_tuple() (*pythonfinder.models.python.PythonVersion* property), 15



VersionMap (*class in pythonfinder.models.python*), 16  
VersionPath (*class in pythonfinder.models.path*), 12  
versions() (*pythonfinder.models.python.PythonFinder property*),  
14  
versions() (*pythonfinder.models.windows.WindowsFinder prop-  
erty*), 16  
versions() (*pythonfinder.WindowsFinder property*), 8

## W

which() (*pythonfinder.Finder method*), 6  
which() (*pythonfinder.models.mixins.BasePath  
method*), 9  
which() (*pythonfinder.models.path.SystemPath  
method*), 12  
which() (*pythonfinder.models.python.PythonFinder  
method*), 14  
which() (*pythonfinder.pythonfinder.Finder method*), 17  
which() (*pythonfinder.SystemPath method*), 7  
windows\_finder() (*pythonfinder.Finder property*), 6  
windows\_finder() (*python-  
finder.pythonfinder.Finder property*), 18  
WindowsFinder (*class in pythonfinder*), 7  
WindowsFinder (*class in python-  
finder.models.windows*), 16