

---

# **pythonfinder Documentation**

***Release 2.0.6.dev0***

**Dan Ryan <[dan@danryan.co](mailto:dan@danryan.co)>**

**July 23, 2023**



## CONTENTS:

<b>1</b>	<b>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</b>	<b>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 . . . . .	18
2.2.1	pythonfinder.cli module . . . . .	18
2.2.2	pythonfinder.environment module . . . . .	18
2.2.3	pythonfinder.exceptions module . . . . .	19
2.2.4	pythonfinder.pythonfinder module . . . . .	19
2.2.5	pythonfinder.utils module . . . . .	20
<b>3</b>	<b>Indices and tables</b>	<b>23</b>
	<b>Python Module Index</b>	<b>25</b>
	<b>Index</b>	<b>27</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)
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 metadata 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.pyonfinder module

```
class pythonfinder.pyonfinder.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

`pythonfinder`, 5  
`pythonfinder.cli`, 18  
`pythonfinder.environment`, 18  
`pythonfinder.exceptions`, 19  
`pythonfinder.models`, 8  
`pythonfinder.models.mixins`, 8  
`pythonfinder.models.path`, 11  
`pythonfinder.models.python`, 13  
`pythonfinder.pythonfinder`, 19  
`pythonfinder.utils`, 20



# INDEX

## A

add\_entry() (`pythonfinder.models.python.VersionMap method`), 18  
allow\_mutation (`pythonfinder.models.mixins.PathEntry.Config attribute`), 8  
allow\_mutation (`pythonfinder.models.path.SystemPath.Config attribute`), 11  
allow\_mutation (`pythonfinder.models.python.PythonFinder.Config attribute`), 13  
allow\_mutation (`pythonfinder.models.python.PythonVersion.Config attribute`), 15  
allow\_mutation (`pythonfinder.models.python.VersionMap.Config attribute`), 18  
allow\_mutation (`pythonfinder.SystemPath.Config attribute`), 6  
arbitrary\_types\_allowed (`pythonfinder.models.mixins.PathEntry.Config attribute`), 8  
arbitrary\_types\_allowed (`pythonfinder.models.path.SystemPath.Config attribute`), 11  
arbitrary\_types\_allowed (`pythonfinder.models.python.PythonFinder.Config attribute`), 13  
arbitrary\_types\_allowed (`pythonfinder.models.python.PythonVersion.Config attribute`), 15  
arbitrary\_types\_allowed (`pythonfinder.models.python.VersionMap.Config attribute`), 18  
arbitrary\_types\_allowed (`pythonfinder.SystemPath.Config attribute`), 6  
architecture (`pythonfinder.models.python.PythonVersion attribute`), 15  
as\_dict() (`pythonfinder.models.python.PythonVersion method`), 16

as\_major() (`pythonfinder.models.python.PythonVersion method`), 16  
as\_minor() (`pythonfinder.models.python.PythonVersion method`), 16

as\_python (`pythonfinder.models.mixins.PathEntry property`), 8

asdf\_finder (`pythonfinder.models.path.SystemPath attribute`), 11  
asdf\_finder (`pythonfinder.SystemPath attribute`), 6

## C

check\_for\_asdf() (`pythonfinder.models.path.SystemPath static method`), 11  
check\_for\_asdf() (`pythonfinder.SystemPath static method`), 6

check\_for\_pyenv() (`pythonfinder.models.path.SystemPath static method`), 11  
check\_for\_pyenv() (`pythonfinder.SystemPath static method`), 6  
children (`pythonfinder.models.mixins.PathEntry property`), 9

children\_ref (`pythonfinder.models.mixins.PathEntry attribute`), 9  
comes\_from (`pythonfinder.models.python.PythonVersion attribute`), 16

company (`pythonfinder.models.python.PythonVersion attribute`), 16  
create() (`pythonfinder.models.mixins.PathEntry class method`), 9

create() (`pythonfinder.models.path.SystemPath class method`), 11  
create() (`pythonfinder.models.python.PythonFinder class method`), 13

create() (`pythonfinder.models.python.PythonVersion class method`), 16  
create() (`pythonfinder.SystemPath class method`), 6

create\_system\_path() (`pythonfinder.Finder method`), 5  
create\_system\_path() (`pythonfinder.pythonfinder.Finder method`), 19

## D

`dedup()` (*in module pythonfinder.utils*), 20

## E

`ensure_path()` (*in module pythonfinder.utils*), 20

`executable` (*pythonfinder.models.python.PythonVersion attribute*), 16

`executables` (*pythonfinder.models.path.SystemPath property*), 11

`executables` (*pythonfinder.SystemPath property*), 7

`executables_tracking` (*pythonfinder.models.path.SystemPath attribute*), 11

`executables_tracking` (*pythonfinder.SystemPath attribute*), 7

`exists_and_is_accessible()` (*in module pythonfinder.models.path*), 13

`expand_paths()` (*in module pythonfinder.utils*), 20

## F

`filter_pythons()` (*in module pythonfinder.utils*), 20

`find_all()` (*pythonfinder.models.path.SystemPath method*), 12

`find_all()` (*pythonfinder.SystemPath method*), 7

`find_all_python_versions()` (*pythonfinder.Finder method*), 5

`find_all_python_versions()` (*pythonfinder.models.mixins.PathEntry method*), 9

`find_all_python_versions()` (*pythonfinder.models.path.SystemPath method*), 12

`find_all_python_versions()` (*pythonfinder.models.python.PythonFinder method*), 13

`find_all_python_versions()` (*pythonfinder.pythonfinder.Finder method*), 19

`find_all_python_versions()` (*pythonfinder.SystemPath method*), 7

`find_python_version()` (*pythonfinder.Finder method*), 5

`find_python_version()` (*pythonfinder.models.mixins.PathEntry method*), 9

`find_python_version()` (*pythonfinder.models.path.SystemPath method*), 12

`find_python_version()` (*pythonfinder.models.python.PythonFinder method*), 14

`find_python_version()` (*pythonfinder.pythonfinder.Finder method*), 19

`find_python_version()` (*pythonfinder.SystemPath method*), 7

`Finder` (*class in pythonfinder*), 5

`Finder` (*class in pythonfinder.pythonfinder*), 19

`finders` (*pythonfinder.models.path.SystemPath property*), 12

`finders` (*pythonfinder.SystemPath property*), 7

`finders_dict` (*pythonfinder.models.path.SystemPath attribute*), 12

`finders_dict` (*pythonfinder.SystemPath attribute*), 7

`from_path()` (*pythonfinder.models.python.PythonVersion class method*), 16

`from_windows_launcher()` (*pythonfinder.models.python.PythonVersion class method*), 16

## G

`get_architecture()` (*pythonfinder.models.python.PythonVersion method*), 17

`get_bin_dir()` (*pythonfinder.models.python.PythonFinder method*), 14

`get_path()` (*pythonfinder.models.path.SystemPath method*), 12

`get_path()` (*pythonfinder.SystemPath method*), 7

`get_paths()` (*pythonfinder.models.python.PythonFinder class method*), 14

`get_py_version()` (*pythonfinder.models.mixins.PathEntry method*), 10

`get_python_version()` (*in module pythonfinder.utils*), 21

`get_pythons()` (*pythonfinder.models.path.SystemPath method*), 12

`get_pythons()` (*pythonfinder.models.python.PythonFinder method*), 14

`get_pythons()` (*pythonfinder.SystemPath method*), 7

`get_version_order()` (*pythonfinder.models.python.PythonFinder method*), 14

`global_search` (*pythonfinder.Finder attribute*), 5

`global_search` (*pythonfinder.models.path.SystemPath attribute*), 12

`global_search` (*pythonfinder.pythonfinder.Finder attribute*), 20

`global_search` (*pythonfinder.SystemPath attribute*), 7

`guess_company()` (*in module pythonfinder.utils*), 21

## I

`ignore_unsupported` (*pythonfinder.Finder attribute*), 5

`ignore_unsupported` (*pythonfinder.models.path.SystemPath attribute*), 12

ignore\_unsupported (pythonfinder.models.python.PythonFinder attribute), 14

ignore\_unsupported (pythonfinder.pythonfinder.Finder attribute), 20

ignore\_unsupported (pythonfinder.SystemPath attribute), 7

include\_private\_attributes (pythonfinder.models.mixins.PathEntry.Config attribute), 8

include\_private\_attributes (pythonfinder.models.path.SystemPath.Config attribute), 11

include\_private\_attributes (pythonfinder.models.python.PythonFinder.Config attribute), 13

include\_private\_attributes (pythonfinder.models.python.PythonVersion.Config attribute), 15

include\_private\_attributes (pythonfinder.models.python.VersionMap.Config attribute), 18

include\_private\_attributes (pythonfinder.SystemPath.Config attribute), 6

InvalidPythonVersion, 6, 19

is\_asdf (pythonfinder.models.python.PythonFinder property), 15

is\_debug (pythonfinder.models.python.PythonVersion attribute), 17

is\_devrelease (pythonfinder.models.python.PythonVersion attribute), 17

is\_dir (pythonfinder.models.mixins.PathEntry property), 10

is\_dir\_ref (pythonfinder.models.mixins.PathEntry attribute), 10

is\_executable (pythonfinder.models.mixins.PathEntry property), 10

is\_executable\_ref (pythonfinder.models.mixins.PathEntry attribute), 10

is\_in\_path() (in module pythonfinder.utils), 21

is\_postrelease (pythonfinder.models.python.PythonVersion attribute), 17

is\_prerelease (pythonfinder.models.python.PythonVersion attribute), 17

is\_pyenv (pythonfinder.models.python.PythonFinder property), 15

is\_python (pythonfinder.models.mixins.PathEntry property), 10

is\_python\_ref (pythonfinder.models.mixins.PathEntry attribute), 10

is\_root (pythonfinder.models.mixins.PathEntry attribute), 10

is\_type\_checking() (in module pythonfinder.environment), 18

## K

keep\_untouched (pythonfinder.models.path.SystemPath.Config attribute), 11

keep\_untouched (pythonfinder.SystemPath.Config attribute), 6

## L

looks\_like\_python() (in module pythonfinder.utils), 21

## M

major (pythonfinder.models.python.PythonVersion attribute), 17

matches() (pythonfinder.models.python.PythonVersion method), 17

merge() (pythonfinder.models.python.VersionMap method), 18

minor (pythonfinder.models.python.PythonVersion attribute), 17

module

- pythonfinder, 5
- pythonfinder.cli, 18
- pythonfinder.environment, 18
- pythonfinder.exceptions, 19
- pythonfinder.models, 8
- pythonfinder.models.mixins, 8
- pythonfinder.models.path, 11
- pythonfinder.models.python, 13
- pythonfinder.pythonfinder, 19
- pythonfinder.utils, 20

## N

name (pythonfinder.models.mixins.PathEntry attribute), 10

name (pythonfinder.models.python.PythonVersion attribute), 17

next() (pythonfinder.models.mixins.PathEntry method), 10

normalize\_path() (in module pythonfinder.utils), 21

## O

only\_python (pythonfinder.models.mixins.PathEntry attribute), 10

only\_python (pythonfinder.models.path.SystemPath attribute), 12

only\_python (pythonfinder.SystemPath attribute), 7

## P

parse() (`pythonfinder.models.python.PythonVersion` class method), 17  
parse\_asdf\_version\_order() (in module `pythonfinder.utils`), 21  
parse\_executable() (`pythonfinder.models.python.PythonVersion` class method), 17  
parse\_major() (`pythonfinder.Finder` class method), 5  
parse\_major() (`pythonfinder.pythonfinder.Finder` class method), 20  
parse\_pyenv\_version\_order() (in module `pythonfinder.utils`), 21  
parse\_python\_version() (in module `pythonfinder.utils`), 21  
patch (`pythonfinder.models.python.PythonVersion` attribute), 17  
path (`pythonfinder.models.mixins.PathEntry` attribute), 10  
path\_entries (`pythonfinder.models.path.SystemPath` attribute), 12  
path\_entries (`pythonfinder.SystemPath` attribute), 7  
path\_is\_executable() (in module `pythonfinder.utils`), 21  
path\_is\_known\_executable() (in module `pythonfinder.utils`), 21  
path\_is\_python() (in module `pythonfinder.utils`), 22  
path\_is\_pythoncore() (in module `pythonfinder.utils`), 22  
path\_order (`pythonfinder.models.path.SystemPath` attribute), 12  
path\_order (`pythonfinder.SystemPath` attribute), 8  
path\_prepend (`pythonfinder.Finder` attribute), 6  
path\_prepend (`pythonfinder.pythonfinder.Finder` attribute), 20  
PathEntry (class in `pythonfinder.models.mixins`), 8  
PathEntry.Config (class in `pythonfinder.models.mixins`), 8  
paths (`pythonfinder.models.path.SystemPath` attribute), 12  
paths (`pythonfinder.models.python.PythonFinder` attribute), 15  
paths (`pythonfinder.SystemPath` attribute), 8  
possibly\_convert\_to\_windows\_style\_path() (in module `pythonfinder.environment`), 19  
py\_version (`pythonfinder.models.mixins.PathEntry` property), 10  
py\_version\_ref (`pythonfinder.models.mixins.PathEntry` attribute), 10  
pyenv\_finder (`pythonfinder.models.path.SystemPath` attribute), 12  
pyenv\_finder (`pythonfinder.SystemPath` attribute), 8  
python\_executables (`pythonfinder.models.path.SystemPath` attribute), 12  
python\_executables\_tracking (`pythonfinder.models.path.SystemPath` attribute), 12  
python\_executables\_tracking (`pythonfinder.SystemPath` attribute), 8  
python\_version\_dict (`pythonfinder.models.path.SystemPath` attribute), 12  
python\_version\_dict (`pythonfinder.SystemPath` attribute), 8  
pythonfinder module, 5  
PythonFinder (class in `pythonfinder.models.python`), 13  
pythonfinder.cli module, 18  
PythonFinder.Config (class in `pythonfinder.models.python`), 13  
pythonfinder.environment module, 18  
pythonfinder.exceptions module, 19  
pythonfinder.models module, 8  
pythonfinder.models.mixins module, 8  
pythonfinder.models.path module, 11  
pythonfinder.models.python module, 13  
pythonfinder.pythonfinder module, 19  
pythonfinder.utils module, 20  
pythons (`pythonfinder.models.mixins.PathEntry` property), 10  
pythons (`pythonfinder.models.python.PythonFinder` property), 15  
pythons\_ref (`pythonfinder.models.mixins.PathEntry` attribute), 10  
pythons\_ref (`pythonfinder.models.python.PythonFinder` attribute), 15  
PythonVersion (class in `pythonfinder.models.python`), 15  
PythonVersion.Config (class in `pythonfinder.models.python`), 15

## R

root (`pythonfinder.models.python.PythonFinder` attribute), 15

**S**

- roots (`pythonfinder.models.python.PythonFinder` attribute), 15
- S**
- set\_asdf\_paths() (in module `pythonfinder.environment`), 19
- set\_children() (`pythonfinder.models.mixins.PathEntry` class method), 10
- set\_defaults() (`pythonfinder.models.path.SystemPath` class method), 12
- set\_defaults() (`pythonfinder.SystemPath` class method), 8
- set\_pyenv\_paths() (in module `pythonfinder.environment`), 19
- sort\_by\_path (`pythonfinder.Finder` attribute), 6
- sort\_by\_path (`pythonfinder.pythonfinder.Finder` attribute), 20
- sort\_function (`pythonfinder.models.python.PythonFinder` attribute), 15
- split\_version\_and\_name() (in module `pythonfinder.utils`), 22
- SUBPROCESS\_TIMEOUT (in module `pythonfinder.environment`), 18
- system (`pythonfinder.Finder` attribute), 6
- system (`pythonfinder.models.path.SystemPath` attribute), 12
- system (`pythonfinder.pythonfinder.Finder` attribute), 20
- system (`pythonfinder.SystemPath` attribute), 8
- system\_path (`pythonfinder.Finder` attribute), 6
- system\_path (`pythonfinder.pythonfinder.Finder` attribute), 20
- SystemPath (class in `pythonfinder`), 6
- SystemPath (class in `pythonfinder.models.path`), 11
- SystemPath.Config (class in `pythonfinder`), 6
- SystemPath.Config (class in `pythonfinder.models.path`), 11

**U**

- unnest() (in module `pythonfinder.utils`), 22
- update\_metadata() (`pythonfinder.models.python.PythonVersion` method), 17

**V**

- validate\_assignment (`pythonfinder.models.mixins.PathEntry.Config` attribute), 8
- validate\_assignment (`pythonfinder.models.path.SystemPath.Config` attribute), 11

**W**

- validate\_assignment (`pythonfinder.models.python.PythonFinder.Config` attribute), 13
- validate\_assignment (`pythonfinder.models.python.PythonVersion.Config` attribute), 15
- validate\_assignment (`pythonfinder.models.python.VersionMap.Config` attribute), 18
- validate\_assignment (`pythonfinder.SystemPath.Config` attribute), 6
- version (`pythonfinder.models.python.PythonVersion` attribute), 17
- version\_dict (`pythonfinder.models.path.SystemPath` attribute), 13
- version\_dict (`pythonfinder.SystemPath` attribute), 8
- version\_dict\_tracking (`pythonfinder.models.path.SystemPath` attribute), 13
- version\_dict\_tracking (`pythonfinder.SystemPath` attribute), 8
- version\_from\_bin\_dir() (`pythonfinder.models.python.PythonFinder` class method), 15
- version\_glob\_path (`pythonfinder.models.python.PythonFinder` attribute), 15
- version\_paths (`pythonfinder.models.python.PythonFinder` property), 15
- version\_sort (`pythonfinder.models.python.PythonVersion` property), 18
- version\_tuple (`pythonfinder.models.python.PythonVersion` property), 18
- VersionMap (class in `pythonfinder.models.python`), 18
- VersionMap.Config (class in `pythonfinder.models.python`), 18
- versions (`pythonfinder.models.python.PythonFinder` property), 15
- versions (`pythonfinder.models.python.VersionMap` attribute), 18

**W**

- which() (`pythonfinder.Finder` method), 6
- which() (`pythonfinder.models.mixins.PathEntry` method), 10
- which() (`pythonfinder.models.path.SystemPath` method), 13
- which() (`pythonfinder.models.python.PythonFinder` method), 15
- which() (`pythonfinder.pythonfinder.Finder` method), 20
- which() (`pythonfinder.SystemPath` method), 8