Package 'reticulate'

July 25, 2019

Type Package

Title Interface to 'Python'

Version 1.13

Description Interface to 'Python' modules, classes, and functions. When calling into 'Python', R data types are automatically converted to their equivalent 'Python' types. When values are returned from 'Python' to R they are converted back to R types. Compatible with all versions of 'Python' >= 2.7.

License Apache License 2.0

URL `https://github.com/rstudio/reticulate`

BugReports `https://github.com/rstudio/reticulate/issues`

SystemRequirements Python (>= 2.7.0)

Encoding UTF-8

LazyData true

Depends R (>= 3.0)

Imports utils, graphics, jsonlite, Rcpp (>= 0.12.7), Matrix

Suggests testthat, knitr, callr, rmarkdown

LinkingTo Rcpp

RoxygenNote 6.1.1

VignetteBuilder knitr

NeedsCompilation yes

Author Kevin Ushey [aut, cre],
JJ Allaire [aut],
RStudio [cph, fnd],
Yuan Tang [aut, cph] (<https://orcid.org/0000-0001-5243-233X>),
Dirk Eddelbuettel [ctb, cph],
Bryan Lewis [ctb, cph],
Marcus Geelnard [ctb, cph] (TinyThread library,
http://tinythreadpp.bitsnbites.eu/)

Maintainer Kevin Ushey <kevin@rstudio.com>

Repository CRAN

Date/Publication 2019-07-24 22:40:03 UTC

R topics documented:

`array_reshape` *Reshape an Array*

Description

Reshape (reindex) a multi-dimensional array, using row-major (C-style) reshaping semantics by default.

Usage

```
array_reshape(x, dim, order = c("C", "F"))
```

Arguments

`x`	An array
`dim`	The new dimensions to be set on the array.
`order`	The order in which elements of `x` should be read during the rearrangement. `"C"` means elements should be read in row-major order, with the last index changing fastest; `"F"` means elements should be read in column-major order, with the first index changing fastest.

Details

This function differs from e.g. `dim(x) <-dim` in a very important way: by default, `array_reshape()` will fill the new dimensions in row-major (`C`-style) ordering, while `dim<-()` will fill new dimensions in column-major (`F`ortran-style) ordering. This is done to be consistent with libraries like NumPy, Keras, and TensorFlow, which default to this sort of ordering when reshaping arrays. See the examples for why this difference may be important.

Examples

```
## Not run:
# let's construct a 2x2 array from a vector of 4 elements
x <- 1:4

# rearrange will fill the array row-wise
array_reshape(x, c(2, 2))
#      [,1] [,2]
# [1,]    1    2
# [2,]    3    4
# setting the dimensions 'fills' the array col-wise
dim(x) <- c(2, 2)
x
#      [,1] [,2]
# [1,]    1    3
# [2,]    2    4

## End(Not run)
```

as.character.python.builtin.bytes *Convert Python bytes to an R character vector*

Description

Convert Python bytes to an R character vector

Usage

```
## S3 method for class 'python.builtin.bytes'
as.character(x, encoding = "utf-8",
  errors = "strict", ...)
```

Arguments

x	object to be coerced or tested.
encoding	Encoding to use for conversion (defaults to utf-8)
errors	Policy for handling conversion errors. Default is 'strict' which raises an error. Other possible values are 'ignore' and 'replace'
...	further arguments passed to or from other methods.

dict *Create Python dictionary*

Description

Create a Python dictionary object, including a dictionary whose keys are other Python objects rather than character vectors.

Usage

```
dict(..., convert = FALSE)

py_dict(keys, values, convert = FALSE)
```

Arguments

...	Name/value pairs for dictionary (or a single named list to be converted to a dictionary).
convert	TRUE to automatically convert Python objects to their R equivalent. If you pass FALSE you can do manual conversion using the py_to_r() function.
keys	Keys to dictionary (can be Python objects)
values	Values for dictionary

Value

A Python dictionary

Note

The returned dictionary will not automatically convert its elements from Python to R. You can do manual converstion with the `py_to_r()` function or pass `convert = TRUE` to request automatic conversion.

`eng_python` *A reticulate Engine for Knitr*

Description

This provides a `reticulate` engine for `knitr`, suitable for usage when attempting to render Python chunks. Using this engine allows for shared state between Python chunks in a document – that is, variables defined by one Python chunk can be used by later Python chunks.

Usage

```
eng_python(options)
```

Arguments

`options`	Chunk options, as provided by `knitr` during chunk execution.

Details

The engine can be activated by setting (for example)

```
knitr::knit_engines$set(python = reticulate::eng_python)
```

Typically, this will be set within a document's setup chunk, or by the environment requesting that Python chunks be processed by this engine. Note that `knitr` (since version 1.18) will use the `reticulate` engine by default when executing Python chunks within an R Markdown document.

`import` *Import a Python module*

Description

Import the specified Python module for calling from R.

Usage

```
import(module, as = NULL, convert = TRUE, delay_load = FALSE)

import_main(convert = TRUE)

import_builtins(convert = TRUE)

import_from_path(module, path = ".", convert = TRUE)
```

Arguments

`module`	Module name
`as`	Alias for module name (affects names of R classes). Note that this is an advanced parameter that should generally only be used in package development (since it affects the S3 name of the imported class and can therefore interfere with S3 method dispatching).
`convert`	`TRUE` to automatically convert Python objects to their R equivalent. If you pass `FALSE` you can do manual conversion using the `py_to_r()` function.
`delay_load`	`TRUE` to delay loading the module until it is first used. `FALSE` to load the module immediately. If a function is provided then it will be called once the module is loaded. If a list containing `on_load()` and `on_error(e)` elements is provided then `on_load()` will be called on successful load and `on_error(e)` if an error occurs.
`path`	Path to import from

Details

The `import_from_path` function imports a Python module from an arbitrary filesystem path (the directory of the specified python script is automatically added to the `sys.path`).

Value

A Python module

Examples

```
## Not run:
main <- import_main()
sys <- import("sys")

## End(Not run)
```

`iterate` *Traverse a Python iterator or generator*

Description

Traverse a Python iterator or generator

Usage

```
iterate(it, f = base::identity, simplify = TRUE)

iter_next(it, completed = NULL)

as_iterator(x)
```

Arguments

`it`	Python iterator or generator
`f`	Function to apply to each item. By default applies the `identity` function which just reflects back the value of the item.
`simplify`	Should the result be simplified to a vector if possible?
`completed`	Sentinel value to return from `iter_next()` if the iteration completes (defaults to `NULL` but can be any R value you specify).
`x`	Python iterator or iterable

Details

Simplification is only attempted all elements are length 1 vectors of type "character", "complex", "double", "integer", or "logical".

Value

For `iterate()`, A list or vector containing the results of calling `f` on each item in `x` (invisibly); For `iter_next()`, the next value in the iteration (or the sentinel `completed` value if the iteration is complete).

np_array *NumPy array*

Description

Create NumPy arrays and convert the data type and in-memory ordering of existing NumPy arrays.

Usage

```
np_array(data, dtype = NULL, order = "C")
```

Arguments

data	Vector or existing NumPy array providing data for the array
dtype	Numpy data type (e.g. "float32", "float64", etc.)
order	Memory ordering for array. "C" means C order, "F" means Fortran order.

Value

A NumPy array object.

py *Interact with the Python Main Module*

Description

The py object provides a means for interacting with the Python main session directly from R. Python objects accessed through py are automatically converted into R objects, and can be used with any other R functions as needed.

Usage

```
py
```

Format

An R object acting as an interface to the Python main module.

py_available *Check if Python is available on this system*

Description

Check if Python is available on this system

Usage

```
py_available(initialize = FALSE)

py_numpy_available(initialize = FALSE)
```

Arguments

initialize	TRUE to attempt to initialize Python bindings if they aren't yet available (defaults to FALSE).

Value

Logical indicating whether Python is initialized.

Note

The py_numpy_available function is a superset of the py_available function (it calls py_available first before checking for NumPy).

py_capture_output *Capture and return Python output*

Description

Capture and return Python output

Usage

```
py_capture_output(expr, type = c("stdout", "stderr"))
```

Arguments

expr	Expression to capture stdout for
type	Streams to capture (defaults to both stdout and stderr)

Value

Character vector with output

py_config *Python configuration*

Description

Information on Python and Numpy versions detected

Usage

```
py_config()
```

Value

Python configuration object; Logical indicating whether Python bindings are available

py_del_item *Delete / remove an item from a Python object*

Description

Delete an item associated with a Python object, as through its `__delitem__` method.

Usage

```
py_del_item(x, name)
```

Arguments

`x`	A Python object.
`name`	The item name.

Value

The (mutated) object `x`, invisibly.

See Also

Other item-related APIs: `py_get_item`, `py_set_item`

py_discover_config *Discover the version of Python to use with reticulate.*

Description

This function enables callers to check which versions of Python will be discovered on a system as well as which one will be chosen for use with reticulate.

Usage

```
py_discover_config(required_module = NULL, use_environment = NULL)
```

Arguments

required_module	A optional module name that must be available in order for a version of Python to be used.
use_environment	An optional virtual/conda environment name to prefer in the search

Value

Python configuration object.

py_func *Wrap an R function in a Python function with the same signature.*

Description

This function could wrap an R function in a Python function with the same signature. Note that the signature of the R function must not contain esoteric Python-incompatible constructs.

Usage

```
py_func(f)
```

Arguments

f	An R function

Value

A Python function that calls the R function `f` with the same signature.

py_function_custom_scaffold *Custom Scaffolding of R Wrappers for Python Functions*

Description

This function can be used to generate R wrapper for a specified Python function while allowing to inject custom code for critical parts of the wrapper generation, such as process the any part of the docs obtained from `py_function_docs()` and append additional roxygen fields. The result from execution of `python_function` is assigned to a variable called `python_function_result` that can also be processed by `postprocess_fn` before writing the closing curly braces for the generated wrapper function.

Usage

```
py_function_custom_scaffold(python_function, r_function = NULL,
  additional_roxygen_fields = NULL, process_docs_fn = function(docs)
  docs, process_param_fn = function(param, docs) param,
  process_param_doc_fn = function(param_doc, docs) param_doc,
  postprocess_fn = function() { }, file_name = NULL)
```

Arguments

`python_function`
: Fully qualfied name of Python function or class constructor (e.g. `tf$layers$average_pooling1d`

`r_function`
: Name of R function to generate (defaults to name of Python function if not specified)

`additional_roxygen_fields`
: A list of additional roxygen fields to write to the roxygen docs, e.g. `list(export = "",rdname = "generated-wrappers")`.

`process_docs_fn`
: A function to process docs obtained from `reticulate::py_function_docs(python_functio`

`process_param_fn`
: A function to process each parameter needed for `python_funcion` before executing `python_funcion`.

`process_param_doc_fn`
: A function to process the roxygen docstring for each parameter.

`postprocess_fn`
: A function to inject any custom code in the form of a string before writing the closing curly braces for the generated wrapper function.

`file_name`
: The file name to write the generated wrapper function to. If `NULL`, the generated wrapper will only be printed out in the console.

Examples

```
## Not run:

library(tensorflow)
library(stringr)

# Example of a `process_param_fn` to cast parameters with default values
# that contains "L" to integers
process_int_param_fn <- function(param, docs) {
  # Extract the list of parameters that have integer values as default
  int_params <- gsub(
    " = [-]?[0-9]+L",
    "",
    str_extract_all(docs$signature, "[A-z]+ = [-]?[0-9]+L")[[1]])
  # Explicitly cast parameter in the list obtained above to integer
  if (param %in% int_params) {
    param <- paste0("as.integer(", param, ")")
  }
  param
}

# Note that since the default value of parameter `k` is `1L`. It is wrapped
# by `as.integer()` to ensure it's casted to integer before sending it to `tf$nn$top_k`
# for execution. We then print out the python function result.
py_function_custom_scaffold(
  "tf$nn$top_k",
  r_function = "top_k",
  process_param_fn = process_int_param_fn,
  postprocess_fn = function() { "print(python_function_result)" })


## End(Not run)
```

`py_get_attr` *Get an attribute of a Python object*

Description

Get an attribute of a Python object

Usage

```
py_get_attr(x, name, silent = FALSE)
```

Arguments

`x`	Python object
`name`	Attribute name
`silent`	`TRUE` to return `NULL` if the attribute doesn't exist (default is `FALSE` which will raise an error)

Value

Attribute of Python object

---

py_get_item *Get an item from a Python object*

Description

Retrieve an item from a Python object, similar to how `x[name]` might be used in Python code to access an item indexed by `key` on an object `x`. The object's `__getitem__` method will be called.

Usage

```
py_get_item(x, key, silent = FALSE)
```

Arguments

`x`	A Python object.
`key`	The key used for item lookup.
`silent`	Boolean; when `TRUE`, attempts to access missing items will return `NULL` rather than throw an error.

See Also

Other item-related APIs: `py_del_item`, `py_set_item`

---

py_has_attr *Check if a Python object has an attribute*

Description

Check whether a Python object `x` has an attribute `name`.

Usage

```
py_has_attr(x, name)
```

Arguments

`x`	A python object.
`name`	The attribute to be accessed.

Value

`TRUE` if the object has the attribute `name`, and `FALSE` otherwise.

py_help *Documentation for Python Objects*

Description

Documentation for Python Objects

Usage

```
py_help(object)
```

Arguments

object	Object to print documentation for

py_id *Unique identifer for Python object*

Description

Get a globally unique identifer for a Python object.

Usage

```
py_id(object)
```

Arguments

object	Python object

Value

Unique identifer (as integer) or NULL

Note

In the current implementation of CPython this is the memory address of the object.

`py_install` *Install Python packages*

Description

Install Python packages into a virtual environment or Conda environment.

Usage

```
py_install(packages, envname = NULL, method = c("auto", "virtualenv",
  "conda"), conda = "auto", python_version = NULL, ...)
```

Arguments

`packages`	Character vector with package names to install.
`envname`	The name, or full path, of the environment in which Python packages are to be installed. When `NULL` (the default), the active environment as set by the `RETICULATE_PYTHON_ENV` variable will be used; if that is unset, then the `r-reticulate` environment will be used.
`method`	Installation method. By default, "auto" automatically finds a method that will work in the local environment. Change the default to force a specific installation method. Note that the "virtualenv" method is not available on Windows.
`conda`	Path to conda executable (or "auto" to find conda using the PATH and other conventional install locations).
`python_version`	The requested Python version. Ignored when attempting to install with a Python virtual environment.
`...`	Additional arguments passed to `conda_install()` or `virtualenv_install()`.

Details

On Linux and OS X the "virtualenv" method will be used by default ("conda" will be used if virtualenv isn't available). On Windows, the "conda" method is always used.

See Also

conda-tools, virtualenv-tools

py_is_null_xptr *Check if a Python object is a null externalptr*

Description

Check if a Python object is a null externalptr

Usage

```
py_is_null_xptr(x)

py_validate_xptr(x)
```

Arguments

x	Python object

Details

When Python objects are serialized within a persisted R environment (e.g. .RData file) they are deserialized into null externalptr objects (since the Python session they were originally connected to no longer exists). This function allows you to safely check whether whether a Python object is a null externalptr.

The py_validate function is a convenience function which calls py_is_null_xptr and throws an error in the case that the xptr is NULL.

Value

Logical indicating whether the object is a null externalptr

py_iterator *Create a Python iterator from an R function*

Description

Create a Python iterator from an R function

Usage

```
py_iterator(fn, completed = NULL)
```

Arguments

fn	R function with no arguments.
completed	Special sentinel return value which indicates that iteration is complete (defaults to NULL)

Details

Python generators are functions that implement the Python iterator protocol. In Python, values are returned using the `yield` keyword. In R, values are simply returned from the function.

In Python, the `yield` keyword enables successive iterations to use the state of previous iterations. In R, this can be done by returning a function that mutates its enclosing environment via the `<<-` operator. For example:

```
sequence_generator <- function(start) {
  value <- start
  function() {
    value <<- value + 1
    value
  }
}
```

Then create an iterator using `py_iterator()`:

```
g <- py_iterator(sequence_generator(10))
```

Value

Python iterator which calls the R function for each iteration.

Ending Iteration

In Python, returning from a function without calling `yield` indicates the end of the iteration. In R however, `return` is used to yield values, so the end of iteration is indicated by a special return value (`NULL` by default, however this can be changed using the `completed` parameter). For example:

```
sequence_generator <-function(start) {
  value <- start
  function() {
    value <<- value + 1
    if (value < 100)
      value
    else
      NULL
  }
}
```

Threading

Some Python APIs use generators to parallelize operations by calling the generator on a background thread and then consuming its results on the foreground thread. The `py_iterator()` function creates threadsafe iterators by ensuring that the R function is always called on the main thread (to be compatible with R's single-threaded runtime) even if the generator is run on a background thread.

py_last_error *Get or clear the last Python error encountered*

Description

Get or clear the last Python error encountered

Usage

```
py_last_error()

py_clear_last_error()
```

Value

For `py_last_error()`, a list with the type, value, and traceback for the last Python error encountered (can be `NULL` if no error has yet been encountered).

py_len *Length of Python object*

Description

Get the length of a Python object (equivalent to the Python `len()` built in function).

Usage

```
py_len(x)
```

Arguments

`x`	Python object

Value

Length as integer

py_list_attributes *List all attributes of a Python object*

Description

List all attributes of a Python object

Usage

```
py_list_attributes(x)
```

Arguments

x	Python object

Value

Character vector of attributes

py_main_thread_func *Create a Python function that will always be called on the main thread*

Description

This function is helpful when you need to provide a callback to a Python library which may invoke the callback on a background thread. As R functions must run on the main thread, wrapping the R function with `py_main_thread_func()` will ensure that R code is only executed on the main thread.

Usage

```
py_main_thread_func(f)
```

Arguments

f	An R function with artibrary arguments

Value

A Python function that delegates to the passed R function, which is guaranteed to always be called on the main thread.

py_module_available *Check if a Python module is available on this system.*

Description

Check if a Python module is available on this system.

Usage

```
py_module_available(module)
```

Arguments

module	Name of module

Value

Logical indicating whether module is available

py_run *Run Python code*

Description

Execute code within the the __main__ Python module.

Usage

```
py_run_string(code, local = FALSE, convert = TRUE)

py_run_file(file, local = FALSE, convert = TRUE)

py_eval(code, convert = TRUE)
```

Arguments

code	Code to execute
local	Whether to create objects in a local/private namespace (if FALSE, objects are created within the main module).
convert	TRUE to automatically convert Python objects to their R equivalent. If you pass FALSE you can do manual conversion using the py_to_r() function.
file	Source file

Value

For `py_eval()`, the result of evaluating the expression; For `py_run_string()` and `py_run_file()`, the dictionary associated with the code execution.

`py_save_object` *Save and load Python objects with pickle*

Description

Save and load Python objects with pickle

Usage

```
py_save_object(object, filename, pickle = "pickle", ...)

py_load_object(filename, pickle = "pickle", ...)
```

Arguments

`object`	Object to save
`filename`	File name
`pickle`	The implementation of pickle to use (defaults to "pickle" but could e.g. also be "cPickle")
`...`	Optional arguments to be passed to the `load()` function defined by the associated `pickle` module.

`py_set_attr` *Set an attribute of a Python object*

Description

Set an attribute of a Python object

Usage

```
py_set_attr(x, name, value)
```

Arguments

`x`	Python object
`name`	Attribute name
`value`	Attribute value

py_set_item *Set an item for a Python object*

Description

Set an item on a Python object, similar to how `x[name] = value` might be used in Python code to set an item called `name` with value `value` on object `x`. The object's `__setitem__` method will be called.

Usage

```
py_set_item(x, name, value)
```

Arguments

`x`	A Python object.
`name`	The item name.
`value`	The item value.

Value

The (mutated) object `x`, invisibly.

See Also

Other item-related APIs: `py_del_item`, `py_get_item`

py_set_seed *Set Python and NumPy random seeds*

Description

Set various random seeds required to ensure reproducible results. The provided `seed` value will establish a new random seed for Python and NumPy, and will also (by default) disable hash randomization.

Usage

```
py_set_seed(seed, disable_hash_randomization = TRUE)
```

Arguments

`seed`	A single value, interpreted as an integer
`disable_hash_randomization`	Disable hash randomization, which is another common source of variable results. See `https://docs.python.org/3.4/using/cmdline.html#envvar-PYTHONHASHSEED`

Details

This function does not set the R random seed, for that you should call `set.seed()`.

`py_str` *An S3 method for getting the string representation of a Python object*

Description

An S3 method for getting the string representation of a Python object

Usage

```
py_str(object, ...)
```

Arguments

`object`	Python object
`...`	Unused

Details

The default implementation will call `PyObject_Str` on the object.

Value

Character vector

`py_suppress_warnings` *Suppress Python warnings for an expression*

Description

Suppress Python warnings for an expression

Usage

```
py_suppress_warnings(expr)
```

Arguments

`expr`	Expression to suppress warnings for

Value

Result of evaluating expression

py_unicode *Convert to Python Unicode Object*

Description

Convert to Python Unicode Object

Usage

```
py_unicode(str)
```

Arguments

str	Single element character vector to convert

Details

By default R character vectors are converted to Python strings. In Python 3 these values are unicode objects however in Python 2 they are 8-bit string objects. This function enables you to obtain a Python unicode object from an R character vector when running under Python 2 (under Python 3 a standard Python string object is returend).

r-py-conversion *Convert between Python and R objects*

Description

Convert between Python and R objects

Usage

```
r_to_py(x, convert = FALSE)

py_to_r(x)
```

Arguments

x	A Python object.
convert	TRUE to automatically convert Python objects to their R equivalent. If you pass FALSE you can do manual conversion using the py_to_r() function.

Value

An R object, as converted from the Python object.

repl_python *Run a Python REPL*

Description

This function provides a Python REPL in the R session, which can be used to interactively run Python code. All code executed within the REPL is run within the Python main module, and any generated Python objects will persist in the Python session after the REPL is detached.

Usage

```
repl_python(module = NULL, quiet = getOption("reticulate.repl.quiet",
  default = FALSE))
```

Arguments

module	An (optional) Python module to be imported before the REPL is launched.
quiet	Boolean; print a startup banner when launching the REPL? If TRUE, the banner will be suppressed.

Details

When working with R and Python scripts interactively, one can activate the Python REPL with `repl_python()`, run Python code, and later run `exit` to return to the R console.

See Also

py, for accessing objects created using the Python REPL.

Examples

```
## Not run:

# enter the Python REPL, create a dictionary, and exit
repl_python()
dictionary = {'alpha': 1, 'beta': 2}
exit

# access the created dictionary from R
py$dictionary
# $alpha
# [1] 1
#
# $beta
# [1] 2


## End(Not run)
```

reticulate *R Interface to Python*

Description

R interface to Python modules, classes, and functions. When calling into Python R data types are automatically converted to their equivalent Python types. When values are returned from Python to R they are converted back to R types. The reticulate package is compatible with all versions of Python >= 2.7. Integration with NumPy requires NumPy version 1.6 or higher.

source_python *Read and evaluate a Python script*

Description

Evaluate a Python script within the Python main module, then make all public (non-module) objects within the main Python module available within the specified R environment.

Usage

```
source_python(file, envir = parent.frame(), convert = TRUE)
```

Arguments

file	Source file
envir	The environment to assign Python objects into (for example, parent.frame() or globalenv()). Specify NULL to not assign Python objects.
convert	TRUE to automatically convert Python objects to their R equivalent. If you pass FALSE you can do manual conversion using the py_to_r() function.

Details

To prevent assignment of objects into R, pass NULL for the envir parameter.

tuple *Create Python tuple*

Description

Create a Python tuple object

Usage

```
tuple(..., convert = FALSE)
```

Arguments

...	Values for tuple (or a single list to be converted to a tuple).
convert	TRUE to automatically convert Python objects to their R equivalent. If you pass FALSE you can do manual conversion using the py_to_r() function.

Value

A Python tuple

Note

The returned tuple will not automatically convert its elements from Python to R. You can do manual converstion with the py_to_r() function or pass convert = TRUE to request automatic conversion.

use_python *Configure which version of Python to use*

Description

Configure which version of Python to use

Usage

```
use_python(python, required = FALSE)

use_virtualenv(virtualenv = NULL, required = FALSE)

use_condaenv(condaenv = NULL, conda = "auto", required = FALSE)
```

Arguments

python	Path to Python binary
required	Is this version of Python required? If TRUE then an error occurs if it's not located. Otherwise, the version is taken as a hint only and scanning for other versions will still proceed.
virtualenv	Directory of Python virtualenv
condaenv	Name of Conda environment
conda	Conda executable. Default is "auto", which checks the PATH as well as other standard locations for Anaconda installations.

---

virtualenv-tools *Interface to Python Virtual Environments*

Description

R functions for managing Python virtual environments[1].

Usage

```
virtualenv_list()

virtualenv_create(envname = NULL, python = NULL)

virtualenv_install(envname = NULL, packages, ignore_installed = TRUE,
  ...)

virtualenv_remove(envname = NULL, packages = NULL,
  confirm = interactive())

virtualenv_root()

virtualenv_python(envname = NULL)
```

Arguments

envname	The name of, or path to, a Python virtual environment. If this name contains any slashes, the name will be interpreted as a path; if the name does not contain slashes, it will be treated as a virtual environment within virtualenv_root(). When NULL, the virtual environment as specified by the RETICULATE_PYTHON_ENV environment variable will be used instead.
python	The path to a Python interpreter, to be used with the created virtual environment. When NULL, the Python interpreter associated with the current session will be used.

[1] https://virtualenv.pypa.io/en/stable/

packages	A character vector with package names to install or remove.
ignore_installed	Boolean; ignore previously-installed versions of the requested packages? (This should normally be TRUE, so that pre-installed packages available in the site libraries are ignored and hence packages are installed into the virtual environment.)
...	Optional arguments; currently ignored for future expansion.
confirm	Boolean; confirm before removing packages or virtual environments?

Details

Virtual environments are by default located at ~/.virtualenvs (accessed with the virtualenv_root function). You can change the default location by defining defining the WORKON_HOME environment variable.

Virtual environment functions are not supported on Windows (the use of conda environments is recommended on Windows).

---

with.python.builtin.object

Evaluate an expression within a context.

Description

The with method for objects of type python.builtin.object implements the context manager protocol used by the Python with statement. The passed object must implement the context manager[2] (__enter__ and __exit__ methods.

Usage

```
## S3 method for class 'python.builtin.object'
with(data, expr, as = NULL, ...)
```

Arguments

data	Context to enter and exit
expr	Expression to evaluate within the context
as	Name of variable to assign context to for the duration of the expression's evaluation (optional).
...	Unused

[2] https://docs.python.org/2/reference/datamodel.html#context-managers