

Using Python in R with *reticulate* :: CHEAT SHEET



Getting Started

Reticulate makes it very easy to get started with Python in R. Simply use `py_require()` to let Reticulate know which Python version and packages you will need:

```
library(reticulate)
py_require("polars", python_version="3.12")
pl <- import("polars")
```

An isolated Python virtual environment that you will not need to manage is created, this eliminates the risk of the environment becoming unstable overtime.

U Reticulate uses an extremely fast Python package manager called **uv**. Some features of this system are::

- 10x–100x faster than **pip** at setting up virtual environments
- Downloads, installs and manages Python versions
- Supports macOS, Linux and **Windows**

Reticulate will automatically download and install **uv** for you in a location that does not make system changes to your machine.

Manage environments

Manage Virtualenv and Conda environments. Use if `py_require()` is not an option, or if using an existing Python virtual environment.

Create new	<code>virtualenv_create()</code> / <code>conda_create()</code> <code>virtualenv_create("my-env")</code>
Use in R session	<code>use_virtualenv()</code> / <code>use_condaenv()</code> <code>/ use_python()</code> <code>use_virtualenv("my-env")</code>
List available	<code>virtualenv_list()</code> / <code>conda_list()</code>
Install packages	<code>py_install()</code> / <code>virtualenv_install()</code> / <code>conda_install()</code> <code>py_install("polars", "my-env")</code>
Delete from disk	<code>virtualenv_remove()</code> / <code>conda_remove()</code> <code>virtualenv_remove("my-env")</code>

Calling Python

IMPORT PYTHON MODULES

Import any Python module into R, and access the attributes of a module with `$`.

import(module, as = NULL, convert = TRUE, delay_load = FALSE) - Import a Python module. If convert = TRUE, Python objects are converted to their equivalent R types. Access the attributes of a module with `$`.

import_from_path(module, path = ".") - Import module from an arbitrary filesystem path.

import_main(convert = TRUE) - Import the main module, where Python executes code by default.

import_builtins(convert = TRUE) - Import Python's built-in functions.

SOURCE PYTHON FILES

Source a Python script and make the Python functions and objects it creates available in R

source_python(file, envir = parent.frame(), convert = TRUE) - Run a Python script, assigning objects to a specified R environment.

`source_python("file.py")`

RUN PYTHON CODE

Execute Python code into the **main** Python module. Access the results, and anything else in Python's **main** module, with `py$`.

py_run_file(file, local = FALSE, convert = TRUE) - Run Python file in the main module.
`py_run_file("my-script.py")`

py_eval(code, convert = TRUE) - Run a Python expression, return the result.
`py_eval("1 + 1")`

py_run_string(code, local = FALSE, convert = TRUE) - Run Python code (passed as a string) in the main module.
`py_run_string("x = 10"); py$x`

IN A NOTEBOOK

Call Python as a code chunk in **Quarto** and **R Markdown**

Begin Python chunks with `{python}`
(Chunk options like `echo`, `include`, etc. all work as expected)

Use the `py` object to access objects created in Python chunks from R chunks.

Python chunks all execute within a **single** Python session so you have access to all objects created, and modules loaded, in previous chunks.



With `py_require()`, define which Python libraries will be used in the notebook

Use the `r` object to access objects created in R chunks from Python chunks

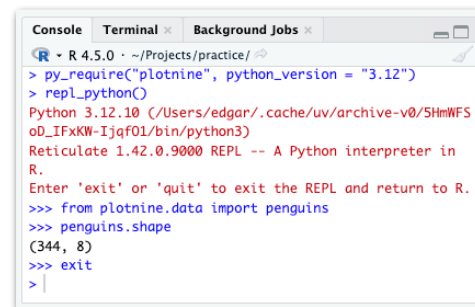
Output displays below chunk, including plots.

AS A PYTHON CONSOLE (REPL)

A **REPL** (Read, Eval, Print Loop) is a command line where you can run Python code and view the results.

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

1. Use `py_require()` to define libraries and Python version to use
2. Open in the console with `repl_python()`, or by running code in a Python script with **Cmd + Enter** (Windows: **Ctrl + Enter**).
3. Type commands at the `>>>` prompt.
4. Press **Enter** to run code.
5. Type **exit** to close and return to R console.



Using Python in R with *reticulate* :: CHEAT SHEET



Object Conversion

AUTOMATIC CONVERSIONS

Reticulate provides **automatic** built-in conversion between Python and R for many Python types.

py_to_r(x) Convert a Python object to an R object. Also **r_to_py()**.

R	Python
Single-element vector	Scalar
Multi-element vector	List
List of multiple types	Tuple
Named list	Dict
Matrix/Array	NumPy ndarray
Data Frame	Pandas DataFrame
Function	Python function
NULL, TRUE, FALSE	None, True, False

MANUAL CONVERSIONS

Specify how the objects will be converted

tuple(..., convert = FALSE) - Create a Python tuple.

`tuple("a", "b", "c")`

dict(..., convert = FALSE) - Create a Python dictionary.

`dict(foo = "bar", index = 42L)`

py_dict() - A dictionary that uses Python objects as keys.

`py_dict("foo", "bar")`

np_array(data, dtype = NULL, order = "C") - Create NumPy arrays.

`np_array(c(1:8), dtype = "float16")`

array_reshape(x, dim, order = c("C", "F")) - Reshape a Python array.

`x <- 1:4; array_reshape(x, c(2, 2))`

py_func(f) - Wrap an R function in a Python function with the same signature.

`py_func(xor)`

iterate(it, f = base::identity, simplify = TRUE) - Apply an R function to each value of a Python iterator or return the values as an R vector, draining the iterator as you go. Also **iter_next()** and **as_iterator()**.

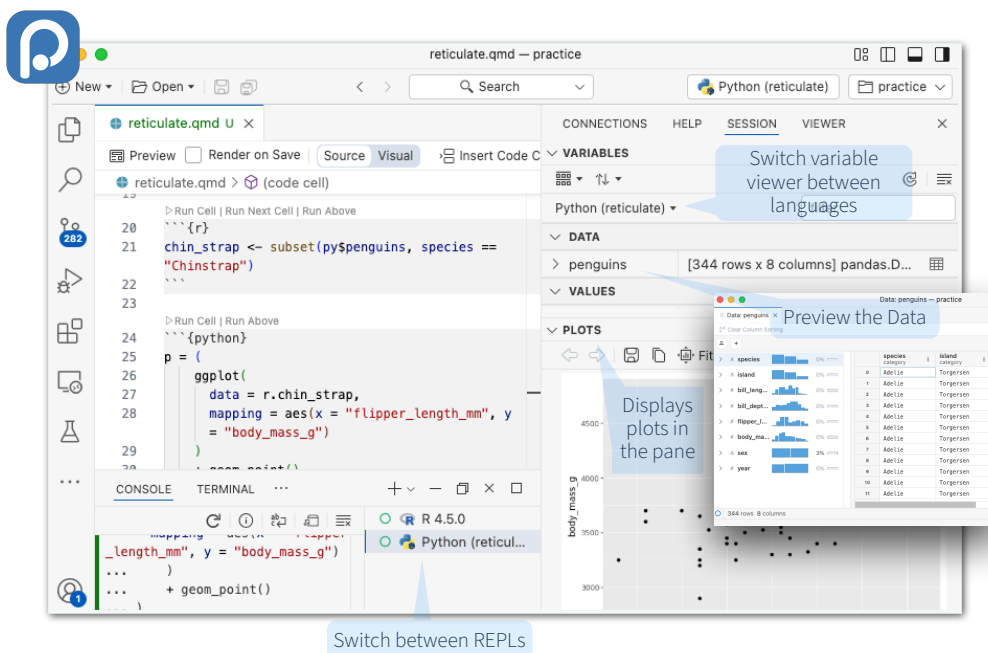
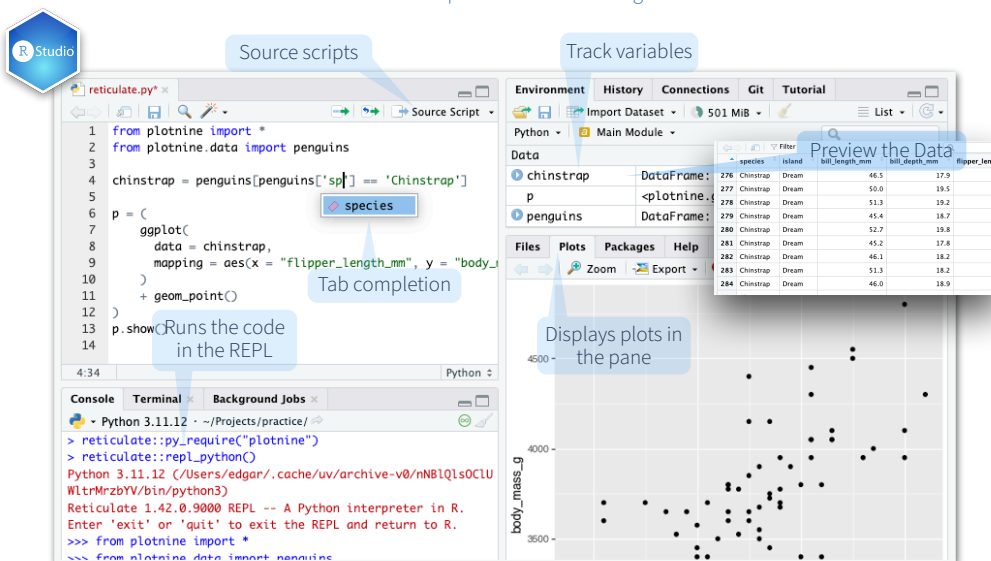
py_iterator(in, completed = NULL) - Create a Python iterator from an R function.

`seq_gen <- function(x){n <- x; function() {n <- n + 1; n}};`

`py_iterator(seq_gen(9))`

Python in the IDE

The RStudio and Positron IDE's provide first-class integration with Reticulate.



Helpers

py_capture_output(expr, type = c("stdout", "stderr")) Capture and return Python output. Also **py_suppress_warnings()**.

py_get_attr(x, name, silent = FALSE) Get an attribute of a Python object. Also **py_set_attr()**, **py_has_attr()**, and **py_list_attributes()**.

py_help(object) Open the documentation page for a Python object.

`py_help(sns)`

py_last_error() Get the last Python error encountered. Also **py_clear_last_error()** to clear the last error.

py_save_object(object, filename, pickle = "pickle", ...) Save and load Python objects with pickle. Also **py_load_object()**.

`py_save_object(x, "x.pickle")`

with(data, expr, as = NULL, ...) Evaluate an expression within a Python context manager.

`py <- import_builtins();`

`with(py$open("output.txt", "w") %as% file,`
`{ file$write("Hello, there!")})`

Choosing Python

Reticulate follows a specific order to discover and choose the Python environment to use

1. **RETICULATE_PYTHON** or **RETICULATE_PYTHON_ENV**
2. **use_python()** or **use_virtualenv()**, if called before **import()**.
3. Working directory contains a virtual env: **./venv**
4. Environments named after the imported module, e.g. `~/virtualenvs/r-scipy/` for `import("scipy")`
5. **RETICULATE_PYTHON_FALLBACK**
6. The default virtualenv: **r-reticulate**
7. Specifications from **py_require()** *

* To have **py_require()** take more precedence, set **RETICULATE_USE_MANAGED_VENV="yes"**. It will become number 3 on the list.

This is a partial list of the order of discovery, to see the full list visit:
rstudio.github.io/reticulate/articles/versions.html#order-of-discovery