

# Polyglot programming for single-cell analysis

Louise Deconinck Benjamin Rombaut Robrecht Cannoodt

2024-09-12

114 responses





# Introduction

1. How do you interact with a package in another language?
2. How do you make your package useable for developers in other languages?

We will be focusing on R & Python

## Summary

**Interoperability** between languages allows analysts to take advantage of the strengths of different ecosystems

**On-disk** interoperability uses standard file formats to transfer data and is typically more reliable

**In-memory** interoperability transfers data directly between parallel sessions and is convenient for interactive analysis

While interoperability is currently possible developers continue to improve the experience

[Single-cell best practices: Interoperability](#)

# How do you interact with a package in another language?

1. In-memory interoperability
2. Disk-based interoperability

# How do you make your package useable for developers in other languages?

1. Package-based interoperability
2. Best practices

# Package-based interoperability

or: the question of reimplementation.

Consider the pros:

1. Discoverability
2. Can your package be useful in other domains?
3. Very user friendly

Consider the cons:

1. Think twice: is it worth it?
2. **It's a lot of work**
3. How will you keep it up to date?
4. How will you ensure parity?



## **Package-based interoperability**

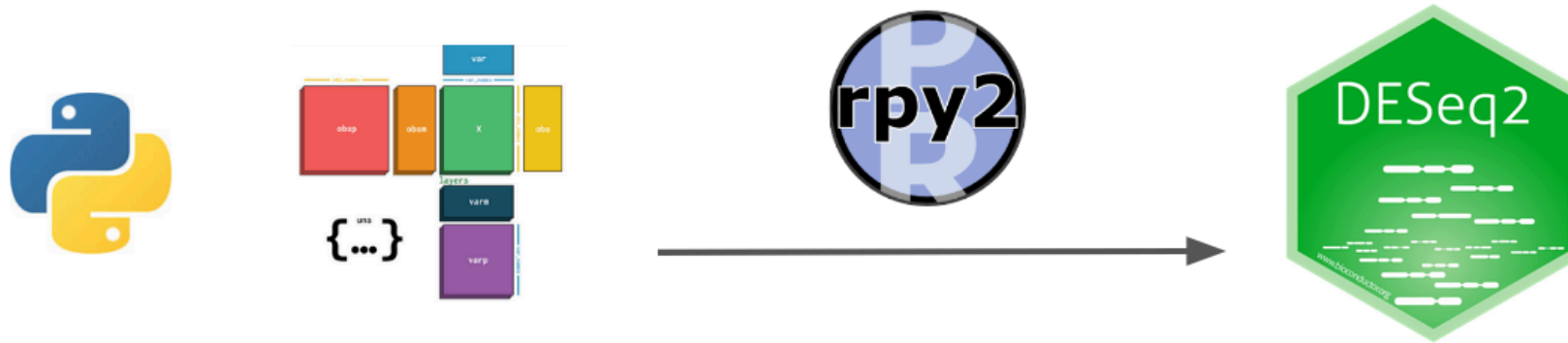
Please learn both R & Python

## Best practices

1. Work with the standards
2. Work with matrices, arrays and dataframes
3. Provide vignettes on interoperability

# In-memory interoperability

A Python user with an anndata object can use rpy2 to run the DESeq2 method in R



An R user with an anndata object can use reticulate to run scanpy functions in Python



# Overview

1. Advantages & disadvantages
2. Pitfalls when using Python & R
3. Rpy2
4. Reticulate

## in-memory interoperability advantages and disadvantages

### advantages

- no need to write & read results
- useful when you need a limited amount of functions in another language

### disadvantages

- not always access to all classes
- data duplication
- you need to manage the environments

# Pitfalls when using Python and R

# Column major vs row major matrices

In R: every dense matrix is stored as column major

one matrix

|    |    |    |
|----|----|----|
| 1  | 2  | 3  |
| 4  | 5  | 6  |
| 7  | 8  | 9  |
| 10 | 11 | 12 |

|    |    |    |
|----|----|----|
| 1  | 2  | 3  |
| 4  | 5  | 6  |
| 7  | 8  | 9  |
| 10 | 11 | 12 |

in-memory: row major

|   |   |   |   |   |   |   |   |   |    |    |    |
|---|---|---|---|---|---|---|---|---|----|----|----|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|----|----|----|

in-memory: column major

|   |   |   |    |   |   |   |    |   |   |   |    |
|---|---|---|----|---|---|---|----|---|---|---|----|
| 1 | 4 | 7 | 10 | 2 | 5 | 8 | 11 | 3 | 6 | 9 | 12 |
|---|---|---|----|---|---|---|----|---|---|---|----|

# Indexing

index (Python)

0

1

2

3

4

5



index (R)

1

2

3

4

5

6



# Dots and underscores

mapping in rpy2

```
1 from rpy2.robjects.packages import importr
2
3 d = {'package.dependencies': 'package_dot_dependencies',
4      'package_dependencies': 'package_uscore_dependencies'}
5 tools = importr('tools', robject_translations = d)
```

# Integers

```
1 library(reticulate)
2 bi <- reticulate::import_builtins()
3
4 bi$list(bi$range(0, 5))
5 # TypeError: 'float' object cannot be interpreted as an integer
```

```
1 library(reticulate)
2 bi <- reticulate::import_builtins()
3
4 bi$list(bi$range(0L, 5L))
5 # [1] 0 1 2 3 4
```

# Rpy2

## Accessing R from Python

- `rpy2.rinterface`, the low-level interface
- `rpy2.robj`, the high-level interface

```
1 import rpy2
2 import rpy2.robj as robj
3
4 vector = robj.IntVector([1,2,3])
5 rsum = robj.r['sum']
6
7 rsum(vector)
```

IntVector with 1 elements.

6

# Rpy2: basics

```
1 str_vector = robjects.StrVector(['abc', 'def', 'ghi'])
2 flt_vector = robjects.FloatVector([0.3, 0.8, 0.7])
3 int_vector = robjects.IntVector([1, 2, 3])
4 mtx = robjects.r.matrix(robjecs.IntVector(range(10)), nrow=5)
5 print(mtx)
```

```
      [,1] [,2]
[1,]     0     5
[2,]     1     6
[3,]     2     7
[4,]     3     8
[5,]     4     9
```

# Rpy2: numpy

```
1 import numpy as np
2
3 from rpy2.robjects import numpy2ri
4 from rpy2.robjects import default_converter
5
6 rd_m = np.random.random((5, 4))
7
8 with (default_converter + numpy2ri.converter).context():
9     mtx = robjects.r.matrix(rd_m, nrow = 5)
10    print(mtx)
```

```
[[0.31332317 0.35364146 0.56282682 0.85862223]
 [0.62741044 0.572032   0.69183808 0.7092725 ]
 [0.3976838  0.34202251 0.23438556 0.59881822]
 [0.78234652 0.99630959 0.18262528 0.83434156]
 [0.54189244 0.31409299 0.85954986 0.70672631]]
```

# Rpy2: pandas

```
1 import pandas as pd
2
3 from rpy2.robjects import pandas2ri
4
5 pd_df = pd.DataFrame({'int_values': [1,2,3],
6                           'str_values': ['abc', 'def', 'ghi']})
7
8 with (default_converter + pandas2ri.converter).context():
9     pd_df_r = robjects.DataFrame(pd_df)
10    print(pd_df_r)
```

|   | int_values | str_values |
|---|------------|------------|
| 0 | 1          | abc        |
| 1 | 2          | def        |
| 2 | 3          | ghi        |

# Rpy2: sparse matrices

```
1 import scipy as sp
2
3 from anndata2ri import scipy2ri
4
5 sparse_matrix = sp.sparse.csc_matrix(rd_m)
6
7 with (default_converter + scipy2ri.converter).context():
8     sp_r = scipy2ri.py2rpy(sparse_matrix)
9     print(sp_r)
```

5 x 4 sparse Matrix of class "dgCMatrix"

```
[1,] 0.3133232 0.3536415 0.5628268 0.8586222
[2,] 0.6274104 0.5720320 0.6918381 0.7092725
[3,] 0.3976838 0.3420225 0.2343856 0.5988182
[4,] 0.7823465 0.9963096 0.1826253 0.8343416
[5,] 0.5418924 0.3140930 0.8595499 0.7067263
```

# Rpy2: anndata

```
1 import anndata as ad
2 import scanpy.datasets as scd
3
4 import anndata2ri
5
6 adata_paul = scd.paul15()
7
8 with anndata2ri.converter.context():
9     sce = anndata2ri.py2rpy(adata_paul)
10     ad2 = anndata2ri.rpy2py(sce)
```



# Rpy2: interactivity

```
1 %load_ext rpy2.ipython # line magic that loads the rpy2 ipython extension.
2                         # this extension allows the use of the following cell magic
3
4 %%R -i input -o output # this line allows to specify inputs
5                         # (which will be converted to R objects) and outputs
6                         # (which will be converted back to Python objects)
7                         # this line is put at the start of a cell
8                         # the rest of the cell will be run as R code
```

# Reticulate

| R                      | Python            | Examples  |
|------------------------|-------------------|---|
| Single-element vector  | Scalar            | <code>1, 1L, TRUE, "foo"</code>                             |
| Multi-element vector   | List              | <code>c(1.0, 2.0, 3.0), c(1L, 2L, 3L)</code>                |
| List of multiple types | Tuple             | <code>list(1L, TRUE, "foo")</code>                          |
| Named list             | Dict              | <code>list(a = 1L, b = 2.0), dict(x = x_data)</code>        |
| Matrix/Array           | NumPy ndarray     | <code>matrix(c(1,2,3,4), nrow = 2, ncol = 2)</code>         |
| Data Frame             | Pandas DataFrame  | <code>data.frame(x = c(1,2,3), y = c("a", "b", "c"))</code> |
| Function               | Python function   | <code>function(x) x + 1</code>                              |
| Raw                    | Python bytearray  | <code>as.raw(c(1:10))</code>                                |
| NULL, TRUE, FALSE      | None, True, False | <code>NULL, TRUE, FALSE</code>                              |

# Reticulate: basics

```
1 library(reticulate)
2
3 bi <- reticulate::import_builtins()
4 rd <- reticulate::import("random")
5
6 example <- c(1,2,3)
7 bi$max(example)
8 # [1] 3
9 rd$choice(example)
10 # [1] 2
11 cat(bi$list(bi$reversed(example)))
12 # [1] 3 2 1
```

# Reticulate numpy

```
1 np <- reticulate::import("numpy")
2
3 a <- np$asarray(tuple(list(1,2), list(3, 4)))
4 b <- np$asarray(list(5,6))
5 b <- np$reshape(b, newshape = tuple(1L,2L))
6
7 np$concatenate(tuple(a, b), axis=0L)
8 #      [,1] [,2]
9 # [1,]    1    2
10 # [2,]    3    4
11 # [3,]    5    6
```

# Reticulate conversion

```
1 np <- reticulate::import("numpy", convert = FALSE)
2
3 a <- np$asarray(tuple(list(1,2), list(3, 4)))
4 b <- np$asarray(list(5,6))
5 b <- np$reshape(b, newshape = tuple(1L,2L))
6
7 np$concatenate(tuple(a, b), axis=0L)
8 # array([[1., 2.],
9 #        [3., 4.],
10 #        [5., 6.]])
```

You can explicitly convert data types:

```
1 result <- np$concatenate(tuple(a, b), axis=0L)
2
3 py_to_r(result)
4 #      [,1] [,2]
5 # [1,]    1    2
6 # [2,]    3    4
7 # [3,]    5    6
8
9 result_r <- py_to_r(result)
10 r_to_py(result_r)
11 # array([[1., 2.],
12 #        [3., 4.],
13 #        [5., 6.]])
```

# Reticulate scanpy

```
1 library(anndata)
2 library(reticulate)
3 sc <- import("scanpy")
4
5 adata_path <- "../usecase/data/sc_counts_subset.h5ad"
6 adata <- anndata::read_h5ad(adata_path)
```

We can preprocess & analyse the data:

```
1 sc$pp$filter_cells(adata, min_genes = 200)
2 sc$pp$filter_genes(adata, min_cells = 3)
3 sc$pp$pca(adata)
4 sc$pp$neighbors(adata)
5 sc$tl$umap(adata)
6
7 adata
8 # AnnData object with n_obs × n_vars = 32727 × 20542
9 #   obs: 'dose_uM', 'timepoint_hr', 'well', 'row', 'col', 'plate_name', 'cell_id', 'cell_type', 'split', 'c
10 #   var: 'highly_variable', 'means', 'dispersions', 'dispersions_norm', 'n_cells'
11 #   uns: 'cell_type_colors', 'celltypist_celltype_colors', 'donor_id_colors', 'hvg', 'leiden_res1_colors',
12 #   obsm: 'HTO_clr', 'X_pca', 'X_umap', 'protein_counts'
13 #   varm: 'PCs'
14 #   obsp: 'connectivities', 'distances'
```



## Disk-based interoperability

Disk-based interoperability is a strategy for achieving interoperability between tools written in different programming languages by **storing intermediate results in standardized, language-agnostic file formats**.

Upside:

- Simple, just add reading and writing lines
- Modular scripts

Downside:

- increased disk usage
- less direct interaction, debugging...



# Important features of interoperable file formats

Compression

Sparse matrix support

Large images























































Lazy chunk loading

Remote storage

## General single cell file formats of interest for Python and R

| File Format | Python                           | R                                | Sparse matrix                    | Large images                     | Lazy chunk loading               | Remote storage                   |
|-------------|----------------------------------|----------------------------------|----------------------------------|----------------------------------|----------------------------------|----------------------------------|
| RDS         | <input type="radio"/>            | <input checked="" type="radio"/> | <input type="radio"/>            | <input checked="" type="radio"/> | <input type="radio"/>            | <input type="radio"/>            |
| Pickle      | <input checked="" type="radio"/> | <input type="radio"/>            | <input type="radio"/>            | <input checked="" type="radio"/> | <input type="radio"/>            | <input type="radio"/>            |
| CSV         | <input checked="" type="radio"/> | <input checked="" type="radio"/> | <input type="radio"/>            | <input type="radio"/>            | <input type="radio"/>            | <input type="radio"/>            |
| JSON        | <input checked="" type="radio"/> | <input checked="" type="radio"/> | <input type="radio"/>            | <input type="radio"/>            | <input type="radio"/>            | <input type="radio"/>            |
| TIFF        | <input checked="" type="radio"/> | <input checked="" type="radio"/> | <input type="radio"/>            | <input checked="" type="radio"/> | <input checked="" type="radio"/> | <input checked="" type="radio"/> |
| .npy        | <input checked="" type="radio"/> | <input type="radio"/>            | <input type="radio"/>            | <input checked="" type="radio"/> | <input type="radio"/>            | <input type="radio"/>            |
| Parquet     | <input checked="" type="radio"/> | <input checked="" type="radio"/> | <input type="radio"/>            | <input type="radio"/>            | <input checked="" type="radio"/> | <input checked="" type="radio"/> |
| Feather     | <input checked="" type="radio"/> | <input checked="" type="radio"/> | <input checked="" type="radio"/> | <input type="radio"/>            | <input checked="" type="radio"/> | <input checked="" type="radio"/> |
| Lance       | <input checked="" type="radio"/> | <input type="radio"/>            | <input checked="" type="radio"/> | <input type="radio"/>            | <input checked="" type="radio"/> | <input checked="" type="radio"/> |
| HDF5        | <input checked="" type="radio"/> | <input checked="" type="radio"/> | <input type="radio"/>            | <input checked="" type="radio"/> | <input checked="" type="radio"/> | <input checked="" type="radio"/> |
| Zarr        | <input checked="" type="radio"/> | <input checked="" type="radio"/> | <input type="radio"/>            | <input checked="" type="radio"/> | <input checked="" type="radio"/> | <input checked="" type="radio"/> |
| TileDB      | <input checked="" type="radio"/> | <input checked="" type="radio"/> | <input checked="" type="radio"/> | <input checked="" type="radio"/> | <input checked="" type="radio"/> | <input checked="" type="radio"/> |

## Specialized single cell file formats of interest for Python and R

| File Format       | Python  | R   | Sparse matrix   | Large images  | Lazy chunk loading  | Remote storage  |
|-------------------|---|---|---|---|---|---|
| Seurat RDS        |    |    |    |    |    |    |
| Indexed OME-TIFF  |    |    |    |    |    |    |
| h5Seurat          |    |    |    |    |    |    |
| Loom HDF5         |    |    |    |    |    |    |
| AnnData h5ad      |    |    |    |    |    |    |
| AnnData Zarr      |    |    |    |    |    |    |
| TileDB-SOMA       |   |   |   |   |   |   |
| TileDB-BioImaging |  |  |  |  |  |  |
| SpatialData Zarr  |  |  |  |  |  |  |

# Disk-based pipelines

## Script pipeline:

```
1 #!/bin/bash
2
3 bash scripts/1_load_data.sh
4 python scripts/2_compute_pseudobulk.py
5 Rscript scripts/3_analysis_de.R
```

## Notebook pipeline:

```
1 # Every step can be a new notebook execution with inspectable output
2 jupyter nbconvert --to notebook --execute my_notebook.ipynb --allow-errors --output-dir outputs/
```

## Just stay in your language and call scripts

```
1 import subprocess
2
3 subprocess.run("bash scripts/1_load_data.sh", shell=True)
4 # Alternatively you can run Python code here instead of calling a Python script
5 subprocess.run("python scripts/2_compute_pseudobulk.py", shell=True)
6 subprocess.run("Rscript scripts/3_analysis_de.R", shell=True)
```

## Pipelines with different environments

1. interleave with environment (de)activation functions
2. use rvenv
3. use Pixi

# Pixi to manage different environments

```
1 pixi run -e bash scripts/1_load_data.sh  
2 pixi run -e scverse scripts/2_compute_pseudobulk.py  
3 pixi run -e rverse scripts/3_analysis_de.R
```

## Define tasks in Pixi

```
1 ...
2 [feature.bash.tasks]
3 load_data = "bash book/disk_based/scripts/1_load_data.sh"
4 ...
5 [feature.scverse.tasks]
6 compute_pseudobulk = "python book/disk_based/scripts/2_compute_pseudobulk.py"
7 ...
8 [feature.rverse.tasks]
9 analysis_de = "Rscript --no-init-file book/disk_based/scripts/3_analysis_de.R"
10 ...
11 [tasks]
12 pipeline = { depends-on = ["load_data", "compute_pseudobulk", "analysis_de"] }
```

```
1 pixi run pipeline
```



## Also possible to use containers

```
1 docker pull berombau/polygloty-docker:latest
2 docker run -it -v $(pwd)/usecase:/app/usecase -v $(pwd)/book:/app/book berombau/polygloty-docker:latest pixi
```

Another approach is to use multi-package containers to create custom combinations of packages. - [Multi-Package BioContainers](#) - [Seqera Containers](#)

## Workflows

You can go a long way with a folder of notebooks or scripts and the right tools. But as your project grows more bespoke, it can be worth the effort to use a **workflow framework** like Viash, Nextflow or Snakemake to manage the pipeline for you.

See [https://saeyslab.github.io/polygloty/book/workflow\\_frameworks/](https://saeyslab.github.io/polygloty/book/workflow_frameworks/)

# Takeaways