

ROOT as a Python module

Enric Tejedor for the ROOT team

ROOT Users Workshop 2022
Fermilab, CERN

ROOT
Data Analysis Framework
<https://root.cern>

Introduction

ROOT

Data Analysis Framework

<https://root.cern>



- ▶ ROOT can be used as a Python module via PyROOT
 - Python-C++ bindings
 - `import ROOT`
- ▶ Access all the ROOT C++ functionality from Python
 - And also user-defined C++ code!
- ▶ Automatic, dynamic
 - No static wrapper generation
 - Dynamic python proxies for C++ entities
 - Lazy class/variable lookup
- ▶ Powered by the ROOT type system and Cling
 - Reflection information, JIT C++ compilation, execution



The New PyROOT

- ▶ Available since v6.22
- ▶ Based on Cppyy
 - Set of packages for automatic Python-C++ binding generation
 - Forked from old PyROOT by Wim Lavrijsen
- ▶ Goal: benefit from all the new features of Cppyy
 - Modern C++ support
- ▶ ROOT-specific *pythonizations* added on top



A Concrete Example

► Automatic bindings + Pythonizations

```
import ROOT
f = ROOT.TFile('myfile.root')
t = f.mytree
# vs f.GetObject('mytree')
```

TFFile is a (dynamic) Python proxy of a C++ class

f is a (dynamic) Python proxy of a C++ object

Pythonization: access tree as an attribute

ROOT Pythonizations

ROOT
Data Analysis Framework
<https://root.cern>



▶ **Pythonization:**

- Code that modifies the behaviour of a C++ class when used from Python
 - Makes it easier / more “pythonic” to use that class
- ▶ A number of pythonizations are already available (*docs* in the reference guide!):
- [TFile](#), [TTree](#), [RDataFrame](#), [RVec](#), ...
 - [RooFit](#) → see [Jonas' presentation](#) on Wednesday



RDataFrame to NumPy (and pandas)

- ▶ Process dataset with RDataFrame, convert columns to NumPy arrays

```
from ROOT import RDataFrame
df = RDataFrame('myTree', 'file.root')

# Apply cuts, define new columns
df = df.Filter('x > 0').Define('z', 'x*y')

cols = df.AsNumpy() # cols is a dict of NumPy arrays

import pandas
pd_df = pandas.DataFrame(cols) # convert to pandas DF
```

[Tutorial here](#)

Since v6.18



Python Callables in RDF Operations

- ▶ New decorator to just-in-time compile Python callables
 - Can be used e.g. in `Filter`, `Define`

```
df = RDataFrame('myTree', 'file.root')
```

[Tutorial here](#)

```
df.Filter('x > 0') # jitted C++ expression
```

Since v6.22

```
@ROOT.Numba.Declare(['int'], 'bool')
```

```
def my_cut(x):
```

```
    return x > 0
```

```
df.Filter('Numba::my_cut(x)') # jitted Python callable
```



Python Callables in RDF Operations (II)

- ▶ Coming soon:
 - No decorator needed
 - Jitting done under the hood

```
df = RDataFrame('myTree', 'file.root')

def my_cut(x):
    return x > 0

df.Filter(my_cut, ['x']) # jits Python callable
```



TFile Context Manager

- ▶ Use TFile instances in a with block à la Python file
 - The TFile is closed no matter what happens in the block
 - You can read or write into the TFile
 - Objects owned by the TFile won't survive the end of the block!

```
with TFile.Open('file.root', 'recreate') as f:  
    h = TH1F('my_histo', 'My histo', 10, 0, 10)  
    ...  
    f.WriteObject(h, 'my_histo')
```

```
# h is None here!
```

[Tutorial here](#)

Will be in v6.28

User Pythonizations

ROOT

Data Analysis Framework

<https://root.cern>



Define Your Own Pythonizations

- ▶ New decorator to create pythonizations for user C++ classes
 - User defines a function that performs the pythonization
 - The function is decorated with `@pythonization` → **registers** pythonization
 - The function is executed **lazily**, only if the class is used

```
@pythonization('MyCppClass')  ← Python proxy of the C++ class
def my_pythonizer_function(klass):
    # Inject new behaviour in the class
    klass.__str__ = lambda o : 'A MyCppClass object'

obj = ROOT.MyCppClass() # triggers pythonization

print(obj) # prints 'A MyCppClass object'
```

Since v6.26



How To Use @pythonization

- ▶ Pythonize classes in namespaces (default is global)

```
@pythonization('vector<int>', ns='std')  
def my_pythonizor_function(klass):  
    ...
```

- ▶ Pythonize all classes with a certain prefix

```
@pythonization('vector<', ns='std', is_prefix=True)  
def my_pythonizor_function(klass):  
    ...
```



How To Use @pythonization (II)

- ▶ Decorators for multiple classes can be stacked

```
@pythonization('A')  
@pythonization('B')  
def my_pythonizer_function(klass):  
    ...
```

- ▶ To learn more:
 - See [this tutorial](#)
 - Check how the decorator is applied to ROOT classes for real pythonizations on [ROOT's github repository](#)

A bit on building PyROOT

ROOT

Data Analysis Framework

<https://root.cern>



Since v6.22

- ▶ Single ROOT build for Python 2 & 3
 - Eases the transition from Python 2 to 3
 - Support Python 2 for as long as the experiments need it
- ▶ How it works
 - By default, it will build for both if it finds them in the system
 - Hints can be provided to point to specific Python installations
 - Building for a single Python version is still possible
 - More info on the [ROOT website](#)

Summary

ROOT

Data Analysis Framework

<https://root.cern>



- ▶ New PyROOT available since v6.22
 - Access to modern C++ thanks to cppy
- ▶ Pythonizations
 - Make it easier to use C++ entities from Python
 - Many ROOT pythonizations already available
 - User-defined pythonizations are also possible with the `@pythonization` decorator

Backup Slides

ROOT

Data Analysis Framework

<https://root.cern>



The New Structure

PyROOT

User API

ROOT Pythonizations

Cppyy

Automatic Bindings:
Proxy Creation,
Type Conversion
(Python/C API)

STL
Pythonizations

ROOT & Cling

Reflection Info,
Execution

ROOT Type System
(TClass, TMethod, ...)



New PyROOT: Variadic Templates

- ▶ Support for variadic template arguments of functions

```
>>> import ROOT
>>> ROOT.gInterpreter.ProcessLine("""
template<typename... myTypes>
int f() { return sizeof...(myTypes); }
""")
0L
>>> ROOT.f['int', 'double', 'void*']()
3
```




New PyROOT: Lambdas

- ▶ Possible to use C++ lambdas from Python

```
>>> import ROOT
>>> ROOT.gInterpreter.ProcessLine(
"auto mylambda = [](int i) { std::cout << i << std::endl; };")
140518947094560L
>>> ROOT.mylambda
<cppyy.gbl.function<void(int)>* object at 0x35f9570>
>>> ROOT.mylambda(2)
2
```