Connecting ROOT to the Python world with Numpy arrays

2018-03-08

What is the idea?

- Numpy arrays are the interface for all of the scientific libraries in the Python world (scipy, sklearn, tensorflow, matplotlib, . . .).
- The desired interface would look like this:

```
>>> import ROOT
>>> import numpy as np
>>> x = ROOT.TSomeObjectWithContiguousData()
>>> y = np.asarray(x) # <- Zero-copy operation!
>>> print(y.shape)
(num_dim_1, num_dim_2, ...)
```

There are two solutions to make this possible →

Reminder: Memory-layout of Numpy arrays

Documentation: Link

> *An instance of class ndarray consists of a contiguous one-dimensional segment of computer memory (owned by the array, or by some other object), combined with an indexing scheme that maps N integers into the location of an item in the block. The ranges in which the indices can vary is specified by the shape of the array.*

Short-term solution: The (Numpy) array interface

- Adding the `__array_interface__` magic to ROOT Python objects (Documentation)

```
...
>>> x = ROOT.TSomeObjectWithContiguousData()
>>> print(x.__array_interface__) # This is a dictionary!
{
    "version": 3, # Version of the array interface
    "shape": (100, 4), # Shape information
    "typestr": "<f4", # 4-byte float, little endian
    "data": [12345678, False], # Pointer to first element, read-only flag
    ... # There are more optional fields to support C-style structs, offsets, masks, strides, ...
}
>>> y = np.asarray(x) # Zero-copy operation, adopts the memory
>>> print(y.shape)
(100, 4)
```

- This can happen in the Pythonization-layer of PyROOT.
- Fast and cheap solution.

Long-term solution: The buffer protocol

Description found here:

> *Certain objects available in Python wrap access to an underlying memory array or buffer. Such objects include the built-in `bytes` and `bytearray`, and some extension types like `array.array`. Third-party libraries may define their own types for special purposes, such as image processing or numeric analysis.*

Basic structure, defined in the module source:

```
typedef struct bufferinfo {
    void *buf;
    PyObject *obj;
    Py_ssize_t len;
    Py_ssize_t itemsize;
    int readonly;
    int ndim;
    char *format;
    Py_ssize_t *shape;
    Py_ssize_t *strides;
    Py_ssize_t *suboffsets;
    void *internal;
} Py_buffer;

int PyObject_GetBuffer(PyObject *obj, Py_buffer *view, int flags);
```

Numpy (and others) understand the buffer protocol:

```
...
>>> x = ROOT.TSomeObjectWithContiguousData() # Python object implements the buffer protocol
>>> y = np.asarray(x) # Zero-copy operation
>>> print(y.shape)
(num_dim_1, num_dim_2, ...)
```