Improving reflection layer in cppyy using Cling
Introduction

cppyy: Generates Python C++ binding at runtime, automatically

Cling: interactive C++ interpreter

ROOT meta: A layer in ROOT that provides reflection

```
>>> a = 10
>>> type(a)  # Reflection in Python
<int>
```

```python
import cppyy
s = cppyy.gbl.std.string("Hello World!!")
```
Problem Statement

Convert cppyy-backend to use Cling directly instead of ROOT meta and use it in cppyy

Why?: ROOT meta adds unnecessary code bloat and the performance of cppyy can be improved using Cling

Ultimately we want a cppyy-style python language interop but the ROOT meta dependency of cppyy is unnecessary. So we aim to make a libInterop library without such dependency and add it to llvm mainline
Current Approach

1) Look at CPyCppyy top level functions and list the requirements. E.g. Functions like `CreateScopeProxy` need to lookup C++ classes, namespaces, etc. so we need to have an API that can lookup these easily.

Current implementation:

```cpp
PyObject* CPyCppyy::CreateScopeProxy(const std::string& name, PyObject* parent, const unsigned flags) {
    // search for the scope using ROOT meta
    //
    // if its a namespace return a proxy
    // without any further details
    //
    // if its a class return a proxy with all
    // details of the class:
    //    base classes, data members, functions
    //    etc.
}
```

Intended implementation:

```cpp
PyObject* CPyCppyy::CreateScopeProxy(const std::string& name, PyObject* parent, const unsigned flags) {
    // lookup the name through Cling
    //
    // if the name is a class or namespace return
    // a proxy with the name (without any
    // internal details)
}
```
Current Approach

2) Optimize wherever possible. E.g.:

a) CPyCppyy creates snapshots of classes because it depends on ROOT meta, Cling lookups are O(1) so lazy lookups are better.

```
PyObject* CPyCppyy::CreateScopeProxy (const std::string& name, PyObject* parent, const unsigned flags) {
    // search for the scope using ROOT meta
    //
    // if its a namespace return a proxy
    // without any further details
    //
    // if its a class return a proxy with all
    // details of the class:
    //   base classes, data members, functions
    //   etc.
}
```

Current implementation:

```
PyObject* CPyCppyy::CreateScopeProxy (const std::string& name, PyObject* parent, const unsigned flags) {
    // lookup the name through Cling
    //
    // if the name is a class or namespace return
    // a proxy with the name (without any
    // internal details)
}
```

Intended implementation:
Current Approach

2) Optimize wherever possible. E.g.:

   a) CPyCppyy creates snapshots of classes because it depends on ROOT meta, Cling lookups are $O(1)$ so lazy lookups are better.

   b) cppyy-backend is made with iteration in mind, i.e., a call is first made to get the number of items in a scope and then the user can access these items using indexes. Cling uses a direct access approach and the user can look for items using their name.

Using ROOT meta:

```cpp
const Cppy::TCppIndex_t nDataMembers = Cppy::GetNumDatamembers(scope);
for (Cppy::TCppIndex_t idata = 0; idata < nDataMembers; ++idata) {
    ...
}
```

Using Cling:

```cpp
Decl* D = cling::utils::Lookup::Named(Sema, name);
```
Current Approach

3) Replace the functions in cppyy-backend with their Cling counterparts. These can vary in functionality to a great extent.

For e.g.: functions such as `GetNumDatamembers` might not be required in the cppyy-backend as we will no longer be iterating over the data members of a class.
Extended Approach

An extended approach involves emulating the current functions in cppyy-backend using Cling. This will be required if the library for D and C++ interoperability requires the API to support iterations.

⚠ Emulating the current API might be detrimental to performance

```cpp
class C {
    int a;
    bool f();
    enum Time {
        morning,
        evening
    };
};
```

```cpp
size_t GetNumDatamembers(std::string name) {
    // lookup name through Cling
    //
    // get the DeclContext DC of the class
    //
    // for each Decl in DC check if its a
    // member variable or enum and increase
    // counter
    //
    // return counter
}
```
Goals

● The goal for this month is to convert the functions in ProxyWrappers.cxx to use Cling. This will allow cppyy to run simple examples properly.

● A stretch goal will be to shift clingwrapper.cxx to use CMake and statically link with Cling
Blockers

Input from the D representative is needed to figure out how much of the cppy-backend API can be modified.
Thank you

Any questions?