A Jupyter kernel for C++ based on the `cling` C++ interpreter and the `xeus` native implementation of the Jupyter protocol, xeus.

- GitHub repository: https://github.com/QuantStack/xeus-cling/
- Online documentation: https://xeus-cling.readthedocs.io/

Output and error streams

`std::cout` and `std::cerr` are redirected to the notebook frontend.

In [ ]:
```
#include <iostream>

std::cout << "some output" << std::endl;
```

In [ ]:
```
std::cerr << "some error" << std::endl;
```

In [ ]:
```
#include <stdexcept>
```

In [ ]:
```
try {
    throw std::runtime_error("Unknown exception");
} catch (const std::runtime_error& e) {
    std::cerr << "caught error: " << e.what() << std::endl;
}
```

Omitting the `;` in the last statement of a cell results in an output being printed

In [ ]:
```
int j = 5;
```

In [ ]:
```
j
```

Interpreting the C++ programming language

`cling` has a broad support of the features of C++. You can define functions, classes, templates, etc ...

Functions

In [ ]:
```
double sqr(double a)
{
    return a * a;
}
```

In [ ]:
```
double a = 2.5;
double asqr = sqr(a);
asqr
```

Classes

In [ ]:
```
class Foo
{
public:

    virtual ~Foo() {}

    virtual void print(double value) const
    {
        std::cout << "Foo value = " << value << std::endl;
    }
};
```

In [ ]:
```
Foo bar;
bar.print(1.2);
```

Polymorphism

In [ ]:
```
class Bar : public Foo
{
public:

    virtual ~Bar() {}

    virtual void print(double value) const
    {
        std::cout << "Bar value = " << 2 * value << std::endl;
    }
};
```

In [ ]:
```
Foo* bar2 = new Bar;
bar2->print(1.2);
delete bar2;
```

Templates

In [ ]:
```

```

```
#include <typeinfo>

template <class T>
class FooT
{
public:

    explicit FooT(const T& t) : m_t(t) {}

    void print() const
    {
        std::cout << typeid(T).name() << " m_t = " << m_t << std::endl;
    }

private:

    T m_t;
};

template <>
class FooT<int>
{
public:

    explicit FooT(const int& t) : m_t(t) {}

    void print() const
    {
        std::cout << "m_t = " << m_t << std::endl;
    }

private:

    int m_t;
};
```

In [ ]:
```
FooT<double> foot1(1.2);
foot1.print();
```

In [ ]:
```
FooT<int> foot2(4);
foot2.print();
```

C++11 / C++14 support

In [ ]:
```
class Foo11
{
public:

    Foo11() { std::cout << "Foo11 default constructor" << std::endl; }
    Foo11(const Foo11&) { std::cout << "Foo11 copy constructor" << std::endl; }
    Foo11(Foo11&&) { std::cout << "Foo11 move constructor" << std::endl; }
};
```

In [ ]:
```
Foo11 f1;
Foo11 f2(f1);
Foo11 f3(std::move(f1));
```

In [ ]:
```
#include <vector>

std::vector<int> v = { 1, 2, 3};
auto iter = ++v.begin();
v
```

In [ ]:
```
*iter
```

... and also lambda, universal references, `decltype`, etc ...

Documentation and completion

- Documentation for types of the standard library is retrieved on cppreference.com.
- The quick-help feature can also be enabled for user-defined types and third-party libraries. More documentation on this feature is available at https://xeus-cling.readthedocs.io/en/latest/inline_help.html.

In [ ]:
```
?std::vector
```

Using the `display_data` mechanism

For a user-defined type `T`, the rich rendering in the notebook and JupyterLab can be enabled by by implementing the function `xeus::xjson mime_bundle_repr(const T& im)`, which returns the JSON mime bundle for that type.

More documentation on the rich display system of Jupyter and Xeus-cling is available at https://xeus-cling.readthedocs.io/en/latest/rich_display.html

Image example

In [ ]:
```
#include <string>
#include <fstream>

#include "xtl/xbase64.hpp"
#include "xeus/xjson.hpp"

namespace im
{
    struct image
    {
        inline image(const std::string& filename)
        {
            std::ifstream fin(filename, std::ios::binary);
            m_buffer << fin.rdbuf();
        }

        std::stringstream m_buffer;
    };
```

```
    xeus::xjson mime_bundle_repr(const image& i)
    {
        auto bundle = xeus::xjson::object();
        bundle["image/png"] = xtl::base64encode(i.m_buffer.str());
        return bundle;
    }
}
```

In [ ]:
```
im::image marie("images/marie.png");
marie
```

Audio example

In [ ]:
```
#include <string>
#include <fstream>

#include "xtl/xbase64.hpp"
#include "xeus/xjson.hpp"

namespace au
{
    struct audio
    {
        inline audio(const std::string& filename)
        {
            std::ifstream fin(filename, std::ios::binary);
            m_buffer << fin.rdbuf();
        }

        std::stringstream m_buffer;
    };

    xeus::xjson mime_bundle_repr(const audio& a)
    {
        auto bundle = xeus::xjson::object();
        bundle["text/html"] =
           std::string("<audio controls=\"controls\"><source src=\"data:audio/
wav;base64,")
           + xtl::base64encode(a.m_buffer.str()) +
            "\" type=\"audio/wav\" /></audio>";
        return bundle;
    }
}
```

In [ ]:
```
au::audio drums("audio/audio.wav");
drums
```

Display

In [ ]:
```
#include "xcpp/xdisplay.hpp"
```

In [ ]:
```
xcpp::display(drums);
```

Update-display

In [ ]:
```
#include <string>
#include "xcpp/xdisplay.hpp"

namespace ht
{
    struct html
    {
        inline html(const std::string& content)
        {
            m_content = content;
        }
        std::string m_content;
    };

    xeus::xjson mime_bundle_repr(const html& a)
    {
        auto bundle = xeus::xjson::object();
        bundle["text/html"] = a.m_content;
        return bundle;
    }
}

// A red rectangle
ht::html rect(R"(
<div style='
    width: 90px;
    height: 50px;
    line-height: 50px;
    background-color: blue;
    color: white;
    text-align: center;'>
Original
</div>)");
```

In [ ]:
```
xcpp::display(rect, "some_display_id");
```

In [ ]:
```
// Update the rectangle to be blue
rect.m_content = R"(
<div style='
    width: 90px;
    height: 50px;
    line-height: 50px;
    background-color: red;
    color: white;
    text-align: center;'>
Updated
</div>)";
```

```
xcpp::display(rect, "some_display_id", true);
```

Magics

Magics are special commands for the kernel that are not part of the C++ language.

They are defined with the symbol `%` for a line magic and `%%` for a cell magic.

More documentation for magics is available at https://xeus-cling.readthedocs.io/en/latest/magics.html.

In [ ]:
```
#include <algorithm>
#include <vector>
```

In [ ]:
```
std::vector<double> to_shuffle = {1, 2, 3, 4};
```

In [ ]:
```
%timeit std::random_shuffle(to_shuffle.begin(), to_shuffle.end());
```

Interactive Widgets

Jupyter interactive widgets are supported in the xeus-based C++ kernel.

- GitHub repository: https://github.com/QuantStack/xwidgets/
- Online documentation: https://xwidgets.readthedocs.io/

In [ ]:
```
#include "xwidgets/xslider.hpp"

xw::slider<double> slider;

slider
```

In [ ]:
```
// changing widget attribute will be reflected by the widget appearance

slider.max = 40;
slider.style().handle_color = "blue";
slider.orientation = "vertical";
slider.description = "A slider";
```

A method-chaining syntax allows to initialize widget attributes out of order, effictively mimicking keyword arguments.

In [ ]:
```
#include "xcpp/xdisplay.hpp"
```

```
auto other_slider = xw::slider_generator<double>()
    .min(-1.0)
    .max(1.0)
    .description("Another slider")
    .finalize();

xcpp::display(other_slider);
```

Backends for popular Jupyter widgets libraries have been writen for the C++ kernel. For example, the `xleaflet` package is a C++ backend to the `ipyleaflet` Jupyter interactive widget, offering the same functioalities.

In [ ]:
```
#include "xleaflet/xmap.hpp"
#include "xleaflet/xbasemaps.hpp"
#include "xleaflet/xtile_layer.hpp"
#include "xleaflet/xwms_layer.hpp"
#include "xleaflet/xlayers_control.hpp"
```

In [ ]:
```
// The method chaining syntax applies to all C++ Jupyter interactive widgets.

auto map = xlf::map_generator()
    .center({50, 354})
    .zoom(4)
    .finalize();

map
```

Adding a layers control to the map, allowing to toggle layers interactively.

In [ ]:
```
auto nasa_layer = xlf::basemap({"NASAGIBS", "ModisTerraTrueColorCR"},
"2018-03-30");
map.add_layer(nasa_layer);

auto wms = xlf::wms_layer_generator()
    .url("https://demo.boundlessgeo.com/geoserver/ows?")
    .layers("nasa:bluemarble")
    .name("nasa:bluemarble")
    .finalize();
```

In [ ]:
```
map.add_control(xlf::layers_control());
```

- GitHub repository: https://github.com/QuantStack/xtensor/

- Online documentation: https://xtensor.readthedocs.io/
- NumPy to xtensor cheat sheet: http://xtensor.readthedocs.io/en/latest/numpy.html

`xtensor` is a C++ library for manipulating N-D arrays with an API very similar to that of numpy.

In [ ]:
```
#include <iostream>

#include "xtensor/xarray.hpp"
#include "xtensor/xio.hpp"
#include "xtensor/xview.hpp"

xt::xarray<double> arr1
  {{1.0, 2.0, 3.0},
   {2.0, 5.0, 7.0},
   {2.0, 5.0, 7.0}};

xt::xarray<double> arr2
  {5.0, 6.0, 7.0};

xt::view(arr1, 1) + arr2
```

Together with the C++ Jupyter kernel, `xtensor` offers a similar experience as `NumPy` in the Python Jupyter kernel, including broadcasting and universal functions.

In [ ]:
```
#include <iostream>
#include "xtensor/xarray.hpp"
#include "xtensor/xio.hpp"
```

In [ ]:
```
xt::xarray<int> arr
  {1, 2, 3, 4, 5, 6, 7, 8, 9};

arr.reshape({3, 3});

std::cout << arr;
```

In [ ]:
```
#include "xtensor-blas/xlinalg.hpp"
```

In [ ]:
```
xt::xtensor<double, 2> m = {{1.5, 0.5}, {0.7, 1.0}};
std::cout << "Matrix rank: " << std::endl << xt::linalg::matrix_rank(m) <<
std::endl;
std::cout << "Matrix inverse: " << std::endl << xt::linalg::inv(m) << std::endl;
std::cout << "Eigen values: " << std::endl << xt::linalg::eigvals(m) << std::endl;
```

In [ ]:
```
xt::xarray<double> arg1 = xt::arange<double>(9);
xt::xarray<double> arg2 = xt::arange<double>(18);

arg1.reshape({3, 3});
arg2.reshape({2, 3, 3});

std::cout << xt::linalg::dot(arg1, arg2) << std::endl;
```

In [ ]: