Scientific Computing TU Berlin Winter 2021/22 © Jürgen Fuhrmann
Notebook 06

```
using PlutoUI
```

Table of Contents

Interaction with other languages

C

- C language code has a well defined binary interface
  - `int` ↔ `Int32`
  - `float` ↔ `Float32`
  - `double` ↔ `Float64`
  - C arrays as pointers
- Create a C source file:

```
cadd_source = "double cadd(double x, double y) \n{ \n   return x+y; \n}\n"
```

```
cadd_source="""
double cadd(double x, double y)
{
   return x+y;
}
"""
```

```
open("cadd.c", "w") do io
    write(io,cadd_source)
end;
```

Compile to a shared object (aka "dll" on windows) using the gcc compiler:

```
Process(`gcc --shared cadd.c -o libcadd.so`, ProcessExited(0))
```

```
run(`gcc --shared  cadd.c -o libcadd.so`)
```

- Define wrapper function `cadd` using the Julia `ccall` method
  - `(:cadd, "libcadd")` : call cadd from `libcadd.so`
  - First `Float64` : return type
  - Tuple `(Float64,Float64,)` : parameter types
  - `x,y` : actual data passed
- At its first call it will load `libcadd.so` into Julia
- Direct call of compiled C function `cadd()` , no intermediate wrapper code

```
cadd (generic function with 1 method)
```

```
cadd(x,y)=ccall((:cadd, "libcadd"), Float64, (Float64,Float64,),x,y)
```

```
11.5
```

```
cadd(1.5,10)
```

It is also possible to call Julia code from C

Python

- Both Julia and Python are homoiconic languages, featuring *reflection*
- They can parse the elements of their own data structures ⇒ possibility to automatically build proxies for python objects in Julia

The PyCall package provides the corresponding interface:

```
using PyCall
```

Create a python source file:

```
pyadd_source = "def add(x,y):\n    return x+y\n"
```

```
pyadd_source="""
def add(x,y):
    return x+y
"""
```

```
open("pyadd.py", "w") do io
    write(io,pyadd_source)
end;
```

```
pyadd =
PyObject <module 'pyadd' from '/home/fuhrmann/Wias/teach/scicomp/pluto/pyadd.py'>
```

```
pyadd=pyimport("pyadd")
```

```
4.5
```

```
pyadd.add(3.5,1.0)
```

- Julia allows to call almost any python package
- E.g. matplotlib graphics - this is the python package behind PyPlot (there are more graphics options in Julia)
- There is also a **pyjulia** package allowing to call Julia from python

Javascript

Javascript can be used in Pluto to display things, or to add interactive elements.

We need the ability to interpolate into html strings:

```
using HypertextLiteral
```

We need to be able to generate random id's for html elements:

```
using UUIDs
```

Calculate the sum of two values in javascript and return it to Julia. This uses the magic behind PlutoUI sliders.

```
jsadd (generic function with 1 method)
```

```
function jsadd(a,b)
    id="($uuid1())"
    htl"""
            <div id=$(id)>
            <script>
                document.getElementById($id).value=$(a)+$(b)
            </script>
         </div>
        """
end
```

```
@bind y jsadd(3,25)
```

```
28
```

```
y
```

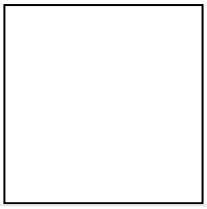
And here an example by **Connor Burns**:

```
MouseMoveInput (generic function with 3 methods)
```

```
function MouseMoveInput(width=300, height=300)
    id="$(uuid1())"
    @htl("""
    <div id=$id style="width: $(width)px; height: $(height)px; border: 1px solid black; cursor:crosshair ">
        <script>
            const mouseInput = document.getElementById( $id );

            mouseInput.value = [0, 0];

            mouseInput.addEventListener('mousemove', function(e) {
                const rect = mouseInput.getBoundingClientRect();
                mouseInput.value = [Math.floor(e.x-rect.left+1), Math.floor(e.y-rect.top+1)];
                mouseInput.dispatchEvent(new Event('input'));
            });
        </script>
    </div>
    """)
end
```



```
@bind xy MouseMoveInput(100,100)
```

```
0×0 Matrix{Float64}
```

```
rand(xy...)
```

Other languages

- There are ways to interact with C++, R, Rust and other langugas
- Interaction with Fortran via `ccall`
- Pluto: integration with Javascript in browser
- Julia and many of its packages use this method to access a number of highly optimized linear algebra and other libraries written in C and Fortran