Programming in Julia & Jupyter Notebooks

ROB 102: Introduction to AI & Programming

Lab Session 7

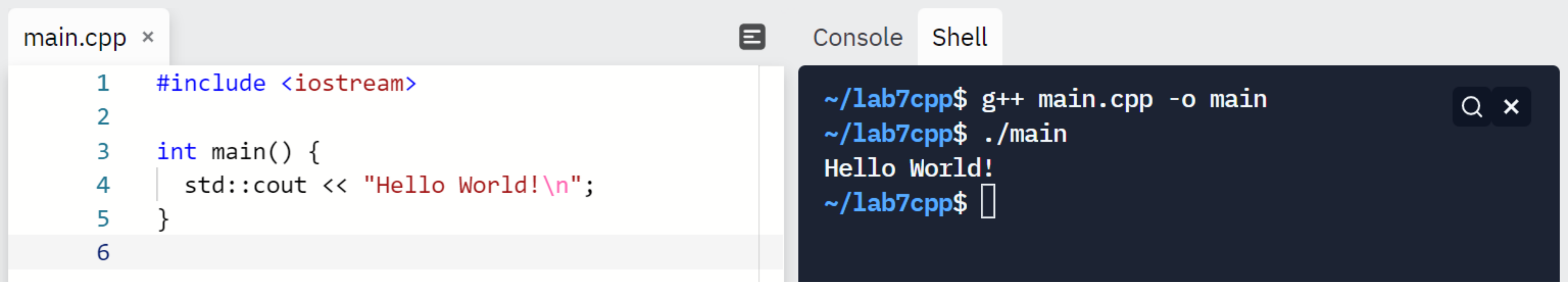
2021/11/19

Today

1. Coding in Julia
2. Using a Jupyter Notebook

Running code in Julia vs. C++

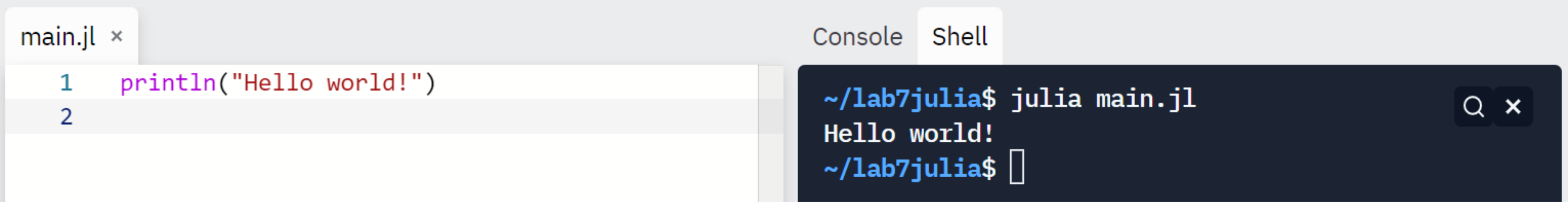
In C++, we need to compile our code into an executable and then run it.



```cpp
#include <iostream>

int main() {
  std::cout << "Hello World!\n";
}

```

```
~/lab7cpp$ g++ main.cpp -o main
~/lab7cpp$ ./main
Hello World!
~/lab7cpp$ 
```

In Julia, code is executed line by line without a compiler. Julia scripts do not need a main function.



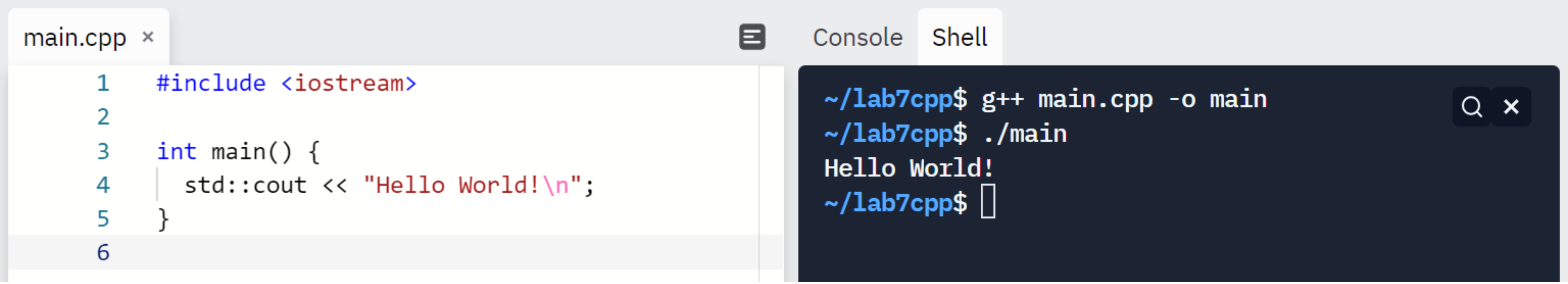
```julia
println("Hello world!")

```

```
~/lab7julia$ julia main.jl
Hello world!
~/lab7julia$ 
```

Statements in Julia do not need a semi-colon at the end of them.

Printing in Julia vs. C++

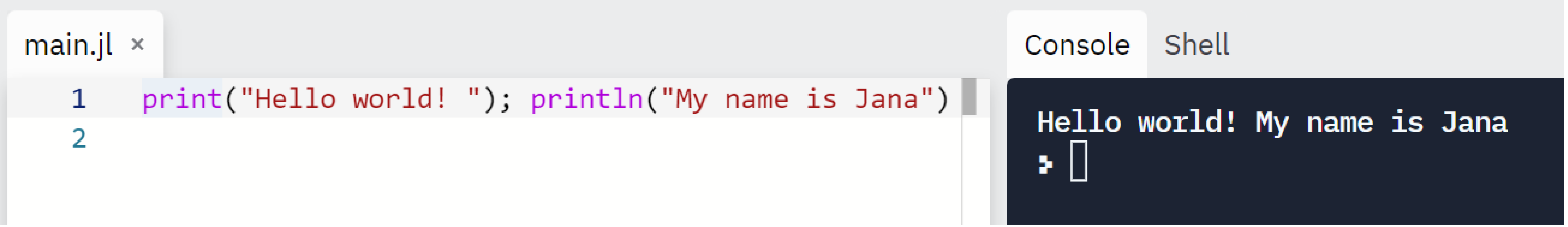
In C++, we print using `std::cout`.



```cpp
#include <iostream>

int main() {
  std::cout << "Hello World!\n";
}

```

```
~/lab7cpp$ g++ main.cpp -o main
~/lab7cpp$ ./main
Hello World!
~/lab7cpp$ 
```

In Julia, the `println(“…”)` function prints the content and then skips a line. `print(“…”)` prints without skipping a line.



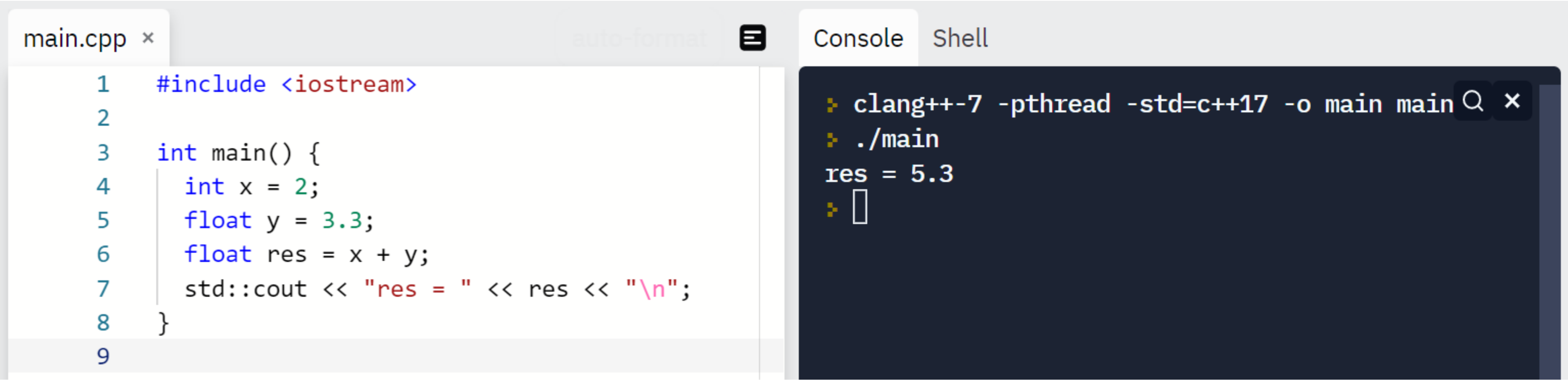
```julia
print("Hello world! "); println("My name is Jana")

```

```
Hello world! My name is Jana
```

In Julia, you can include a semi-colon to indicate the end of a line.

Variables & Types

In C++, variables need to be declared with their types.

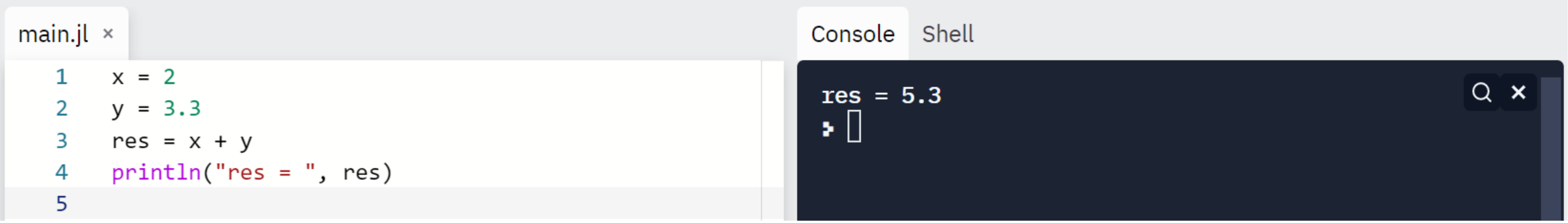


```cpp
#include <iostream>

int main() {
  int x = 2;
  float y = 3.3;
  float res = x + y;
  std::cout << "res = " << res << "\n";
}

```

```
› clang++-7 -pthread -std=c++17 -o main main
› ./main
res = 5.3
› 
```

Variables & Types

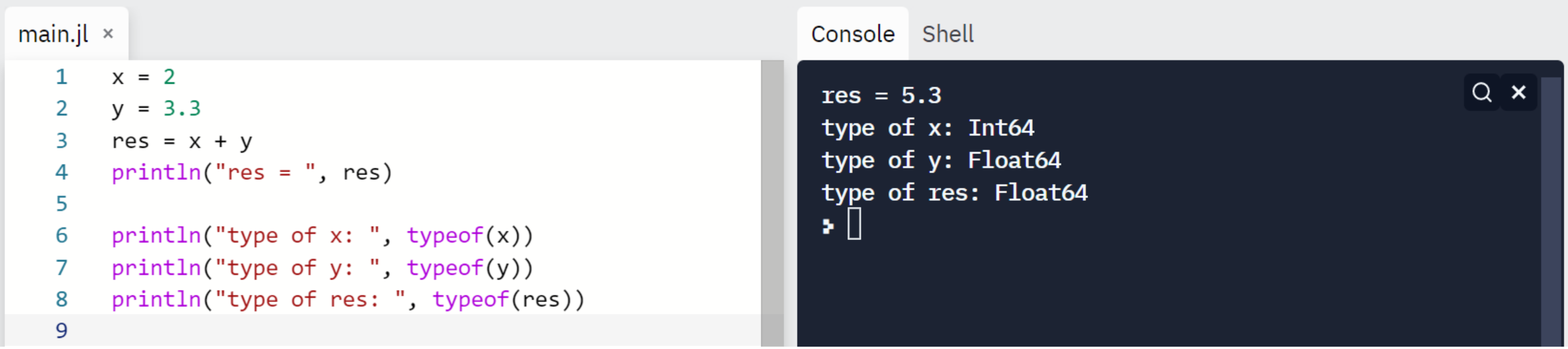
In Julia, variables are declared without any types. Julia figures out what the type is for you.



```julia
x = 2
y = 3.3
res = x + y
println("res = ", res)
```

```
res = 5.3
```

You can find out the type of a variable using the `typeof(…)` function.

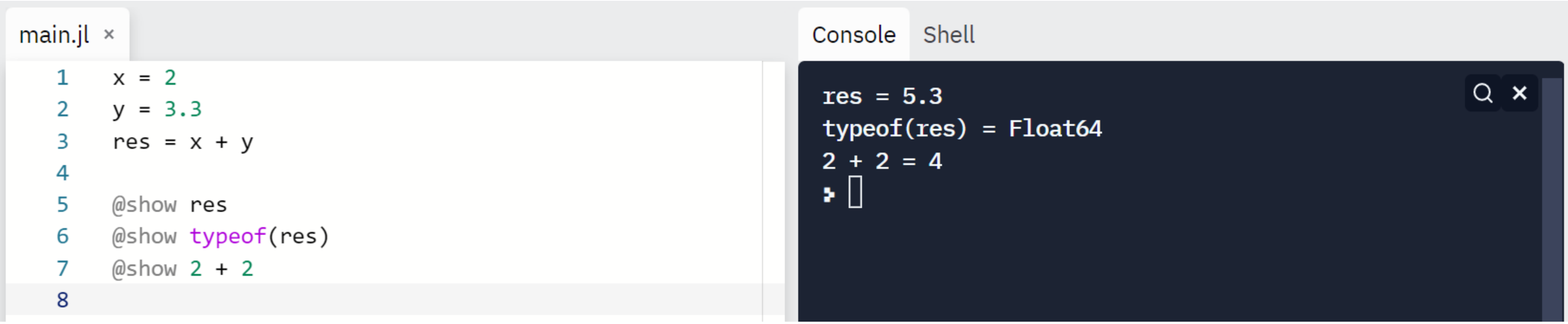


```julia
x = 2
y = 3.3
res = x + y
println("res = ", res)

println("type of x: ", typeof(x))
println("type of y: ", typeof(y))
println("type of res: ", typeof(res))
```

```
res = 5.3
type of x: Int64
type of y: Float64
type of res: Float64
```

Printing Revisited

A useful function for showing a line and the result of the line (instead of `print(…)` or `println(…)` is `show`:

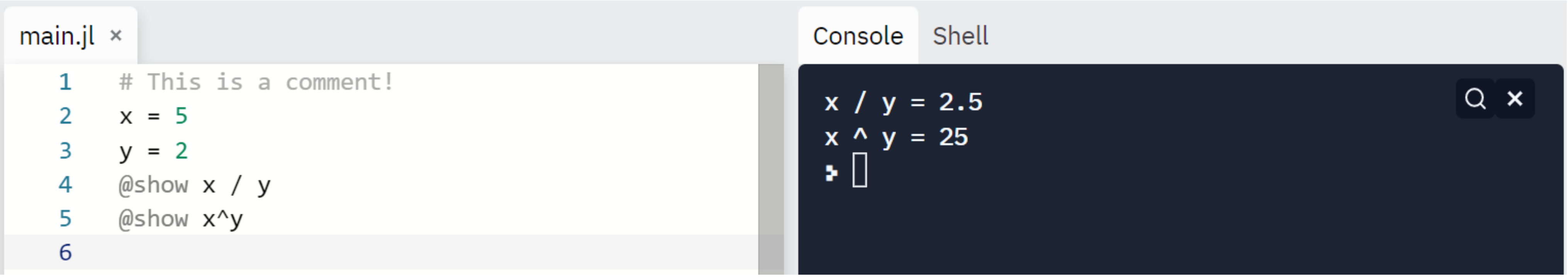


```julia
x = 2
y = 3.3
res = x + y

@show res
@show typeof(res)
@show 2 + 2
```

```
res = 5.3
typeof(res) = Float64
2 + 2 = 4
```

Operators

The standard math operators (+, -, *, /, %) are the same as in C++, except that dividing two integers gives a float. Julia also has an operator, ^, which is the same as pow( ) in C++.



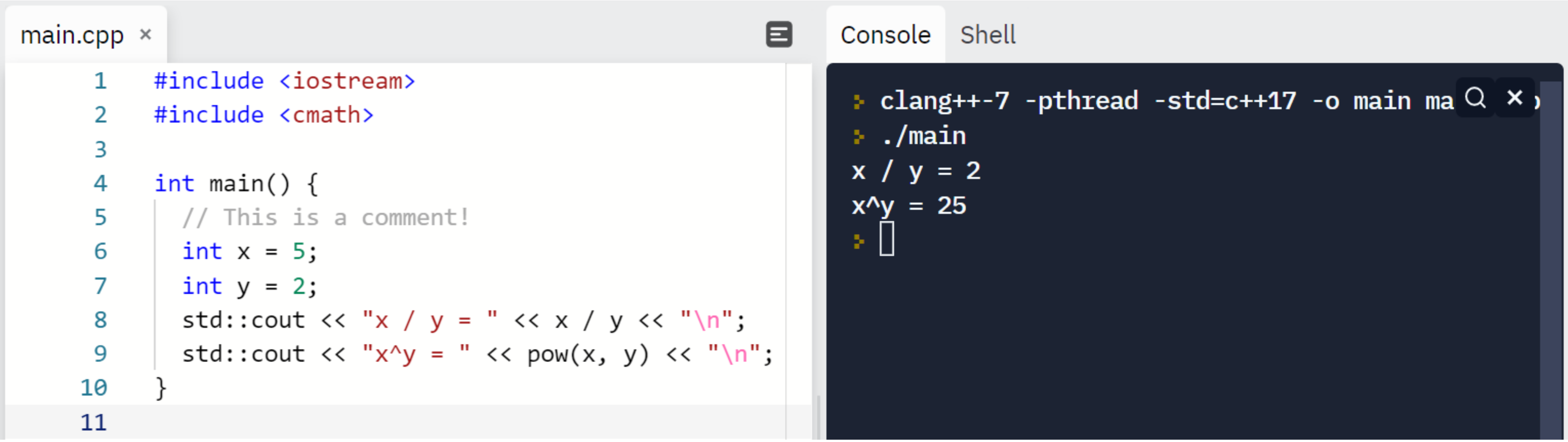
```julia
# This is a comment!
x = 5
y = 2
@show x / y
@show x^y

```

```
x / y = 2.5
x ^ y = 25
```

Julia code.

Operators

The standard math operators (+, -, *, /, %) are the same as in C++, except that dividing two integers gives a float. Julia also has an operator, ^, which is the same as pow( ) in C++.



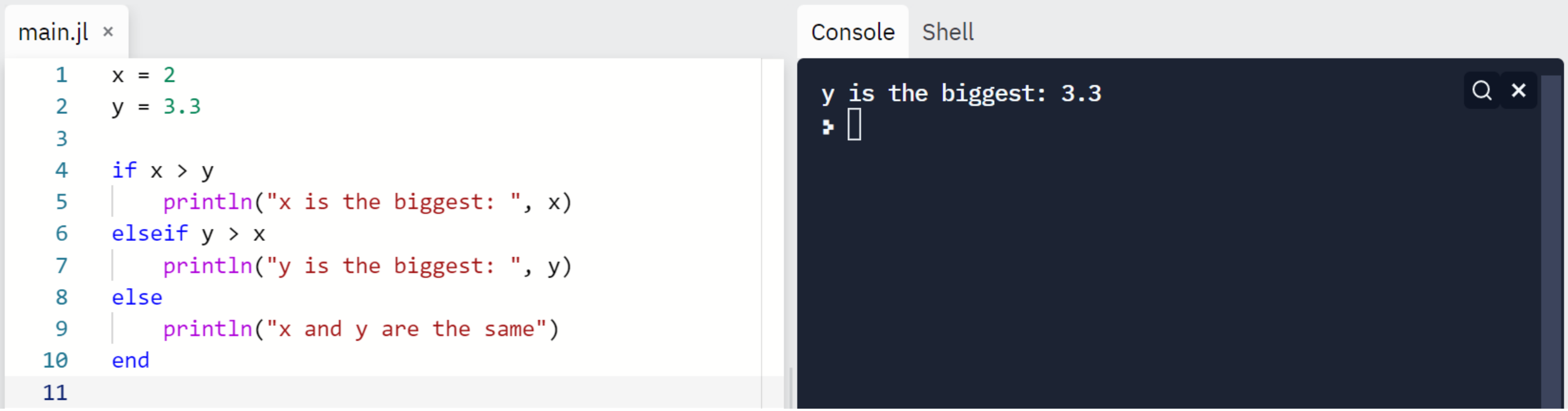
```cpp
#include <iostream>
#include <cmath>

int main() {
  // This is a comment!
  int x = 5;
  int y = 2;
  std::cout << "x / y = " << x / y << "\n";
  std::cout << "x^y = " << pow(x, y) << "\n";
}

```

```
clang++-7 -pthread -std=c++17 -o main ma
./main
x / y = 2
x^y = 25
```

C++ code.

Control flow: If Statements

In Julia, if statements don’t need brackets around the condition or curly brackets around the code.



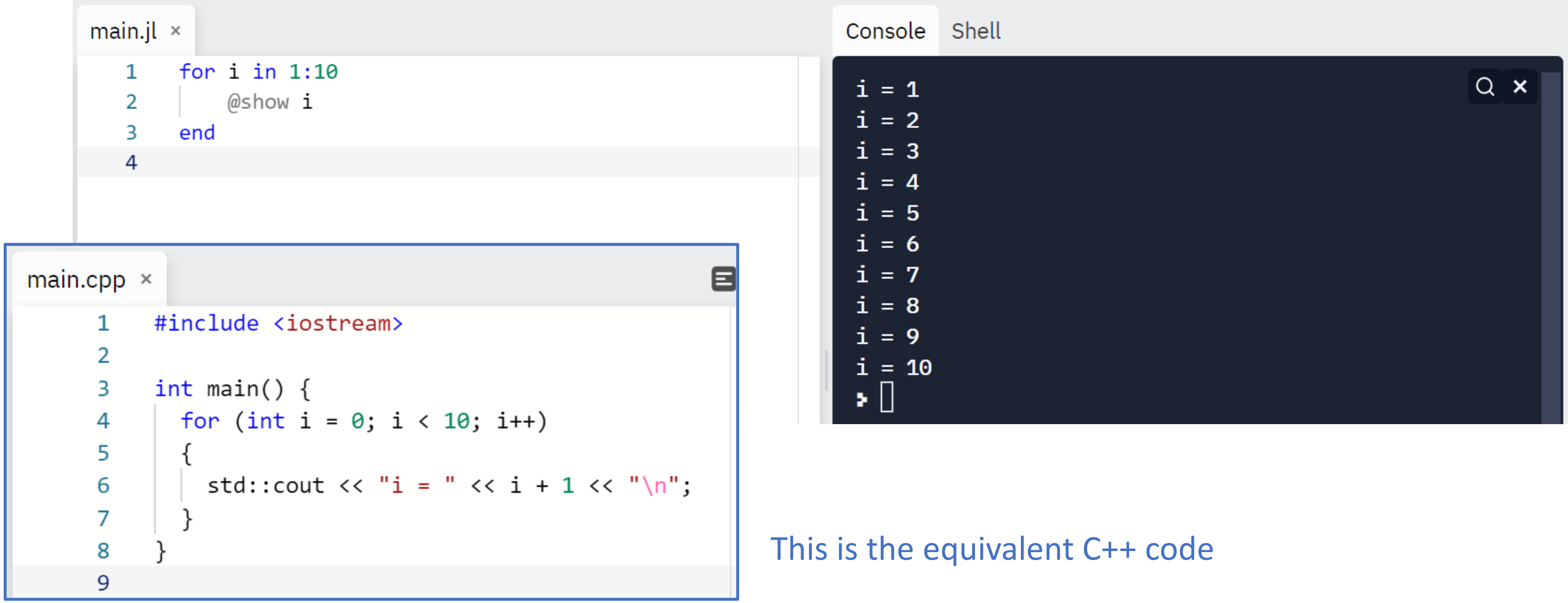
```julia
x = 2
y = 3.3

if x > y
    println("x is the biggest: ", x)
elseif y > x
    println("y is the biggest: ", y)
else
    println("x and y are the same")
end

```

```
y is the biggest: 3.3
```

If statements end with the end keyword. We write `elseif` instead of `else if` in C++.

Control flow: Loops

The syntax for a for loop in Julia looks like this:



```julia
for i in 1:10
    @show i
end
```

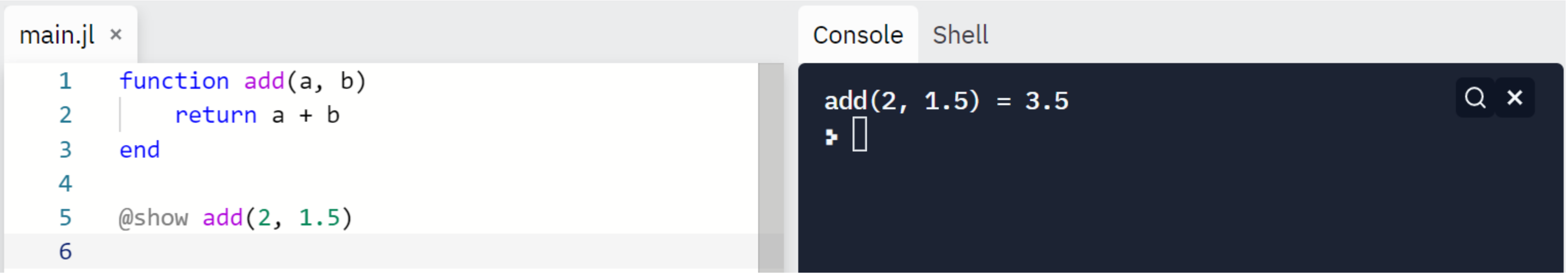
```
i = 1
i = 2
i = 3
i = 4
i = 5
i = 6
i = 7
i = 8
i = 9
i = 10
```

```cpp
#include <iostream>

int main() {
  for (int i = 0; i < 10; i++)
  {
    std::cout << "i = " << i + 1 << "\n";
  }
}
```

This is the equivalent C++ code

Functions

In Julia, functions are declared with the `function` keyword. The end keyword ends a function.



```julia
function add(a, b)
    return a + b
end

@show add(2, 1.5)
```

```
add(2, 1.5) = 3.5
```

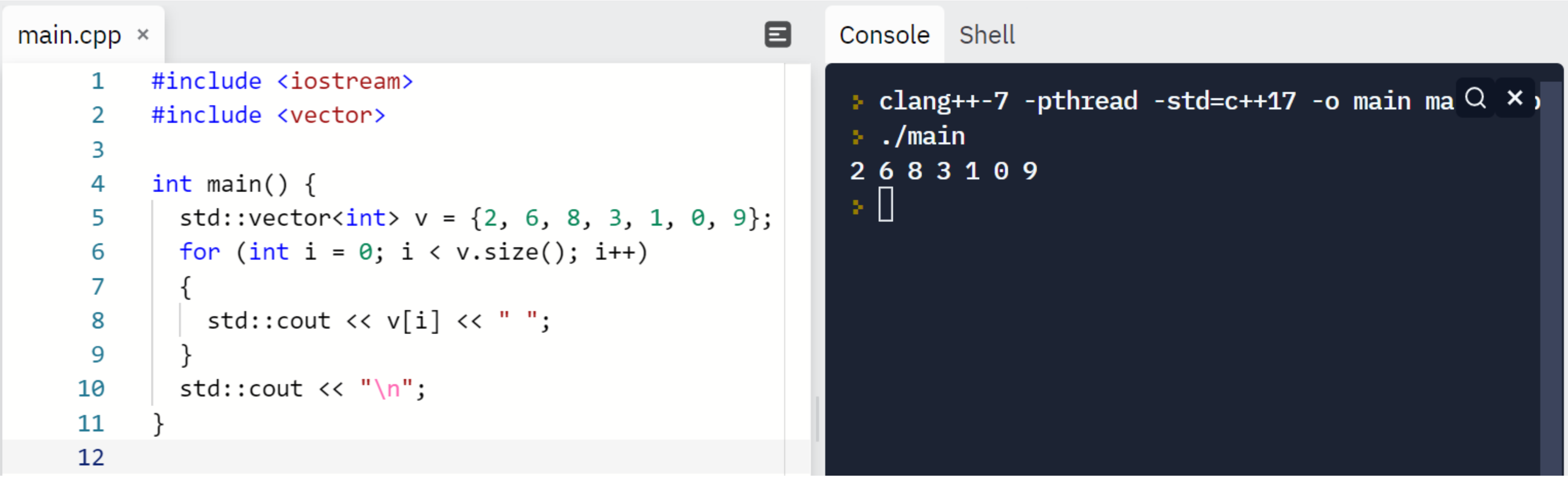
Functions do not need return types, and function parameters do not need types.

Vectors & Matrices

Julia is especially good at dealing with vectors and matrices. A vector can be created like this:

```
v = [1, 2, 3, 4]
```

In C++, we iterated through each index of the vector using its size, like this:



```cpp
#include <iostream>
#include <vector>

int main() {
  std::vector<int> v = {2, 6, 8, 3, 1, 0, 9};
  for (int i = 0; i < v.size(); i++)
  {
    std::cout << v[i] << " ";
  }
  std::cout << "\n";
}

```

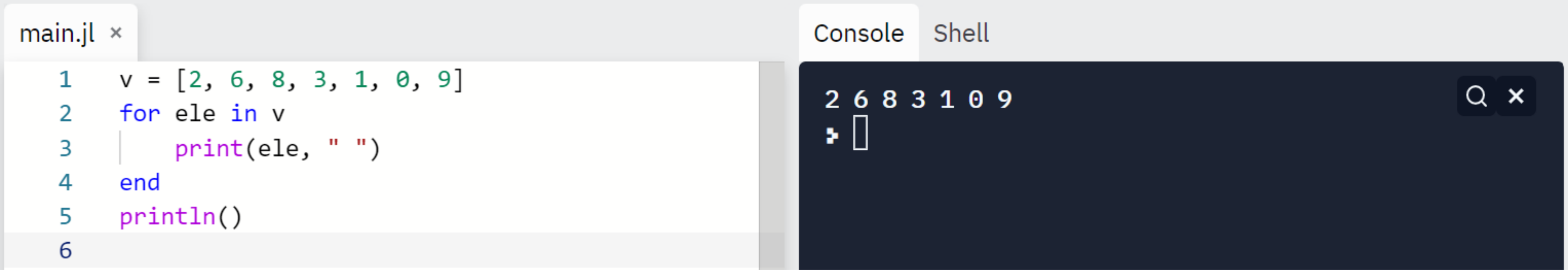
```
clang++-7 -pthread -std=c++17 -o main ma
./main
2 6 8 3 1 0 9
```

Vectors & Matrices

Julia is especially good at dealing with vectors and matrices. A vector can be created like this:

```
v = [1, 2, 3, 4]
```

In Julia, we can iterate through elements in a vector directly:



```
v = [2, 6, 8, 3, 1, 0, 9]
for ele in v
    print(ele, " ")
end
println()
```

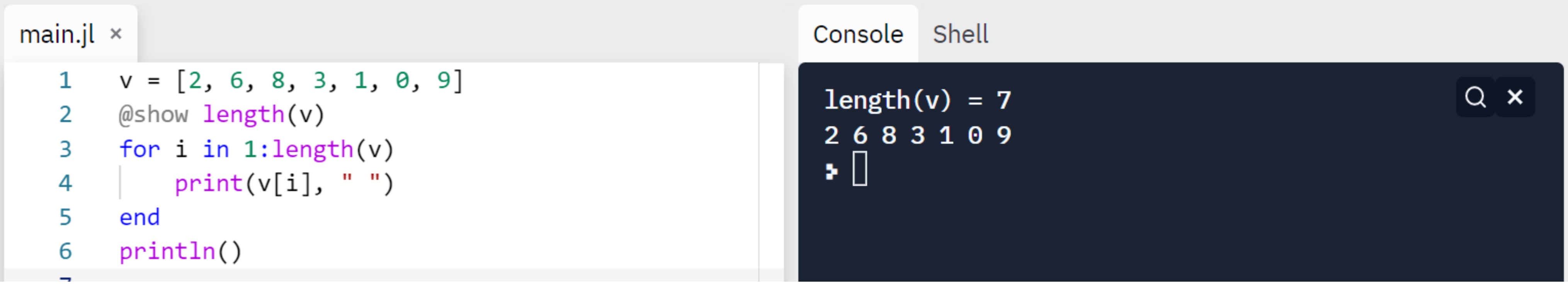
```
2 6 8 3 1 0 9
```

Vectors & Matrices

Julia is especially good at dealing with vectors and matrices. A vector can be created like this:

```
v = [1, 2, 3, 4]
```

We can iterate by index as well, using `length(v)` to get the length.



```julia
v = [2, 6, 8, 3, 1, 0, 9]
@show length(v)
for i in 1:length(v)
    print(v[i], " ")
end
println()
```

```
length(v) = 7
2 6 8 3 1 0 9
```

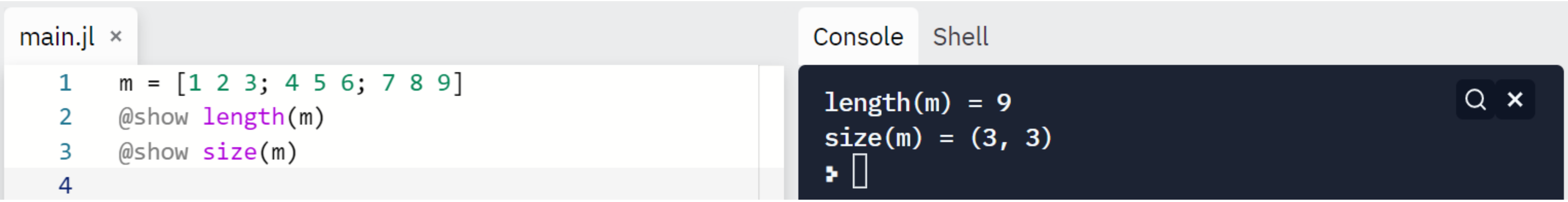
In Julia, vectors are indexed starting at 1. `v[1]` is the first element, not `v[0]`.

Vectors & Matrices

Creating a matrix is similar to creating a vector. Semi-colons separate rows, and spaces separate elements in a row:

```
m = [1 2 3; 4 5 6; 7 8 9]
```

The length of a matrix is the total number of elements in it. The size of the matrix is a tuple containing the number of rows and columns.



```julia
m = [1 2 3; 4 5 6; 7 8 9]
@show length(m)
@show size(m)
```

```
length(m) = 9
size(m) = (3, 3)
```

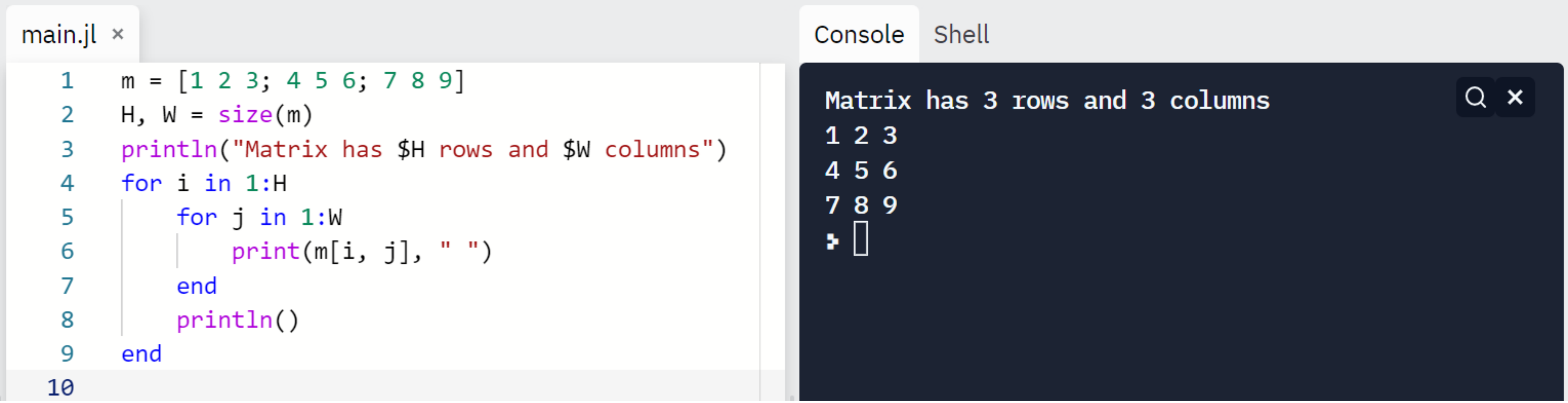
A *tuple* is like a vector, but it doesn't allow its values to be modified.

Vectors & Matrices

Creating a matrix is similar to creating a vector. Semi-colons separate rows, and spaces separate elements in a row:

```
m = [1 2 3; 4 5 6; 7 8 9]
```

Indexing into a matrix can be done by providing the row and column index of a matrix, separated by commas:

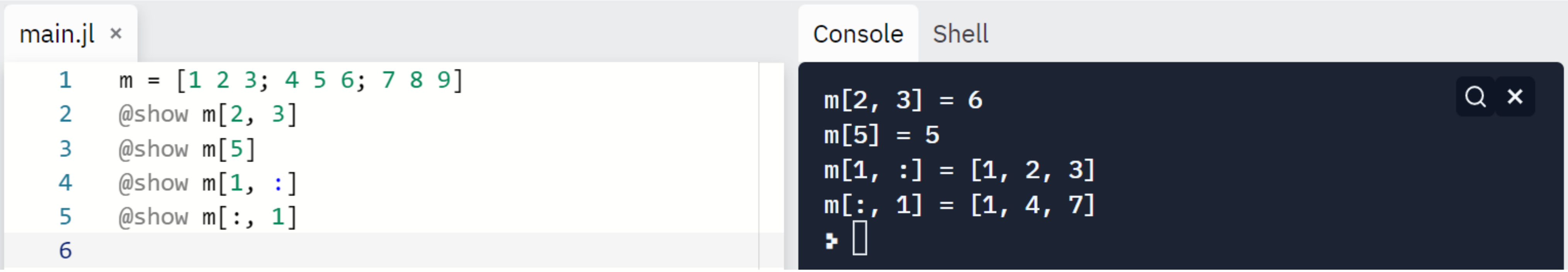


```julia
m = [1 2 3; 4 5 6; 7 8 9]
H, W = size(m)
println("Matrix has $H rows and $W columns")
for i in 1:H
    for j in 1:W
        print(m[i, j], " ")
    end
    println()
end
```

```
Matrix has 3 rows and 3 columns
1 2 3
4 5 6
7 8 9
```

Vectors & Matrices

- Indexing into a matrix can be done by providing the row and column index of a matrix, separated by commas.
- Indexing into a matrix with just one value treats the matrix as a flattened vector.
- Colons let us pick all the values in an axis. `m[1, :]` is all the values in row 1. `m[:, 1]` is all the values in column 1.

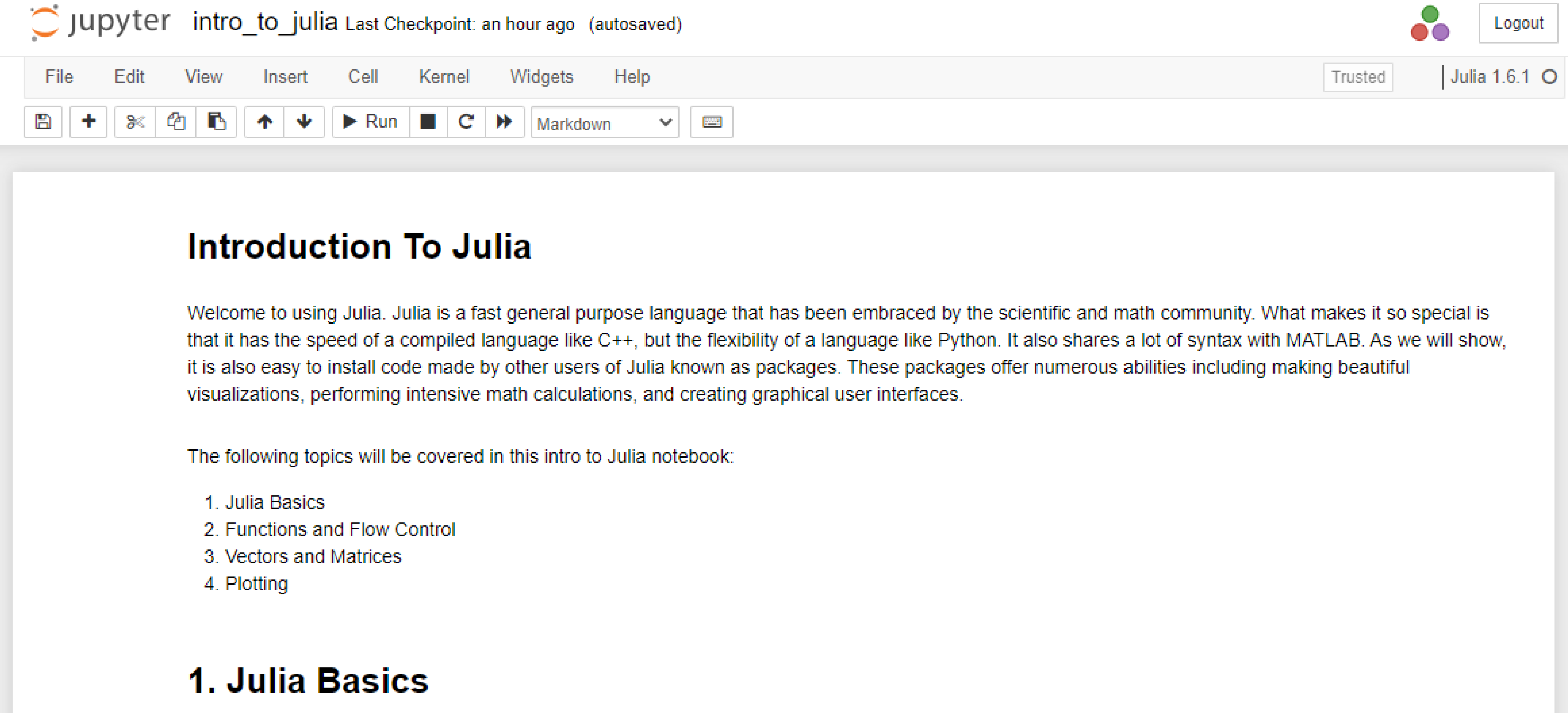


```julia
m = [1 2 3; 4 5 6; 7 8 9]
@show m[2, 3]
@show m[5]
@show m[1, :]
@show m[:, 1]
```

```
m[2, 3] = 6
m[5] = 5
m[1, :] = [1, 2, 3]
m[:, 1] = [1, 4, 7]
```

Jupyter Notebooks

A Jupyter is a collection of code and text cells which allows for fast execution and visualization of code.



Introduction To Julia

Welcome to using Julia. Julia is a fast general purpose language that has been embraced by the scientific and math community. What makes it so special is that it has the speed of a compiled language like C++, but the flexibility of a language like Python. It also shares a lot of syntax with MATLAB. As we will show, it is also easy to install code made by other users of Julia known as packages. These packages offer numerous abilities including making beautiful visualizations, performing intensive math calculations, and creating graphical user interfaces.

The following topics will be covered in this intro to Julia notebook:

1. Julia Basics
2. Functions and Flow Control
3. Vectors and Matrices
4. Plotting

1. Julia Basics

TODO:

1. Complete the setup instructions to install Julia and the necessary packages.
2. Accept the “Intro to Julia” assignment.
   - The assignment links are in the Google Doc linked on Slack.
3. Go through the Intro to Julia assignment and complete the exercises (optional but encouraged).