

Programming in Julia: Pocket Calculator

ROB 102: Introduction to AI & Programming

2021/11/29

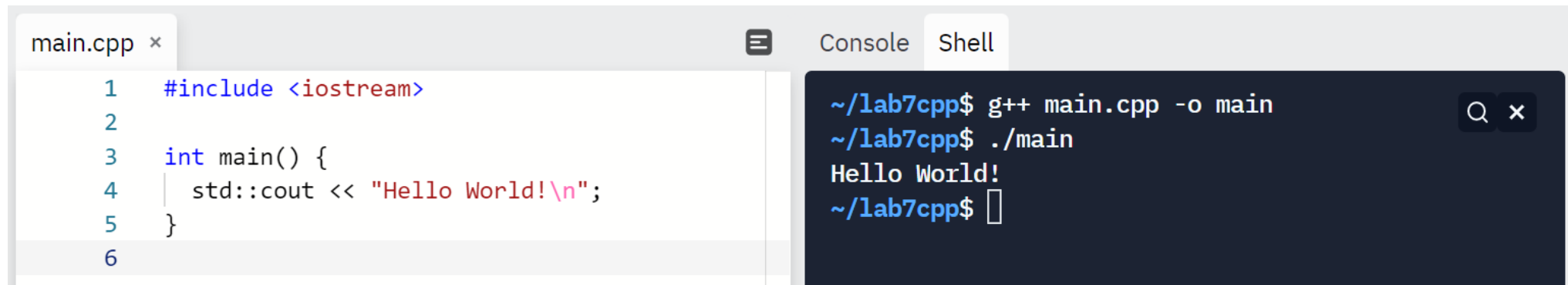
Today...

1. Julia refresher
2. Pocket calculator in Julia

See the lab slides!

Running code in Julia vs. C++

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



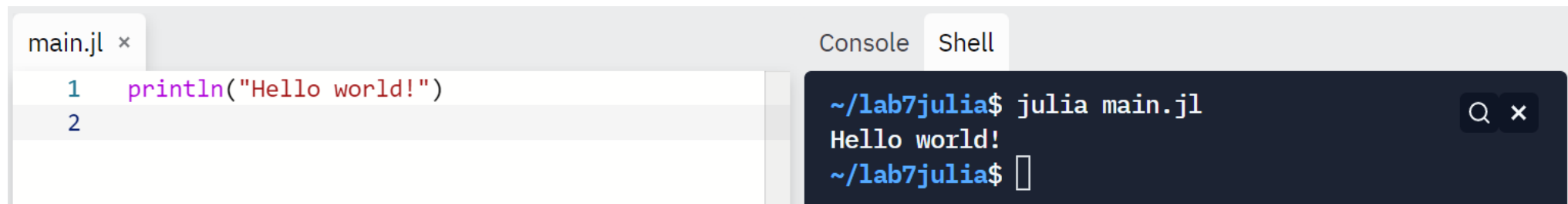
The screenshot shows a code editor with a file named 'main.cpp'. The code is as follows:

```
1 #include <iostream>
2
3 int main() {
4     std::cout << "Hello World!\n";
5 }
6
```

To the right of the code editor is a terminal window with the following commands and output:

```
~/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.



The screenshot shows a code editor with a file named 'main.jl'. The code is as follows:

```
1 println("Hello world!")
2
```

To the right of the code editor is a terminal window with the following commands and output:

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

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

Julia Pocket Calculator

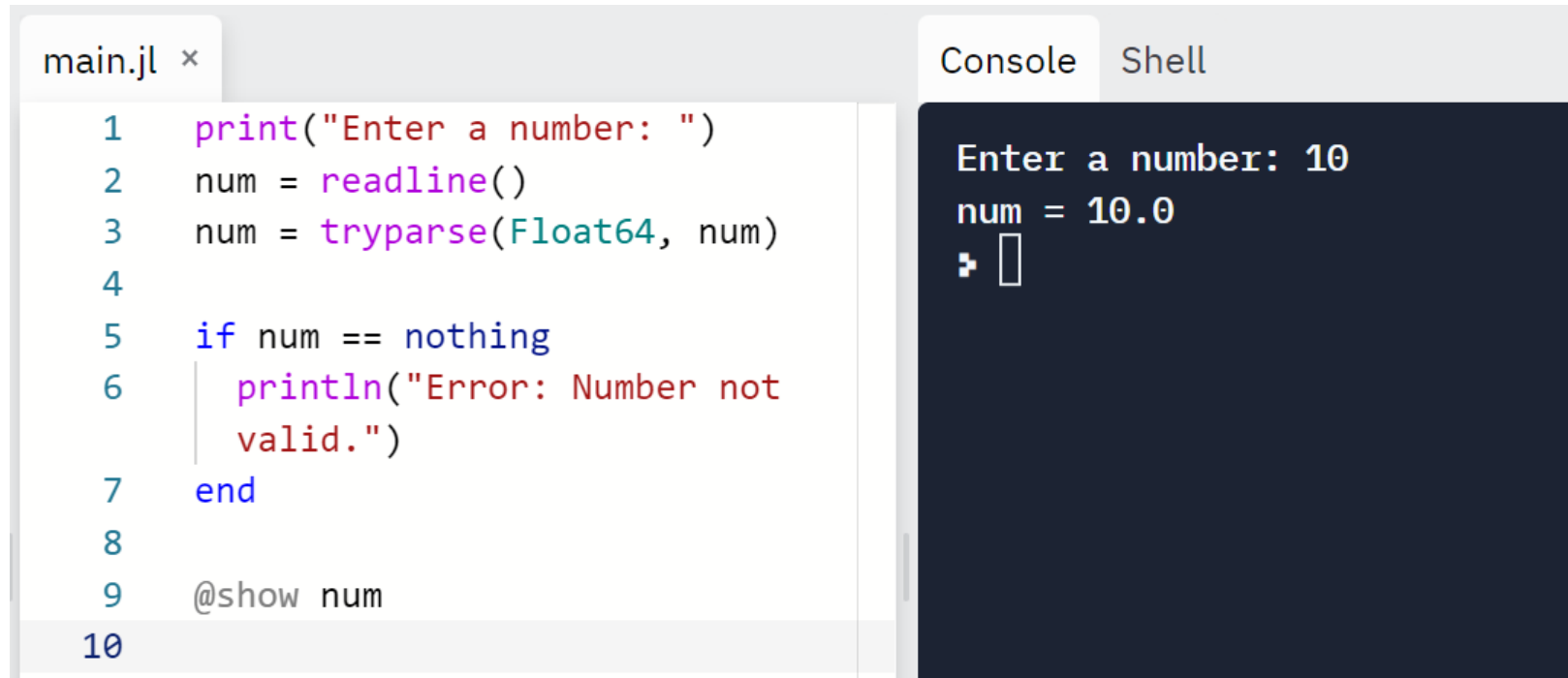
Task for today: Write the basic pocket calculator in Julia.

1. Get a number from the user.
2. Get an operator from the user.
3. If the operator is “q”, end the program.
4. Get another number from the user.
5. Calculate the result and print it out.
6. Save the result as the first number. Go to 2.

```
Console Shell
Welcome to the pocket calculator!
Enter a number: 10
Enter an operator (+, -, *, /, q): +
Enter a number: 5
10.0 + 5.0 = 15.0
Enter an operator (+, -, *, /, q): /
Enter a number: 5
15.0 / 5.0 = 3.0
Enter an operator (+, -, *, /, q): *
Enter a number: 100
3.0 * 100.0 = 300.0
Enter an operator (+, -, *, /, q): q
> 
```

Getting input from the user

The `readline()` function gets user input as a string.



The image shows a code editor window with a file named `main.jl`. The code in the editor is as follows:

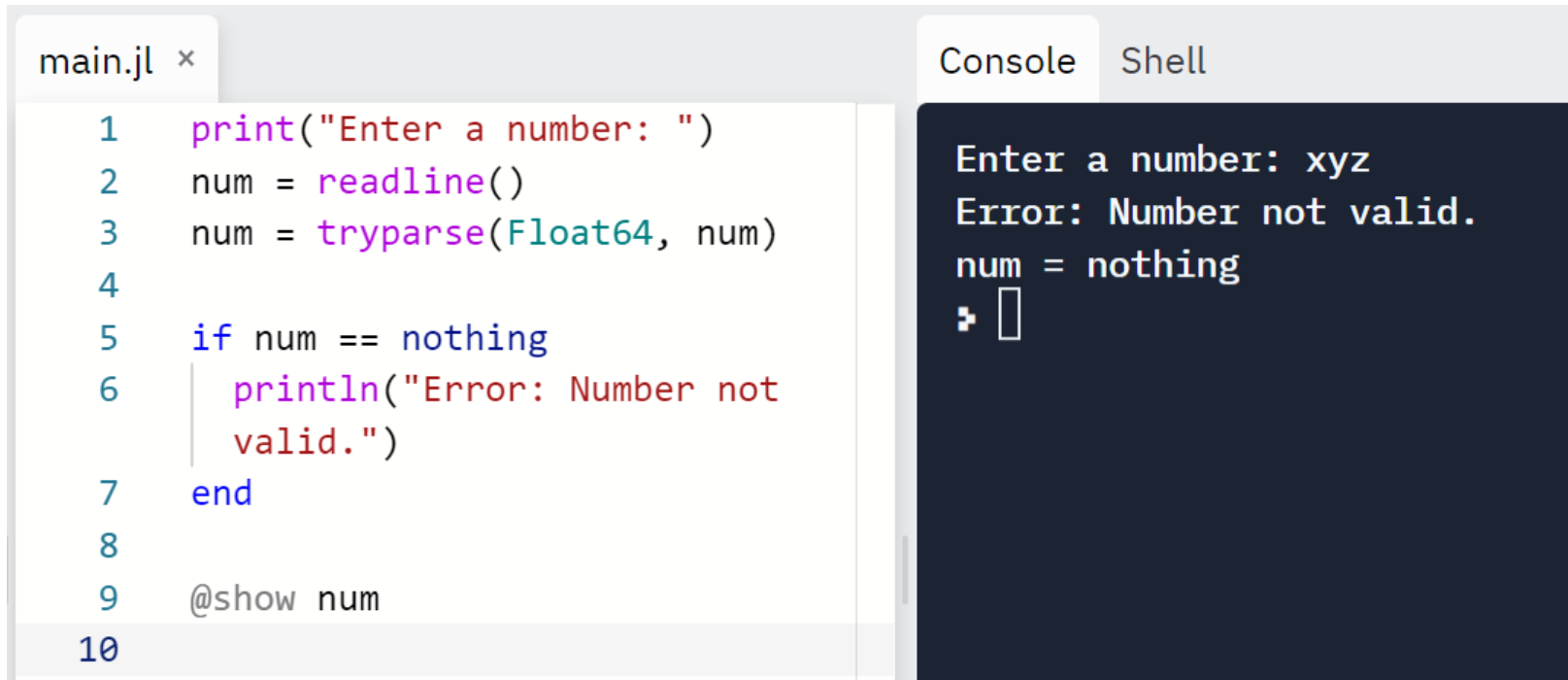
```
1 print("Enter a number: ")
2 num = readline()
3 num = tryparse(Float64, num)
4
5 if num == nothing
6     println("Error: Number not
7     valid.")
8 end
9 @show num
10
```

To the right of the code editor is a console window. The console output is:

```
Enter a number: 10
num = 10.0
└─> []
```

Getting input from the user

The `readline()` function gets user input as a string.



The screenshot shows a code editor window titled 'main.jl' with the following code:

```
1 print("Enter a number: ")
2 num = readline()
3 num = tryparse(Float64, num)
4
5 if num == nothing
6     println("Error: Number not
7     valid.")
8 end
9 @show num
10
```

To the right, the 'Console' window shows the execution output:

```
Enter a number: xyz
Error: Number not valid.
num = nothing
> []
```

`nothing` is a special keyword in Julia which is an empty variable.

Warning about scope...

```
main.jl x
1  i = 0
2  while i < 10
3  |   i += 1
4  end
5  @show i
6
```

What do you expect to happen?

Warning about scope...

```
main.jl x
1  i = 0
2  while i < 10
3  |   i += 1
4  end
5  @show i
6
```

What do you expect to happen?

```
Console Shell
error during bootstrap:
LoadError("main.jl", 2, UndefVarError(:i))
jl_undefined_var_error at /buildworker/worker/package_linux64/build/src/rtutils.c:130
top-level scope at ./main.jl:3
jl_toplevel_eval_flex at /buildworker/worker/package_linux64/build/src/toplevel.c:808
jl_parse_eval_all at /buildworker/worker/package_linux64/build/src/ast.c:872
jl_load at /buildworker/worker/package_linux64/build/src/toplevel.c:872
unknown function (ip: 0x4beb75)
unknown function (ip: 0x45410f)
```

Variable `i` is undefined!!

Warning about scope...

```
main.jl x
1  i = 0
2  while i < 10
3  |   i += 1
4  end
5  @show i
6
```

We say `i` is defined in the **global scope**, since it's defined outside any loop or function. Any code can access variables in global scope.

To protect global variables from getting modified accidentally, **they cannot be written to** outside the global scope (but they can be read).

Warning about scope...

```
main.jl x
1  i = 0
2  while i < 10
3  |  @show i
4  end
5
```

```
Console  Shell
i = 0
i = 0
i = 0
i = 0
i = 0
i = 0
```

We say `i` is defined in the **global scope**, since it's defined outside any loop or function. Any code can access variables in global scope.

To protect global variables from getting modified accidentally, **they cannot be written to** outside the global scope (but they can be read).

Legal, but creates an infinite loop!

Warning about scope...

```
main.jl x
1  i = 0
2  while i < 10
3  |  global i += 1
4  end
5  @show i
6
```

```
Console Shell
i = 10
+ []
```

The `global` keyword lets you modify a global variable locally.

Note: There is no need to use `global` in Jupyter notebooks!

Warning about scope...

```
main.jl x
1  function loop()
2      i = 0
3      while i < 10
4          i += 1
5      end
6      @show i
7  end
8
9  loop()
10
```

Console Shell

```
i = 10
┌─┐
```

Note: There is no need to use `global` inside functions.

Julia Pocket Calculator

Task for today: Write the basic pocket calculator in Julia.

1. Get a number from the user.
2. Get an operator from the user.
3. If the operator is “q”, end the program.
4. Get another number from the user.
5. Calculate the result and print it out.
6. Save the result as the first number. Go to 2.

Bonus 1: Handle bad user input.

Bonus 2: Include an option to select a random operator.