Getting started with Julia

INTRODUCTION TO JULIA

James Fulton
Climate informatics researcher

What is Julia?

- An open-source language designed for data science
- Newer than other common languages
- General purpose
- Designed for scientific computing and data tasks

Language	Release year
MATLAB	1978
Python	1991
R	1993
Julia	**2012**

Why create Julia?

Designed by Jeff Bezanson, Stefan Karpinski, Viral B. Shah, and Alan Edelman

Julia was designed to be:

- Simple to learn
- Able to keep the most serious hackers happy
- As usable for general programming as Python
- As easy for statistics as R
- As powerful and natural for linear algebra as MATLAB
- Speed of C

[1] https://julialang.org/blog/2012/02/why-we-created-julia

This course

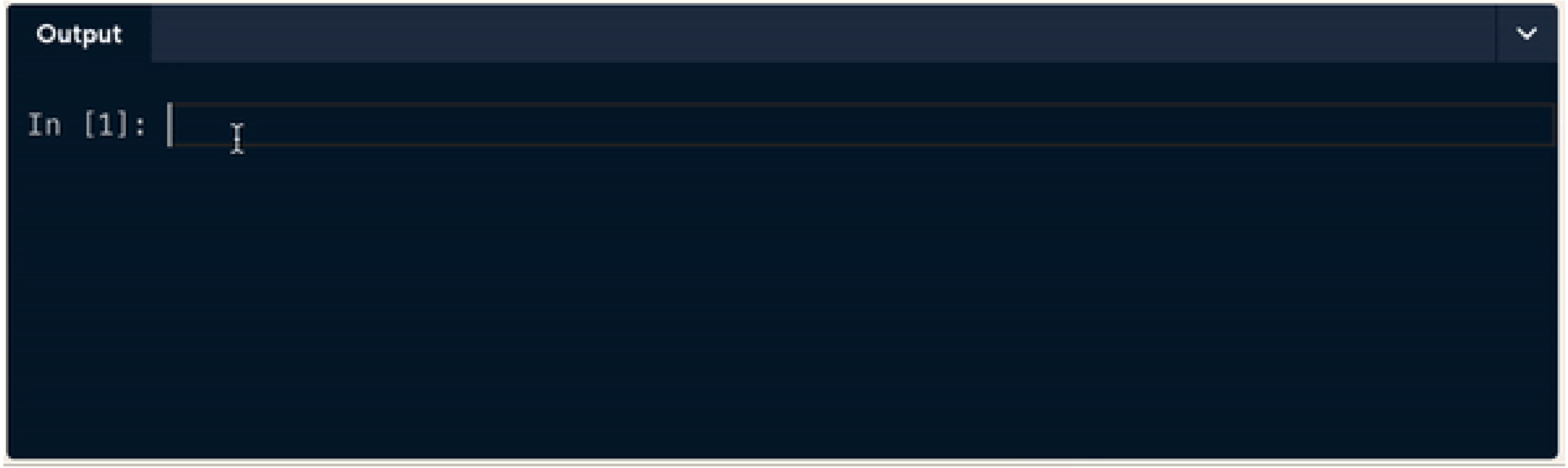
- For beginner programmers
- Basics of Julia
- Working with data in Julia

Installing Julia

- You can download Julia from:
  - https://julialang.org/downloads
- Run Julia online:
  - https://julialang.org/learning/tryjulia

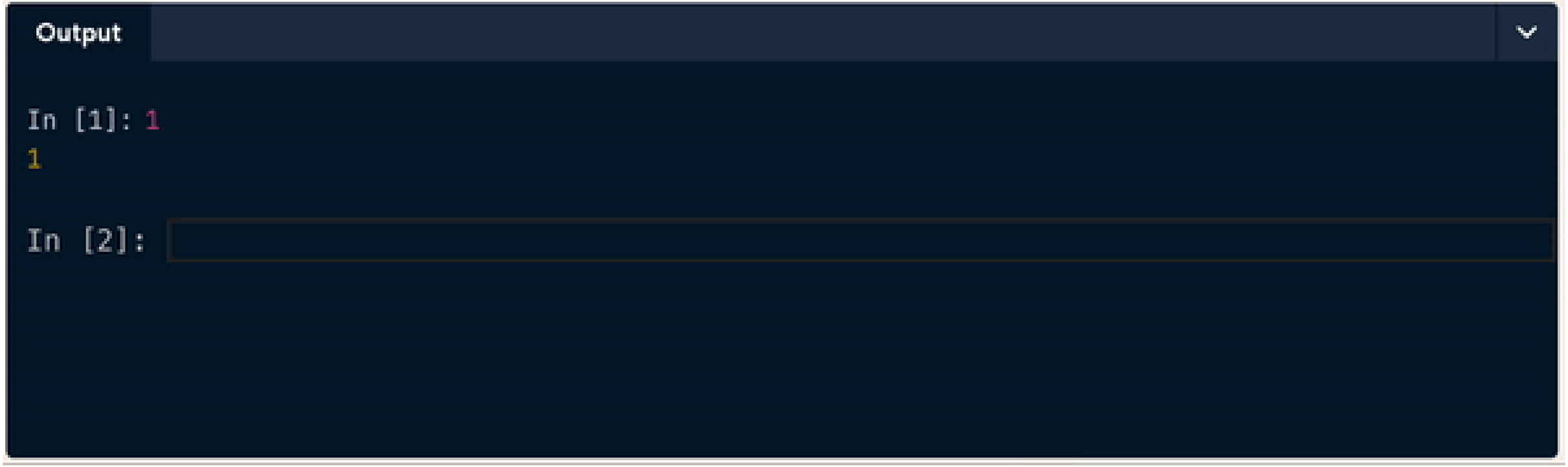
Scripts vs. the console

The console

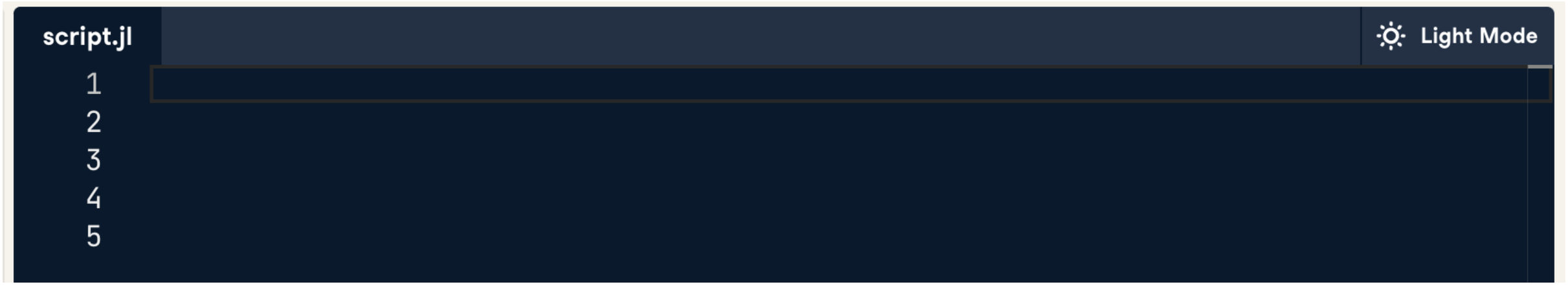


Scripts vs. the console

The console



Scripts vs. the console



Simple calculations and printing

Inside `script.jl`:

```
# Print the number 2
println(2)

# Print the sum of 1+2
println(1+2)
```

```
2
3
```

Comments

Inside `script.jl`:

```
# Print the number 2          <--- these are comments
println(2)

# Print the sum of 1+2        <--- these are comments
println(1+2)
```

```
2
3
```

Comments

Inside `script.jl`:

```
Print the number 2
println(2)

Print the sum of 1+2
println(1+2)
```

```
ERROR: LoadError: syntax: extra token "the" after end of expression
Stacktrace:
 [1] top-level scope
   @ ~/script.jl:1
```

Multi-line comments

Inside `script.jl`:

```
#=



=#
println(2)
```

```
2
```

Multi-line comments

Inside `script.jl`:

```
#=
Print the
number 2
=#
println(2)
```

```
2
```

Let's practice!

INTRODUCTION TO JULIA

Variables

INTRODUCTION TO JULIA

James Fulton
Climate informatics researcher

Assigning variables

```
# Store the number 3 in the variable x
x = 3


# Print the variable
println(x)
```

```
3
```

```
double_x = 2*x
println(double_x)
```

```
6
```

Calculating with variables

```
# Distance run in meters
distance = 5000

# Time taken in minutes
time = 32.3

# Convert distance to miles and time to hours
distance_miles = distance/1609
time_hours = time/60

# Print speed in miles per hour
println(distance_miles/time_hours)
```

```
5.772483341575157
```

Calculating with variables

```
# Distance in meters
distance = 5000

# Time taken in minutes
time = 30.1

# Convert distance to miles and time to hours
distance_miles = distance/1609
time_hours = time/60

# Print speed in miles per hour
println(distance_miles/time_hours)
```

```
6.194392423019188
```

Naming variables

Variable names in previous example:

`distance`, `time`, `distance_miles`, `time_hours`

- Must begin with letter `[a-z]` or `[A-Z]`
- Or other unicode characters e.g. ϕ, β, σ
- Names of variables are in lower case
- Words separated using by underscores `_`
- Underscores can be omitted if still readable
  - e.g. `time_hours` $\rightarrow$ `timehours`
- Can use numbers after first letter
  - `x_times_2` is valid
  - `2_times_x` is not valid

More operations using variables

Operator	Operation	Example	Result
`+`	Add	`1+2`	`3`
`-`	Subtract	`4-2`	`2`
`*`	Multiply	`5*3`	`15`
`/`	Divide	`20/4`	`5`
`^`	Power	`2^3`	`8`

- `2^3` is the same as `2*2*2`

```
speed = (distance/1609)/(time/60)
println(speed)
```

```
5.772483341575157
```

```
speed = distance/1609/time/60
println(speed)
```

```
0.0016034675948819882
```

Let's practice!

INTRODUCTION TO JULIA

Basic data types

INTRODUCTION TO JULIA

James Fulton
Climate informatics researcher

Types of data

Integers are whole numbers

```
# This is stored as an integer
x_int = 1

# Print the type of x_int
println(typeof(x_int))
```

```
Int64
```

Floats are numbers with a decimal point

```
# This is stored as a float
x_float = 1.0

# Print the type of x_float
println(typeof(x_float))
```

```
Float64
```

Other basic data types

Booleans

```
x_bool = true # or false

# Print the type of x_bool
println(typeof(x_bool))
```

```
Bool
```

Strings

```
x_string = "Hello there"

# Print the type of x_string
println(typeof(x_string))
```

```
String
```

- Must use `"..."`
- Cannot use `'...'`

Other basic data types

Characters

```
x_char = 'A'

# Print the type of x_char
println(typeof(x_char))
```

```
Char
```

- Cannot have character `'AB'`
- Only `'A'` or `'B'`, etc.

Consequences of data types

```
# Bank balance as integer
balance = 100

# Interest rate as float
interest_rate = 1.05

new_balance = balance * interest_rate

println(new_balance)
println(typeof(new_balance))
```

```
105.0
Float64
```

```
# Bank balance as string
balance = "$100"

# Interest rate as string
interest_rate = "5%"

new_balance = balance * interest_rate

println(new_balance)
println(typeof(new_balance))
```

```
$1005%
String
```

Consequences of data types

```
# Bank balance as integer
balance = 100

# Interest rate as string
interest_rate = "5%"

new_balance = balance * interest_rate
```

```
ERROR: MethodError:
no method matching *(::Int64, ::String)
```

Converting between types

```
# Convert the integer x to float y
x = 1
y = Float64(x)

println(typeof(y))
```

```
Float64
```

```
# Convert the float x to integer y
x = 1.0
y = Int64(x)

println(typeof(y))
```

```
Int64
```

Converting between types

```
# Convert the float x to integer y
x = 1.01
y = Int64(x)
```

```
ERROR: InexactError: Int64(1.01)
```

```
# Convert the float x to integer y
x = 1.0
y = Int64(x)

println(typeof(y))
```

```
Int64
```

Converting between types

```
# Convert the integer x to string y
x = 1
y = string(x) # y = "1"

println(typeof(y))
```

```
String
```

```
# Convert the string x to integer y
x = "1"
y = parse(Int64, x)

println(typeof(y))
```

```
Int64
```

Converting between types

```
# Convert the integer x to string y
x = 1
y = string(x) # y = "1"

println(typeof(y))
```

```
String
```

```
# Convert the string x to float y
x = "1.0"
y = parse(Float64, x)

println(typeof(y))
```

```
Float64
```

Summary of conversions

- Use `typeof(x)` to find the type of `x`
- Convert between data types using:

From	To	Function
Integer	Float	`Float64(x)`
Float	Integer	`Int64(x)`
Integer or float	String	`string(x)`
String	Float	`parse(Float64, x)`
String	Integer	`parse(Int64, x)`

Let's practice!

INTRODUCTION TO JULIA