



TEAM MEMBERS:

- ANANTHA
NARAYANAN S V (E0119008)
- APARNA S (E0119020)
- KARAN V (E0119039)
- SATHISHKUMAR M (E0119052)
- SIVANT M (E0119004)

FROM

B.TECH CSE (AI & ML)

JULIA LEARNING APP

CSE 220 WEB PROGRAMMING & SCRIPTING
II YEAR V QUARTER

JULIA LEARNING APP

MODULES:

- *Declarative Routing*
- *Julia emulator(Repl)*
- *Conditional Rendering*
- *State Management*
- *Progressive web app (PWA)*
- *Bootstrap 5*

TECHNOLOGIES UTILISED:

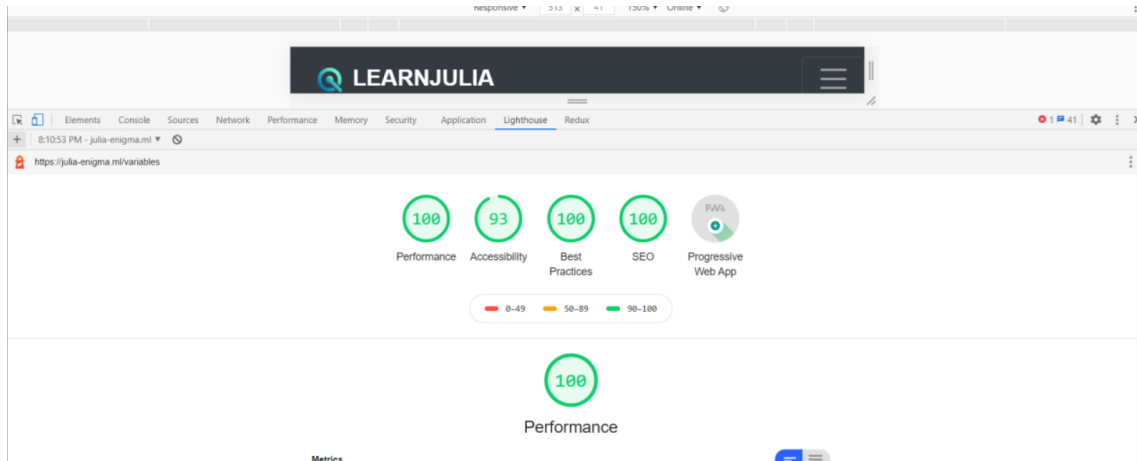
- *Javascript (ES 8)*
- *Vue JS*
- *Vuex*
- *Vue Router*
- *Bootstrap*
- *Firebase*

FEATURES & PERFORMANCE:

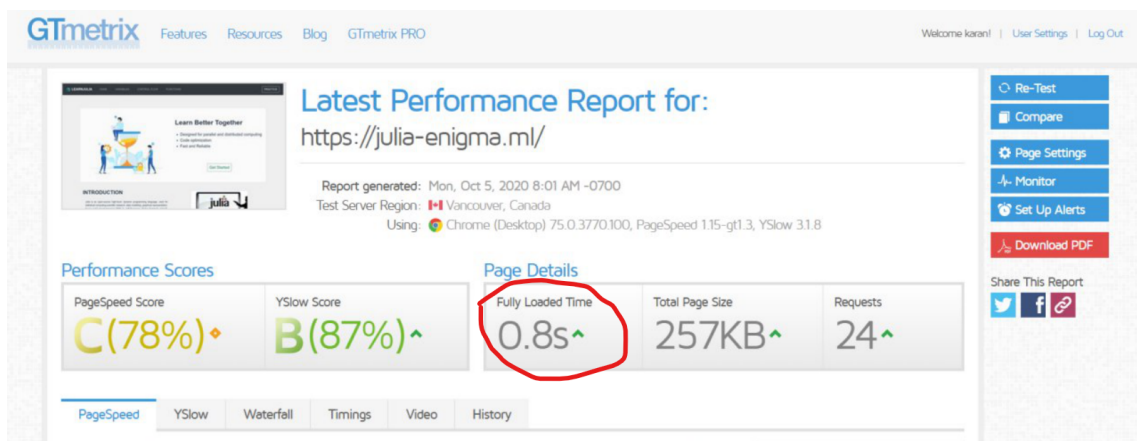
- *Responsive UI design*
- *User friendly*
- *Search engine optimization maxed*
- *Fast load time*
- *PWA offline support*

BENCHMARK

LIGHTHOUSE:



GTMETRIX:



DEMO

BROWSER VIEW

LEARNJULIA

HOME

VARIABLES

CONTROL FLOW

FUNCTIONS

PRACTICE



Learn Better Together

- Designed for parallel and distributed computing
- Code optimization
- Fast and Reliable

[Get Started](#)

INTRODUCTION

Julia is an open-source high-level, dynamic programming language, used for statistical computing, scientific research, data modelling, graphical representation. It was under development in 2009, by Jeff Bezanson, Stefan Karpinski, Viral B. Shah, and Alan Edelman, who set out to create a free language that was both high-level and fast. On February 14, 2012, the team launched a website with a blog post explaining the language's mission. Julia's core is implemented in Julia and C, together with C++ for the LLVM dependency. The LLVM compiler infrastructure project is used as the back end for generation of 64-bit or 32-bit optimized machine code depending on the platform Julia runs on.



INSTALLATION

- Download Julia and run the .exe file
- Proceed with installation and click finish.


[Click here to download](#)

Long-term support (LTS) release: v1.0.5 (Sep 9, 2019)

Checksums for this release are available in both MD5 and SHA256 formats.

Windows (help)	64-bit	32-bit
macOS (help)	64-bit	
Generic Linux on x86 (help)	64-bit (GPG)	32-bit (GPG)
Generic Linux on ARM (help)	64-bit (AArch64) (GPG)	32-bit (ARMv7-a hard float) (GPG)
Generic FreeBSD on x86 (help)	64-bit (GPG)	
Source	Tarball (GPG)	Tarball with dependencies (GPG) GitHub

FUNCTIONS

 **LEARNJULIA**

HOME

VARIABLES

CONTROL FLOW

FUNCTIONS

PRACTICE

FUNCTIONS

User-defined functions

In Julia, a function is an object that maps a tuple of argument values to a return value. Julia functions are not pure mathematical functions, because they can alter and be affected by the global state of the program. Following examples show function definitions and methods to call them.

Built-in functions

collect() - in built function to return an array of items in a collection or iteration specified with some condition.

repeat() - The repeat() is an inbuilt function in julia which is used to construct an array by repeating the specified array elements with the specified number of times.

string functions

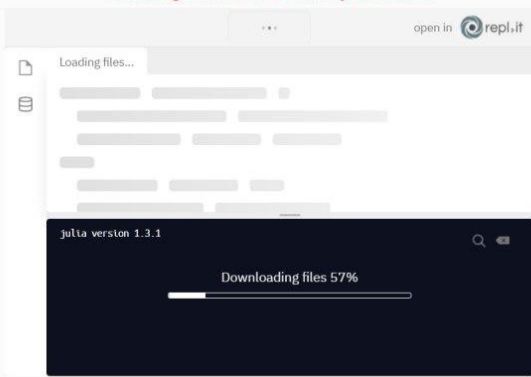
uppercase() - Converts strings to uppercase


lowercase() - converts strings to lowercase

replace() - Replaces substring/string with a new string

NEXT

Click the green button to enable julia terminal



 **LEARNJULIA**

HOME

VARIABLES

CONTROL FLOW

FUNCTIONS

PRACTICE

EXAMPLES OF FUNCTIONS

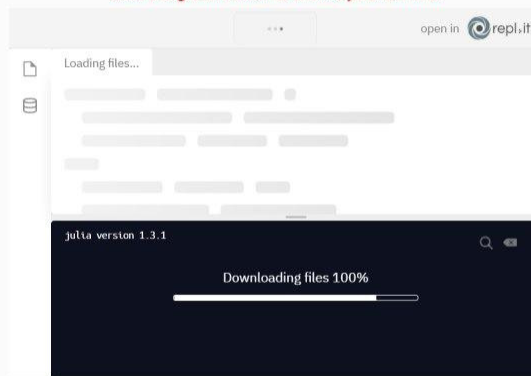
```
function f(x,y)
x + y
end
```

```
f(x,y) = x + y
f (generic function with 1 method)
```


```
s1 = "The quick brown fox jumps over the lazy dog α.β.γ"
s1_caps = uppercase(s1)
s1_lower = lowercase(s1)
println(s1_caps, "\n", s1_lower)
```

Previous

Click the green button to enable julia terminal



VARIABLES

 **LEARNJULIA**

HOMEVARIABLESCONTROL FLOWFUNCTIONS

PRACTICE

SIMPLE EXECUTIONS

- print()** : This message is used to displays a message/statement in the console

```
print("Hello. This is a sample text .")
```
- println()** : This displays message automatically and adds '\n' new line

```
s1 = "The quick brown fox jumps over the lazy dog α.β.γ" println(s1)
```
- printf()** : C language style command used to displays a message/statement in the console

```
using Printf @printf "volume = %0.3f\n" vol #> volume = 113.097
```

Previous

Click the green button to enable julia terminal

open in repl.it


README.md

Julia By Example

© Samuel Colvin 2014, 2015, 2018

Hosted at [juliaexamples.holassaulia/](https://github.com/samuelcolvin/juliaexamples)

julia version 1.3.1

 **LEARNJULIA**

HOMEVARIABLESCONTROL FLOWFUNCTIONS

PRACTICE

INTRODUCTION

Variables are some names given to the memory location to which they are assigned. These memory locations are used to store values that can be accessed by using the name of the location, i.e. Variable. Unlike C and Java, variables in Julia need not to be written with a Datatype. Julia auto-assigns the variable type by analyzing the type of value assigned to it.

RULES FOR LABLING JULIA VARIABLES

- Variable names in Julia must start with an underscore, a letter(A-Z or a-z) or a Unicode character greater than 00A0(nbsp).
- Variable names can also contain digits(0-9) or !, but must not begin with these.
- Operators like (+, ^, etc.) can also be used to name a variable.
- Variable names can also be written as words seperated by underscore, but that is not a good practice and must be avoided unless necessary.

NEXT

Click the green button to enable julia terminal

open in repl.it

Loading files...

julia version 1.3.1

Ready

Connected!

CONDITIONAL STATEMENTS

LEARNJULIA [HOME](#) [VARIABLES](#) [CONTROL FLOW](#) [FUNCTIONS](#) [PRACTICE](#)

IF ELSEIF ELSE

Else if is used when more than one condition is to be checked

```
if 5 > 5
  print("First number is greater than second number")
elseif 5 == 5
  print("Both are equal")
else
  print("First number is smaller than second number")
end
```

Both are equal

[Previous](#) [NEXT](#)

Click the green button to enable julia terminal

open in [repl.it](#)

README.md

Preview

julia version 1.3.1

Ready

[Connected!](#)

LEARNJULIA [HOME](#) [VARIABLES](#) [CONTROL FLOW](#) [FUNCTIONS](#) [PRACTICE](#)

IF ELSE

Conditional evaluation allows portions of code to be evaluated or not evaluated depending on the value of a boolean expression.

```
if 5 == 1
  print("Both are equal")
else
  print("Both are not equal")
end
```

Both are equal

[Previous](#) [NEXT](#)

Click the green button to enable julia terminal


open in [repl.it](#)

Loading files...

julia version 1.3.1

Downloading files 100%

CONTROL FLOW

 LEARNJULIA

HOME

VARIABLES

CONTROL FLOW

FUNCTIONS

PRACTICE

INTRODUCTION

In computer programming, control flow or flow of control is the order function calls, instructions, and statements are executed or evaluated when a program is running. Many programming languages have what are called control flow statements, which determine what section of code is run in a program at any time.

TYPES

CONDITIONAL :

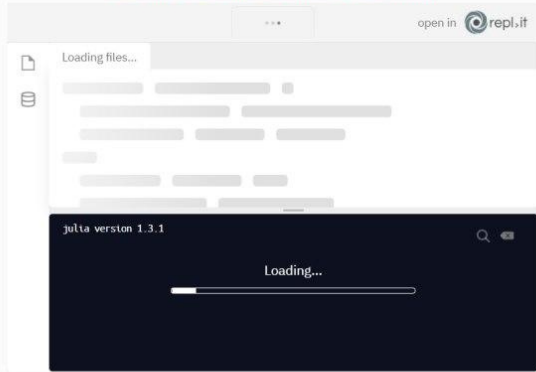
- if-else
- if-elseif-else


LOOPING :

- for
- while

NEXT

Click the green button to enable julia terminal



 LEARNJULIA

HOME

VARIABLES

CONTROL FLOW

FUNCTIONS

PRACTICE


WHILE

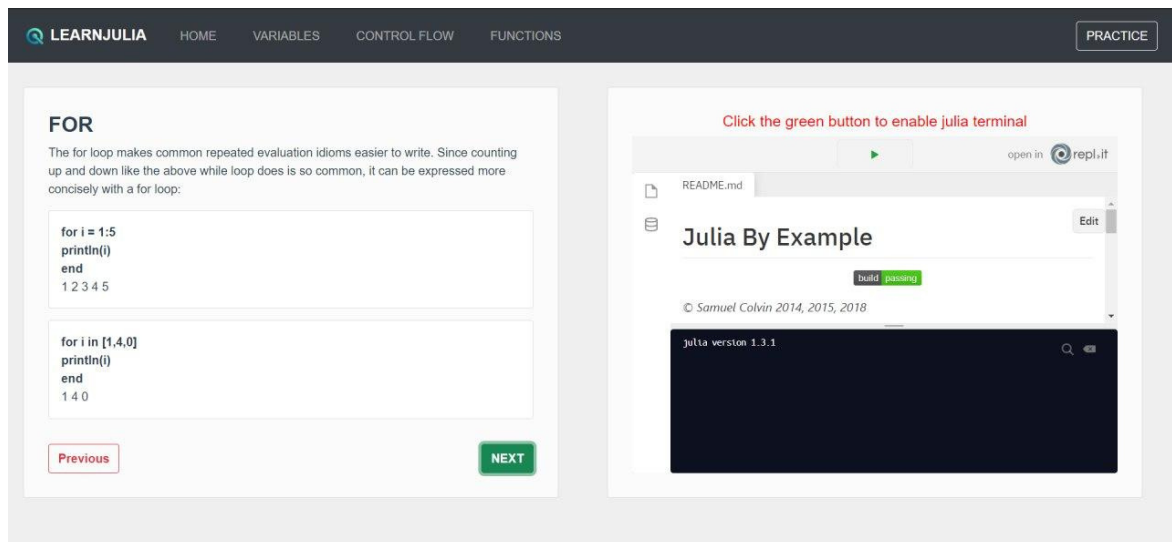
The while loop evaluates the condition expression , and as long it remains true, keeps also evaluating the body of the while loop. If the condition expression is false when the while loop is first reached, the body is never evaluated.

```
i=1
while i<=5
println(i)
end
1 2 3 4 5
```

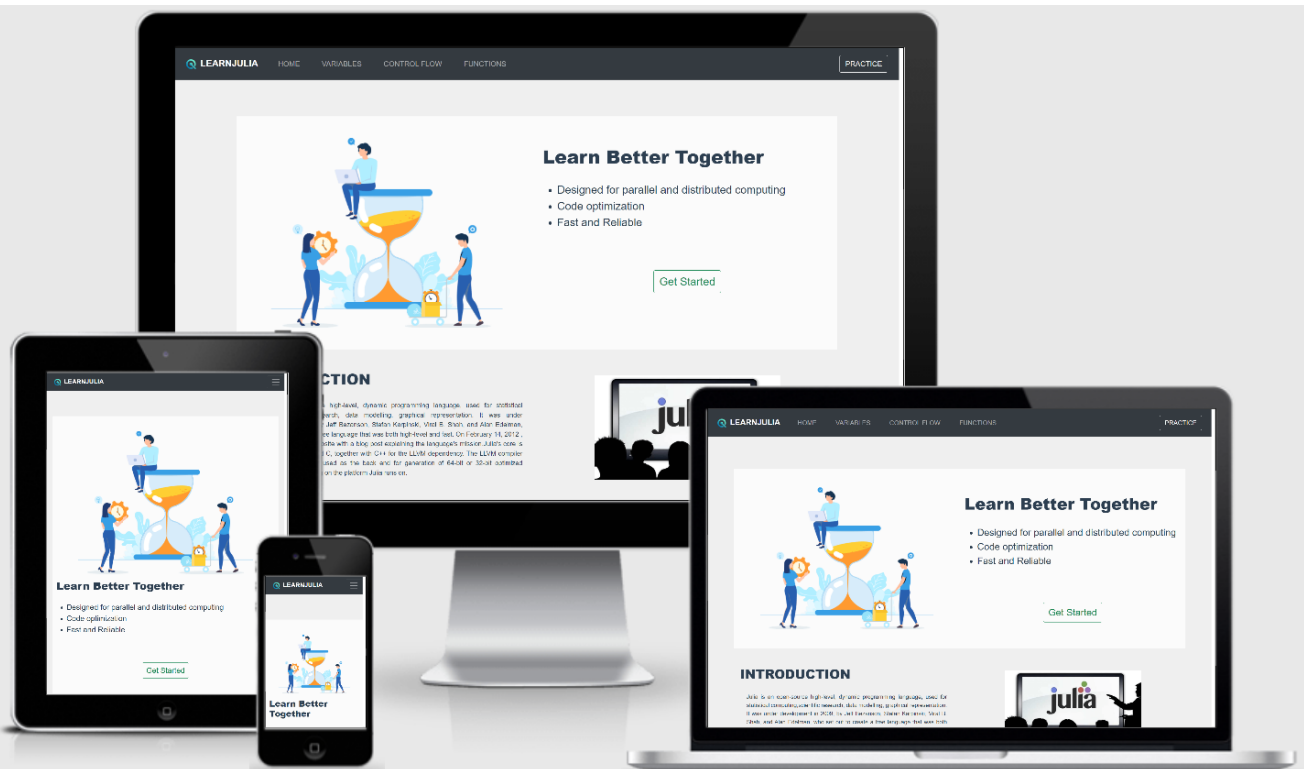
Previous

Click the green button to enable julia terminal





RESPONSIVE LAYOUTS



LEARN JULIA @ julia-engima.ml