This assignment asks you to do a quick preliminary assessment of a programming language. What is it? Who is it for? How is it designed?

Your final project will answer these same questions in more depth, including writing some coding in the language. For this assignment, you'll be doing a very preliminary version of that: you don't have to write any code, and your answers can be short. But it will give you some experience doing an initial evaluation of a programming language.

Step 1: Choose a programming language that is supported on repl.it, except that it should be a somewhat less common one, i.e. not Java or Python. You won't have to stick with the same language you choose here for your final project (but you can if you want).

- Language: Julia

Step 2: See if you can find documentation or a website for the language. Enter some example code into the repl.it if the website has examples. Then based on what you're able to find, answer the following questions.

https://julialang.org/

Examples:

```
for y=1.0:-0.05:-1.0
  for x=-2.0:0.0315:0.5
     abs(mandelbrot(complex(x, y))) < 2 ? print("*") : print(" ")
  end
  println()
end
```

1. Where does the language come from? An individual, group, company, academia, etc.?

- “Julia is an open source project with over 1,000 contributors. It is made available under the MIT license. The source code is available on GitHub.”
- The language originated from a group of programmers who wanted a language with the speed of C and the usability of Python, all while having specific capabilities like linear algebra in MatLab and statistics in R.

- Essentially, they created the language because they wanted all the good parts of other languages in one pretty package.
- It was created by Jeff Bezanson, Stefan Karpinski, Viral B. Shah, and Alan Edelman (why we created julia)

2. Does it seem to be targeted at specific problems or users?

- Julia seems to be a very good option when it comes to mathematical programming. The program allows for the optimization of hand-coded models that normally take days to run and are not particularly data intensive.
- Scientific computing also targets the use of Julia programming language.
- Seems to be most commonly used for data science
- Targets mostly mathematical programming
- Though it has its clear specializations in certain fields, Julia is also a general purpose language
- On the website, they mention that Julia "is about bringing usable,scalable technical computing to a greater audience"
    - It is intended for researchers, scientists, and business owners alike
    - Their goal was to make the language as easy to use and understand as possible

https://discourse.julialang.org/t/designated-target-audience-of-julia-1-0/11804/4

3. What is the language's syntax like? For example, is it similar to C/Java, similar to Python, or something different? You don't have to describe the entirety of the language's syntax, just in general terms what it is most similar to, or how it's unusual if it's not similar to other languages you know.

- The syntax of Julia is similar to that of Python. There are also comparable similarities between Julia and MATLAB.
- Here are some notable differences between Julia and MATLAB:
    - Julia arrays are indexed with square brackets, A[i,j].
    - Julia arrays are not copied when assigned to another variable. After A = B, changing elements of B will modify A as well.
    - Julia values are not copied when passed to a function. If a function modifies an array, the changes will be visible in the caller.
    - The imaginary unit sqrt(-1) is represented in Julia as im, not i or j as in MATLAB.

- Here are some notable differences between Julia and Python:
    - Julia's for, if, while, etc. blocks are terminated by the end keyword. Indentation level is not significant as it is in Python. Unlike Python, Julia has no pass keyword.
    - Strings are denoted by double quotation marks ("text") in Julia (with three double quotation marks for multi-line strings), whereas in Python they can be denoted either by single ('text') or double quotation marks ("text"). Single quotation marks are used for characters in Julia ('c').
    - String concatenation is done with * in Julia, not + like in Python. Analogously, string repetition is done with ^, not *. Implicit string concatenation of string literals like in Python (e.g. 'ab' 'cd' == 'abcd') is not done in Julia.
    - In Julia, indexing of arrays, strings, etc. is 1-based not 0-based.
- “Julia is an open-source language that combines the interactivity and syntax of ‘scripting’ languages, such as Python, Matlab and R, with the speed of ‘compiled’ languages such as Fortran and C.”
- Some basic syntax and operations:
    - println() function is used to print the value/string to the console in Julia
    - The assignment operator used in julia is an equal sign =
    - Commenting operations in Julia are similar to that of Python and R. # pound/hash key is used to write a comment in Julia.
    - " " or """ """ are used to formulate a string in Julia.
    - ' ' is used to formulate only characters in Julia unlike R and Python. It cannot be used to form a string.
    - The $ sign is used to insert existing variables into a string. It brings the functionality of f-strings in python.

4. Does the language emphasize a certain kind of programming language semantics? For example, is it object-oriented, functional, imperative, or something else?

- Julia is a fast programming language designed for high performance. It is dynamically typed with a good support for interactive use.
- Was built under a reproducible environment meaning that the same Julia environment can be reproduced every time across different platforms using pre-built binaries.
- Julia uses multiple dispatch making it easier to express object-oriented and functional programming patterns

    - In Julia, functions are not pure mathematical functions as they can be altered and affected by the global state of the program.
    - Short, simple function definitions are common in Julia
- Julia function arguments follow a “pass-by-sharing” convention, meaning that the values are not copied when they are passed to functions.
    - Function arguments work as new variables whose location can refer to a value. The values referred to are identical to the passed values.