# Research Questions

## Overview Questions

- What are the strengths and weaknesses of your language?
    1. Strengths
        - Julia is fast and high-level. It is one of the fastest programming languages for interactive computing out there.
        - Julia is easy to learn because Julia has the pretty much the same syntax structure as Python, which is very user-friendly. Also, in the data type declaration, I think Julia is more organized than Python because it's well-detailed and follows a good structure for users to have a better debugger. Additionally, Julia can be deployed in the same way as Python in Vscode, Jupiter Notebook for data science.
        - Julia is extensible. There are multiple levels to this extensibility:
            + Julia supports a remarkable degree of interoperability between unrelated libraries and codebases.
            + Julia has the best language libraries for differential equations, mathematical optimization, and automatic differentiation. It has allowed the creation of advanced automatic differentiation tools that can generate novel GPU kernels for ML problems at runtime, in a Jupyter notebook, for example.
    2. Cons
        - Although Julia has the basics inspired by Python, it's still much younger and needs time to develop.
        - Slow loading to run after finishing coding at the beginning. Usually have to wait for about 8s to run simple "Hello World," but it's the same time to run complex code. Then it compiles very fast, the same as C
        - Still doesn't make the most use of a variety of libraries because some may overlap others
        - Deploying graphs take more procedure than Python in VsCode and Jupiter Notebook
        - Not so good-looking graphs

    - Download Julia libraries takes too much time and computer memories comparing to Python and R

- For what type(s) of problems/domains is it particularly well suited?
    - Julia is mainly used for data science and machine learning, but it's still a general-purpose language like Python.

- Other than functional, what other paradigm(s) would the language fall under?
    - Julia uses Dispatch as a paradigm.

## Language Properties

- Are there restrictions on mutation?
    - No, there's no restriction on mutation on Julia. Like Python, Julia's value can be assigned easily within variables using =
        + Assignment changes in which object a name refers to x = ex cause the name x to refer to the value resulting from the evaluation of the expression ex. Assignment never changes the values of any objects.
        + The mutation changes the value of an object: x[i] = ex and x.f = ex both mutate the object referred to by x, changing a value at index or a property with a name, respectively. Mutation never changes what objects any names in any scope refer to.

- Are functions first class?
    - Yes, they are. 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. Without parentheses, the expression f  as functions refers to the function object and can be passed around like any other value.

- What sort of control flow constructs does the language have?
    - Compound Expressions: begin and ;
    - Conditional Evaluation: if-elseif-else and ?: (ternary operator).
    - Short-Circuit Evaluation: logical operators && ("and") and || ("or"), and also chained comparisons.
    - Repeated Evaluation: Loops: while and for.
    - Exception Handling: try-catch, error, and throw.
    - Tasks (aka Coroutines): yieldto.

- Is the language statically or dynamically typed?
    - In my opinion, Julia is dynamically typed. Because in dynamic languages, there are no rules for assigning types to expressions: types are implied by how data flows through the program *as it executes*. In general, expressions can potentially produce values of any type at all. From coding Julia, Julia is definitely dynamic. Because I applied my Python coding mindset to Julia, it works similarly. There are no restrictions or rules for assigning values. And the running result is determined based on the values, not the expression.

- Is there support for type interference?
    - Yes, it is because Julia uses type inference to determine the types of program variables and generate fast optimized code.

- What core abstractions or features of the language make it unique? Define and explain those features, especially if they are concepts not also present in the languages we've discussed in class.
    - Julia's performance is comparable to C/C++ while being dynamic. it's measured on the different benchmarks available on JuliaLang's homepage and simultaneously provides an environment that can be used effectively for prototyping, like Python. Julia can achieve such performance because of its design and Low-Level Virtual Machine (LLVM)-based just-in-time (JIT) compiler. These enable it to approach the performance of C. Therefore, user-defined types in Julia are as fast and compact as built-ins
    - it's flexible and easy to be used with other programming languages. The Pycall package enables Julia to call Python functions in its code and MATLAB packages using the MATLAB.jl package. Functions and libraries written in C can also be called directly without needing APIs or wrappers.

- What is the difficulty of learning to use this language?
    - From my personal experience, I think the assigning values in Julia is not well-bounded compared to Python or C because it's a little bit freestyle (i don't know for sure)
    - Setting up Julia's environment is challenging in VScode and Jupiter Notebook
    - Libraries usage is not as comfortable as Python and R
    - Documentation is just unclear. It has to do many searches online from multiple sources to get the answer or the syntax.
    -

- What form does the programmer interaction take (compiled/interpreter, virtual machine, interaction shell, work from files, etc.)?
    - Julia uses a just-in-time (JIT) compiler that is referred to as "just-ahead-of-time" (JAOT) in the Julia community, as Julia compiles all code (by default) to machine code before running it. When running code, Julia comes with a full-featured interactive command-line REPL (read-eval-print loop) built into the Julia executable.

## Taxonomy/Classification

- What is the history of the language? When was it developed, and by whom?
    - Work on Julia was started in 2009 by Jeff Bezanson, Stefan Karpinski, Viral B. Shah, and Alan Edelman, who set out to create an accessible language that was both high-level and fast. On 14 February 2012, the team launched a website with a blog post explaining the language's mission. In an interview with Info World in April 2012, Karpinski said of the name "Julia": "There's no good reason. It just seemed like a pretty name. Bezanson said he chose the name on the recommendation of a friend.

- What other programming languages is the language related to, if any?
    - Julia is related to Python, C, and R. its primary purpose is to run as fast as C, is easy to study, and have many libraries access as Python, and is designed with mathematical specifications the same as R

- What uses does the language have today (and if it is not used today, why not)?
    - Its primary usage is for data science and machine learning. I think it's not popular in the data science field because it's just too new, and the optimization for data science tools is just not good as Python. Additionally, the charts in Julia are not as beautiful as Python and R