Overview. The goal of this (individual) project is to dive deeper into language design and implementation by developing a significant extension to MyPL. The project will be graded on: (1) the ***quality of your work***; (2) the ***completeness of your work***; (3) the ***quality of your tests and documentation***; and (4) the ***difficulty of your project***. The project is worth 90 points and includes a proposal, a project specification, a final report, and a demo presentation (video). Please be sure to ask if you have questions concerning the project requirements or expectations.

Step 1 (due by Tues, March 7th). The first step is to identify *two* extensions you are interested in exploring.[1] The following is a non-exhaustive list of project ideas (by general difficulty level). You are not required to select one of the following—each is just an idea of a possible project that you may be interested in or may help give your ideas in terms of picking your own direction. You might also look at existing program languages for inspiration.

Significantly Difficult: These projects are more involved and generally require additional thinking about design decisions as well as some independent research into how each would work. Note these would score significantly higher on the "level of difficulty" project criteria.

- Module/import system (e.g., like "import" in Python and Java)
- Concurrency/multi-threading support
- Compilation to another interpreter (e.g., JVM or CPython bytecode, WASM, LLVM IR)
- A completely separate language syntax design that "compiles" to our VM
- Compilation to assembly language (e.g., to ARM or x86)

More Challenging: These projects are less challenging than those above, but still require thought about design issues and/or involve some implementation challenges. These projects would generally score on the higher end of the "level of difficulty" evaluation criteria.

- Classes (with public/private member variables and methods)
- Pointers and/or pass by reference
- Function overloading
- Operator overloading
- Struct type subclassing (inheritance)
- A garbage collector (e.g., mark and sweep)

[1] Because each student in the class will need to work on a different extension, you must provide a first and second choice. Assuming neither of your choices have already been assigned, you will receive your first choice, followed by your second choice (if the first choice is taken), or else be notified to select an additional alternative.

- Performance optimizations (pick a few optimizations, implement and benchmark them)
- Higher-order functions and closures
- New syntax for additional collection types (e.g., dictionaries, lists, data frames)
- Transcompilation to another language (e.g., C, C++, or Python)
- Exception handling (e.g., try-catch blocks)
- Delete operation

Less Challenging: These projects are more straightforward and are generally easier to implement than those above. These projects would generally score on the mid-to-low end of the "level of difficulty" evaluation criteria.

- New but "common" PL constructs (e.g., switch statements, for each loops)
- Constant variables and function args
- Proper handling of associativity and precedence
- Collections of new operations and operators
- File I/O
- New built-in functions

Tooling Projects: These projects (which are also acceptable), some of which are more challenging, involve adding "tool" support to MyPL as opposed to language extensions.

- A debugger (e.g., with checkpoints, stepping capability, variable inspection)
- Save and read MyPL programs in our IR (to "compile" to a file, then execute later)
- A fully functional REPL (read-eval print loop) for MyPL
- A memory or execution profiler (execution monitoring/statistics)
- A suite of useful programs written in MyPL (may require additional built-in functions, e.g., for file I/O)

Again, note that you are free to choose something different than what is listed above.

Submitting your Choices. You must submit your first and second choice preferences in class on or before the due date. For your first and second choices, you must state each extension and provide a short description of what you plan to tackle as part of the extension (the scope). This description should contain a list of the "features" of the extension (e.g., what a user would be able to do, what

capabilities will be added to MyPL, etc.). Note that these are just your initial thoughts to help me understand what you are thinking in terms of the project.

Step 2 (due by Tues, April 11th). For this step, you must submit a specification (plan of attack) for your extension. The details of the specification will differ depending on your project. At a minimum, the specification should include enough information to make it clear what the scope of the extension is and how it will work. You should think of the specification as a write up that we can use later to evaluate whether or not you completed your work. If you are doing a language extension, e.g., you would want to include the grammar for the new language constructs, basic examples of how the construct would work, and initial test cases you will use for testing. You can also think of the specification as your "homework assignment" for the project—give enough detail to make it clear what you are implementing. Submit a typed specification in class on or before the due date.

Step 3 (due by the end of Thursday of finals week.) Your project must be completed on or before the Thursday of finals week. Note that a special "Project" repository will made available for you via GitHub classroom to turn in your work. All of your source code must be pushed to your repository as well as your final project report. Your project report must be typed (preferably 11pt font with 1-inch margins), should be around 2–3 pages, and include:

- Your name, the course, and the date
- A description of the extension with examples
- A description of the parts of the MyPL pipeline that you modified and at a high level what the modifications were
- What was successfully completed, what if anything wasn't completed (compared to your specification in Step 2), and what you would do next if you had additional time
- How you tested your extensions including the new tests you developed
- Instructions for building and running your extension

Finally, you must also create a 3–5 minute demo video (screencast) of your extension. The demo must be "narrated" (an explanation of what you are demonstrating) and (a) show how the extension would be used in a "real-world" setting; (b) give a brief walk-through of the (non unit) tests you developed; and (c) show the extension working over the tests. Note that you should develop unit tests as well, which should be included as part of your source code. Your demo video must be uploaded to YouTube and a link to the video must be provided in your final report.