**Semester Project**

## **Due Dates**

Team selection: Sept 30th Language selection: Oct 7th

Code, checklist, “worksheet,” watch others’ pitch videos: End of final exam time

## Purpose

The purpose of this project is to learn a new programming language independently. There are thousands of programming languages, and one aspect of your future careers will be learning new languages and identifying what is good and bad about them. To do this, you will be put into three to five person groups and choose a language to “pitch” to the class to solve a “problem” of your choosing. You also will need to describe the language in terms of several aspects of PL theory. Therefore, you will need to choose a “complete” language rather than some of the more esoteric languages that can do very little. The questions are *very* general since they must work for everything from MIPS to Java. **Only two teams may choose the same language**. There is also a *required rubric checklist*!

Submission will include a presentation of your “pitch” to the class, the sample code you have written, and the answers about the language regarding its theory. Each time a new part of theory is discussed in class, additional questions involving theory can be answered. ***I highly suggest waiting until these topics are discussed to answer the related questions.***

Since the purpose is to learn a new language, there are several “forbidden languages” that you may have used before or are too similar. If I learn you have used the language in an internship before, I will require you to restart. If you have learned the language as a hobby, you may still use it.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Possible Languages (not inclusive)** | |  | **Forbidden Languages** | **Why** |
| Ruby | F# |  | C | Too close to C++ |
| LISP | Lua |  | C# | Too close to Java AND C++ |
| Prolog | Fortran |  | R | Used in Statistics |
| Pascal | Perl |  | MatLab | Used in too many courses here |
| Swift | Objective-C |  | PHP | Too common in internships |
| Visual Basic | Go |  | Kotlin | It’s a mix of Java and Scala |
| Haskel | Rust |  | Assembly (any) | Used at SDSMT in CSC |
| SmallTalk | TypeScript |  | SQL | Used at SDSMT in CSC |
| Clojure | Elixir |  | C++ | Used at SDSMT in CSC |
| Erlang | Dart |  | Java | Used at SDSMT in CSC |
| BASH | ADA |  | Python | Used at SDSMT in CSC |
| Cobol |  |  | Javascript | Used at SDSMT in CSC |
| More (if not on the forbidden list): <https://www.tutorialspoint.com/codingground.htm> | |  | Android | Android is a flavor of Java |
|  | HTML5 | Used at SDSMT in CSC |

# Team Ground Rules

If you feel there are issues with a partner, I need to be notified in sufficient time to attempt a resolution to allow changes in final grades. This is 10 days before the due date.

There is an **additional coding convention** where a team must put down who wrote which function/class. Unfortunately, this is a check in case I get complaints of a “disappearing” teammate.

Traditionally, the standard time for email response is 24 hours during the work week (excluding holidays and weekends). This is my guidance on confirming a missing teammate.

# Checkpoints

1. Send me the names for your team by the team checkpoint due date *via email or on paper in class*. If I do not hear from you by then, I will choose your teammates.
   1. I reserve the right to add a student to your team.
2. Tell me your language choice by the language checkpoint *via D2L* for record keeping, and current choice of “problem.” I will keep a running list of languages on the course homepage **as there may only be one repeat language**. If not selected on time, I will choose for you, and you will lose 10 points.

You may continue to change the problem to the end of the semester. However, since you must make code for your problem, I want the option to be able to provide early feedback.

# Main Project Tasks

1. Select a “problem”
2. Create sample code
   1. “solution to problem” code for presentation
   2. “Hello world” like code sample showing basic usage
3. Present your “pitch”
4. Answer the theory questions in relation to your language

***Notice:*** If you keep up with the tasks, this project will be very light during “heck week.” If you wait, you will hate your PL life during “heck week.” You are adults, so you can set your own schedule**. You have been *warned*.**

## Select a “Problem”

Select a “problem” that will show the power of your language. This will provide the base for your “pitch,” which will explain why your “team” should take the time to move to a new language compared to staying with C++, Java, Python, and/or Scala.

**NOTE:** If you decide later that the language is “a bad idea,” you maychange the pitch to be in the context of "our team wants to change to this language, let's convince them otherwise."

## Project Code

You must make a project. Since you will be showing your code running in the presentation, you must provide code. Your code should, at minimum, show why the language is good for the problem and should be equivalent in size to the first assignment of this class. Specifically, it should have

* A **minimum** of 2 code **files/classes/modules** per person (or equivalent) that meet the ideas of the SOLID principles (no rounding. If the language supports generated files or you use another language to present graphically (or similar), you may use them, but I only count files *your team* personally wrote).
  + An example **violation** would be having 4 classes in 1 file.
  + Another example violation is functional decomposition (AKA splitting up what should be one class/file into multiple files just to bump up the file count)
  + If you think you do not need these, talk to me about signing a waiver on this item.
    - If you do this embedded in some fashion, I only count your personally constructed languages files.
* A **minimum** of 1 distinct piece of functionality per person (no rounding).
* Each piece of functionality must have an **average** of 2 functions (or equivalent) per person.

You must also provide a “hello world” file demonstrating the basic use of the language **with some input, output, branching, and iteration/recursion at minimum.** You are welcome to do more (e.g. like <https://learnxinyminutes.com/>). I’m also OK if you put the minimum requirements at the top, and use the rest like “scratch paper” to test things.

Please refer to the submission instructions for how to structure your code when zipping the files.

**Tip:** This is a good place to write up the sample code required by some of the theory questions.

## The Pitch

Your “pitch” will be presentation during finals week. Attendance is mandatory. The purpose of doing a “pitch” is to give you training in the future for when you are trying to convince your colleagues to take the effort to learn new language. After doing so, decide on a problem where your languages will do better. Your “pitch” must be presented in the context of providing a better option than C++, Java, Python, and/or Scala **for your problem**. You pitch must include

* 1. Language version (there should be a number or year here)
  2. Language philosophy
  3. Why should it be used (or not) based on the metrics (e.g. orthogonality) given in class
  4. Where it is commonly used (or would like to see it used)
  5. Basic usage of the language [[1]](#footnote-1)
  6. Your selected “problem”
  7. Why it should be used to solve **your problem**
  8. A list of ***all*** the features you implemented
  9. Code sample **for your problem,** ***running,*** that shows all pieces of functionality
  10. Within time (length is TBD)

You will need to submit any, and all, additional materials developed for the pitch. I do not have restrictions on presentation style other than that it includes the above and is in the “context” given. Using PowerPoint, whiteboard, skit, etc. as course materials are all acceptable. I usually video tape the presentations for grading purposes.

## **Theory Questions/Worksheet**

There is a worksheet posted for the theory component of this project. It must be put into a PDF with your names as the title (what shows at the top tab when viewed), and the PDF file should be named “lastName1 lastName2 …” Your total document will likely be 10-20 pages. The sections are based on the theory that will be discussed in class. I ***strongly advise*** completing them immediately after that theory lecture set in class. The worksheet ***must* follow the format** given with default margins and 12 point Times New Roman font or similar. When code examples are given, they must be in the report, not references to sample code. I suggest copying and pasting to start. General tip: Do not get stuck on specific language names. Terms change *a lot* in language design.

# PL’s “Final”

This project is the “final” for PL. This includes the presentations. All of the project, presentations, theory, code, etc. is due at the end of our final exam time. Bonus points for early submission of materials still apply.

# FAQs

* What do you mean by distinct piece of functionality?
  + A task that can be coded and tested near independently of other tasks. Think of the tiers in the other grading documents. Each tier is generally on distinct pieces of functionality.
* How much do you want for the “Basic usage of the language” in the pitch?
  + This is flexible. Compilation, structure, your hello world file walkthrough, etc. all work. A short program works. A *single example* input, if, loop, and function work. The purpose of this is to answer “how do I get started?” with a mix of “what does the language look like?” Usually, if you have more than what would fit on 3+ slides, it is too much!
* Do I have to do *all* of the assignment syntax in the EBNF question?
  + No, the purpose of this is to establish the basic syntax format for a line of code. A basic in x = <some math>; is sufficient. It only has to be a *complete* statement.

# Submission

1. Download the rubric checklist, and check off the completed elements (check mark, strikethrough, etc. all work). Put your team’s names at the top and save as a PDF named, your team name (if applicable).
2. Check the coding conventions before submission.
3. In ONE zip file, place
   * Your PDF for the theory questions at the root
   * Your PDF for the checklist at the root
   * Your video, and any additional “pitch” materials (if any) in a folder named “Pitch materials”
   * Your project code, **without compiled files,** in a folder named: Project Code
   * Your hello world inside a folder named “Basic example”
4. Submit to D2L. The dropbox will be under the associated topic's content page.
5. *Check* that your submission uploaded properly. No receipt, no submission.

If there are concerns about equal work in the project, please contact me early enough to attempt a resolution. If that fails, please put down your names on who did what.

**Checklist of submission structure requirements**

* Theory questions in one PDF in for the worksheet format given
* Checklist PDF is at the root of the zip file
* PDFs are named appropriately
* Additional pitch materials in a folder named “Pitch materials”
* Problem code in a folder named “Sample Code” and NOT zipped within a zip folder
* No compiled files
* Hello World code in a folder named “Basic example”

# Rubric

Additional requirements:

* Not submitting your language on time will be -5 points
* Going over/under time length will be docked -10 points, each.
* Not following the worksheet format will result in up to -10 points depending on the severity.

These are on top of other deductions. This will be almost checklist type “grading.” This rubric is all or nothing for each line item, unless otherwise specified.

|  |  |
| --- | --- |
| **Item** | **Points** |
| **Proposal** | **10** |
| **PL theory** | **74** |
| **Comparison to C++, Java, Python, and/or Scala** | **26** |
| Philosophy | **3** |
| Location it is used | **3** |
| Where it excels, where it fails (4 each) | **8** |
| Portability, simplicity, orthogonality, reliability (3 each) | **12** |
| **Syntax, OOP** | **16** |
| Q1 | **4** |
| Q2 | **4** |
| Q3 | **4** |
| Q4 | **4** |
| **Parsing, binding, type system, and data type range** | **16** |
| Q1 | **4** |
| Q2 | **4** |
| Q3 | **4** |
| Q4 | **4** |
| **Function/OOP/specialties** | **16** |
| Q1 | **4** |
| Q2 | **4** |
| Q3a | **4** |
| Q3b | **4** |
| **Sample code** | **75** |
| Code # of files that meet the ideas of the SOLID principles | **20** |
| Required # of distinct pieces of functionality | **20** |
| Required # of functions | **20** |
| Basic usage code (-2 for minor issue) | **15** |
| **Pitch** | **26** |
| Language version | **2** |
| Philosophy | **3** |
| Why should it be used | **3** |
| Where it is commonly use (or would like to see it used) | **2** |
| Basic usage of the language | **3** |
| The problem | **2** |
| Why it should or should not be used to solve the problem | **3** |
| List of ALL features implemented | **2** |
| Code **for your “problem,”** ***running,*** with all pieces of functionality | **6** |
| Final exam attendance | **15** |
| **Total** | **200** |

1. It is *extremely* tempting to try to teach the entire language here. That is not the purpose and will make you go over time. Tip: if parts a-e take more than 4 minutes, it is longer than necessary. I’ve seen these parts complete in 2:00! 3:00 is more “normal.” [↑](#footnote-ref-1)