Setup

- Install Go
- Install Visual Studio Code
- Install Go language extensions for Visual Studio Code

General Guidelines

- Code must be properly indented as per the indentation format shared with you. All Go code must be indented as per the Go community standards for indentation, which may or may not match with other languages. (Your VS Code editor will help, but you must read, watch YouTube videos, and find out what is the community standard indentation.)
- Code must use meaningful variable names.
- Code must produce output strictly in the expected format wherever such format is provided.
- Unless otherwise specified, always print floating point numbers with two decimal points.
- Add appropriate code for exception handling and print a meaningful exception message in case of an exception.
- Code for all exercises must be pushed within the “basic-go” folder under the branch name allocated to you on a Git repository. Branch name must be Participant’s Merce username (e.g. “indulekhas” for Indulekha Singh) You **must** be able to retrieve the complete code for any assignment later – do not overwrite one assignments’s source code to create the next one. When the assignment spec says “Extend 7 to create 8”, you must make a copy of the source code of 7 to modify it and create 8.

Assignments

1. Write a program to print “Hello World” on the standard output. **[“main” function, 1 hour]**
2. Write a program to prompt the user to enter a name, and print “Hello <name>”. Replace “<name>” with the text entered by the user. **[Scan function, 2 hours]**
3. Write a program to accept “<name>” as command line argument, and print “Hello <name>”. Replace “<name>” with the text entered by a user running the program. **[command line argument, 2 hours]**
4. Modify the above program to take two command-line parameters. The first is the string for the message template, like “Hello <name>” or any other template. The second is the actual name to print in the message by replacing “<name>” with the given name. **[String operation, 4 hours]**

   A template is a string which has a placeholder. For instance, **`Hello, <name>`** is a template. It is a string which has a placeholder. In this case, **`<name>`** is a placeholder. A placeholder is replaced with an actual value before the string is used for a message. In this case, the template **`Hello <name>`** will be processed in your code, the placeholder **`<name>`** will be replaced with the actual name like **`Bharat`** or **`Vasavi`**, then the final result, which is **`Hello Vasavi`** will be printed.

   Your code must be able to work correctly if there are spaces in the template and in the name.

It is not necessary that placeholders are marked with the characters < and >. You can use your own characters to mark the start and end of the placeholder. The following are all valid templates, provided that your code is written to process the template format you have chosen:

- **`Hello {name}`**
- **`Hi there [name]`**
- **`Tu kaisa hai $name$`**

5. Write a program to prompt the user for 2 inputs: “num1” and “num2” and generate a sum of two numbers as output. The program must accept only whole numbers (positive or negative) or throw an error. The output shall be “num1=<num1> num2=<num2> sum=<result>” where “<num1>”, “<num2>” and “<result>” will be replaced with actual value. **[Basic expressions, 2 hours]**
6. Write a program similar to (4) above, but accept floating point numbers. **[Data types, 2 hours]**
7. Extend program (5) to accept 3 inputs: “num1”, “num2” and “operation” where operation could be “+”, “-”, “*” or “/” to represent sum, difference, multiplication or division. The output will be output of “num1” <operation> “num2”. The output shall be “num1=<num1> num2=<num2> <operation>=<result>” where “<operation>” will be replaced by the operation name. Use “sum”, “difference”, “multiply” and “divide” as an operation name when printing the result. **[If/then/else OR switch statement, 3 hours]**
8. Write a program to prompt for a sequence of numbers, one number at a time, till the user enters “proceed”. Upon receiving “proceed”, the program shall calculate the sum of all numbers and produce an output. Ensure that only valid numbers are considered as an input; if you detect an invalid number input, give a meaningful error message and exit. Your program must work correctly even if the user enters zero or just one number. **[Data types and validations & for-loop, 4 hours]**
9. Extend (7) to accept statistical operation instead of “proceed”. Valid values for “<operation>” are **`count`** (to count number of valid numbers), **`mean`** to calculate arithmetic mean, **`min`** to calculate minimum value (minimum of all numbers input), **`max`** for maximum value (maximum of all numbers input). **[Switch & functions, 4 hours]**
10. Extend (8) to support “sort” operation. Use an in-built function call for sorting numbers. **[Core Classes & array operations, 6 hours]**
11. Extend (9) to support “countodd” and “counteven” operations to respectively print number of times odd numbers and number of even numbers found within the list. **[Expressions, 2 hours]**
12. Write a program to prompt for two whole positive numbers -- “num1” and “num2”. Print multiplication table for the number e.g. for num1=3 and num2=20, output will be “3 * 1 = 3\n3 * 2 = 6\n … \n3 * 20 = 60”. **[For loop, 6 hours]**
13. Write a program to prompt for three inputs: character to be used for display, num1 to represent number of rows and num2 to represent number of columns. The output will be a rectangular matrix where each cell will print a character input as a first input value. **[Loops, 4 hours]**
14. Write a program to print current date/time in following formats (one line per format) in UTC time e.g. “16 Mar 2022” “Mar 16, 2022” “2022-03-16” “2022-03-16T15:52:00Z” “Tuesday, 16 March 2022” **[Date manipulation, 3 hours]**
15. Extend (13) to print time in IST timezone. **[Date manipulation, 2 hours]**
16. Extend (14) to print supported timezones, and accept a valid timezone as input and print time as per the time zone selected by an end user. **[Date manipulation & switch statement, 3 hours]**

17. Write a program to accept two dates (any of the formats supported in the earlier problem) and print a difference in human readable format e.g. “1 year 2 day 32 minutes”. **[Date manipulation, 3 hours]**
18. Write a program to accept a date and print whether the date falls in a leap year. Accept a date in any format supported in one of the previous problems. **[Date manipulation, 2 hours]**
19. Write a program to accept two dates (any of the supported period) and print an output whether date1 and date2 are equal, date1 is earlier than date2 or date1 is later than date2. **[Date comparison, 1 hours]**
20. Write a program to accept two dates and print the count of week-end days (Consider Saturdays and Sundays as week-ends). **[Loops & Date manipulation & simple expressions, 6 hours]**
21. Write a program to accept a list of holidays (date in any of the supported formats). Store this list in an internal array. After the user confirms entering of the holiday list, accept a date from the user, and confirm whether it's a working day or not. (All Saturdays and Sundays are implicitly considered as holidays). **[Arrays & Date Manipulation, 4 hours]**
22. Same as (21) to accept a list of holidays, and then prompt a user for two dates (in the supported format) and print the number of working days between two dates. Consider both dates during the calculation. **[Date manipulation & loop & boolean expressions, 4 hours]**
23. Same as (22) to accept a list of holidays, and then prompt a user for two inputs: input date as a first argument and a number of business days as a second argument. Number of business days will be a positive or negative whole number. The output shall be the date relative to an input date +/- the number of business days. Holidays must be excluded while calculating the output date. **[Date manipulation & loop & boolean expressions, 4 hours]**
24. Write a program to take the names of candidates as input. Prompt user to keep entering new names till user enters “done”. Once a list of names are accepted, prompt the user for a name. Output shall be “<name> exists” or “<name> does not exist”. A name exists if the name exactly matches one of the names provided earlier. Use case insensitive match for comparison. **[Hash data structure & String operations, 6 hours]**
25. Write a program to take the names of candidates as input. Prompt user to keep entering new names till user enters “done”. Once a list of names are accepted, prompt the user for a search pattern (regex). Output shall be a list of all names where the search pattern exists. **[Array & Loop & Regex, 8-12 hours]**
26. Visit https://merce.co and save the homepage as an HTML file from a browser as “merce-homepage.html”. Write a program to read the saved HTML file, compress it and store the compressed file as “merce-homepage.html.zip”. Use ready classes for compression. At the end, the program shall print HTML file size, Compressed file size. **[file operation concepts, 8-12 hours]**
27. Extend (26) to use URL classes instead of a browser to download a file. Rest of the functionalities will be the same as the previous problem. **[Java classes & file operations, 8-12 hours]**
28. Extend (26) to accept URL as a command line argument instead of a hardcoded URL within the program. **[Revision of previous concepts, 2 hours]**
29. Write a program to accept a filename from command line argument, read a file and print the number of times each word occurs in a file. Perform case insensitive match while counting the occurrence of each word. **[Hashmap & File operation & String operation, 6 hours]**
30. Extend the above program to ignore common words (e.g. “the”, “a”, “an”, …) and single letter words (e.g. “I”). **[Revision of previous concepts, 3 hours]**