GYROPROMPT USER MANUAL

GYROPROMP USER MANUAL: 2023 EDITION

The GyroPrompt user manual is meant to familiarize the user with the basic functionality of GyroPrompt.

OVERVIEW

GyroPrompt is a command-line centered scripting language.

GyroPrompt is an interpreted scripting language that exists on top of Windows architecture. GyroPrompt aims to simplify higher level tasks, ultimately helping users string together multiple tasks and processes to simplify the development of applications. GyroPrompt script files, which use the .gs file extension, can be automatically ran in the GyroPrompt console once your system is set to recognize GyroScript files.

[THIS PAGE RESERVED FOR TABLE OF CONTENTS]

VARIABLES

There are 4 types of variables:

- **Int (integer)**
- **Float (decimal number)**
- **String (string of character)**
- **Bool (true or false/1 or 0)**

Declaring one of the above variables is very simple, and modifying their values is also incredibly simple. In the code snippet below, we will declare 1 variable of each type. An integer named 'age', a string named 'name' and a float named 'networth'.

```
int age = 20
string name = Toby
float networth = 10.0
println My name is [name], I am [age] years old and worth $[networth]
```

The above code snippet would output "My name is Toby, I am 20 years old and worth $10.0" as **println** tells the interpreter we are going to print a line of text (**print** for just printing the text without a new line). Putting the variable names in square brackets [ ] tells the interpreter it's variable's value.

If later on I decide I want to set 'age' to my current 'age' plus my 'networth', I would use the **calculate** value modifier within the **set** command and it would look something like this:

```
int age = 20
string name = Toby
float networth = 10.0
println My name is [name], I am [age] years old and worth $[networth]
set age = [Calculate:{age}+{networth}]
println If I add my age and networth together, I could be [age] years old!
```

The **calculate** value modifier is a command that can handle mathematical equations containing multiple numbers and operations while abiding by PEMDA. It is important to note that any variables that are nested within a command like **calculate** must use curly brackets { } to properly differentiate.

Environmental Variables

GyroPrompt has a handful of hardcoded environmental variables that are tied directly to the console. The names are reserved, meaning you would be unable to create a local variable with the same name. Access to environmental variables is more managed in order to help manage stability. You can set the values of these commands with **environment set *variable* value** which will set the specified environmental variable to the value that follows.

Here is a list of the environmental variables:

- **CursorY – the Y coordinate of the cursor**
- **CursorX -the X coordinate of the cursor**
- **WindowHeight -the height of the console**
- **WindowWidth – the width of the console**
- **Forecolor – the text color**
- **Backcolor – the background color**
- **ScriptDelay – a number representing the number of milliseconds in a brief pause before executing the next command when a .gs script is running, mostly for smoothing the feel of operation**
- **Title – the title of the console**

The following environmental variables can take only a numerical value: **CursorY**, **CursorX**, **WindowHeight**, **WindowWidth**, and **ScriptDelay**. **Title** can take any typical string value. **Forecolor** and **Backcolor** are unique in that they must take a string specific value that matches one of the recognized console colors: Black, DarkBlue, DarkGreen, DarkCyan, DarkRed, DarkMagenta, DarkYellow, Gray, DarkGray, Blue, Green, Cyan, Red, Magenta, Yellow, and White.

Conditional Statements

A conditional statement will execute a series of commands if the specified condition is satisfied, ie '**if, then, else**' and '**while, do**'. The **if** command will execute one time the code following **then,** if the statement's condition is satisfied, and optionally if the condition is not satisfied the code following **else** will execute. A **while** loop will continuously execute the code following **do** until the condition is no longer satisfied.

```
int counter = 0
int target = 10
while counter < [target] do pause 500|int+ counter|println [counter]
```

The above **while** statement compares variable 'counter' to variable 'target'. While counter is less than target, the commands following **do** will execute. Multiple commands can be supplied and separated by a vertical pipe |

Value Modifiers

Value modifiers are commands and statements that produce an output or value modification. Since value modifiers are often bracketed, meaning they are nested within square brackets [ ] any variables within bracketed value modifiers should be nested within curly brackets { } to properly have their values passed. Value modifiers can sometimes be specific to a single variable type (often not bracketed and more like a standalone command) or they can apply to a more broad range of outputs if not almost all outputs (usually bracketed). Below is a list of common value modifiers and their properties.

Calculate:*expression*	**Processes the expression like a mathematical equation. (bracketed)**
RandomizeInt:*x,y*	**Generates a random number between value x and y. (bracketed)**
Toggle *bool*	**Toggles the value of a bool variable. (false becomes true, true becomes false)**
Int+ *integer*	**Increments an integer variable by 1.**
Int- *integer*	**Decreases an integer by 1.**

Value modifiers that can be operated within **print** or **println** (these are all bracketed)

Color:*color*	**Changes the font color to the console recognized color.**
Background:*color*	**Changes the background color to the console recognized color.**
\n	**Outputs a newline.**

Referencing a variable within a value modifier's square brackets [ ] should be done using curly brackets { }