Readable Code
for the
Present and Future

1.0 Introduction

This is a simple guide to writing reliable and quickly readable code that is universaly known to all programmers. It includes many aspects of coding and many practices of formatting and writing code

2.0 Deprecations and Bad Practices

2.1 The Task Library

https://create.roblox.com/docs/reference/engine/libraries/task Roblox introduced the Task library in august of 2021, with the Task library being used to directly talk to the game engines task schedule of tasks to be ran.

It brang and deprecated many functions such as:

```
wait(x: Number): Number -task.wait(x: Number): Number
```

While function of wait() is fundamentally the same it still differs from task.wait(). The difference is in the precision of the yield, coming from 1/30 of a second to 1/60 of a second with the Task Library

2.2 Parts and MeshParts (BaseParts)

While Parts and MeshParts may look like completely different instances, they are fundamentally the same with just the MeshPart having a different, editable topology

To create a BasePart you can use

```
Instance.new(ClassName, Parent)
```

While this may look like the best way to create an instance it can sometimes be less efficient if you do not know what you are doing. The primary optimization here will be the 2nd arguement when using Instance.new().

BAD:

```
local Part = Instance.new("Part", workspace)
Part.Position = Vector3.new(0,0,0)
```

GOOD:

```
local Part = Instance.new("Part")
Part.Position = Vector3.new(0,0,0)

Part.Parent = workspace
```

When you set the parent with the 2nd arguement of Instance.new it sets its parent immediately after the instance is made, meaning that it will also render and fire off RBXLSignalEvents for when the part is created. If you edit properties after setting the parent of an instance it can cause performance drops since you are editing the properties while it has running events and is being rendered.

2.3 The Invisible Variables

Sometimes you may just not see a built-in function in Roblox and then you try making it yourself. All Roblox API (Application Programming Interface) is written in C++ so it is marginaly faster than recreating it yourself in Luau.

You might just overlook some clear features that the API offers to you such as:

BAD:

```
local Index = 0

for i,v in {} do
    Index += 1
    print(Index)
end
```

GOOD:

```
for i,v in {} do
    print(i)
end
```

While you are looking at what variables you should use, you should also look at what variables to hide. You should also not be scared of naming variables to something that describes them instead of using single letters like i or v

BAD:

```
for i, v: BasePart in {} do
    Part.Position += Vector3.new(1,0,0)
end
```

GOOD:

```
for _, Part: BasePart in {} do
    Part.Position += Vector3.new(1,0,0)
end
```

3.0 The Art Side of Coding (Syntax)

https://roblox.github.io/lua-style-guide/

While one may think that code is purely written to be as most efficient as possible, it may be better to value readability over performance in some cases ** ** ## 3.1 Operator Syntax All operators have their own syntax for shortening and simplifying your code to make it more readable such as:

```
Math Operators
X = X + 1  -> X += 1
X = X - 1  -> X -= 1
X = X * X  -> X *= X
X = X / X  -> X /= X

String Operators
X = X.."JoinStrings" -> X ..= "JoinStrings"

Comparing Operators:
if X == true then -> if X then
if X == false then -> if not X then


if not X == 1 then -> if X ~= 1 then
```

This helps to shorten the code while increasing its readability, there is no performance change between the two ways of writing the syntax. Since it is compiled into the same bytecode

3.2 Variables

There are 2 types of variables: **Local Variables** and **Global Variables**, the difference between these 2 types is the scope of their declaration. Global variables have their scopes outside all functions while Local variables are scoped inside a function Block

Between the 2, Local variables should be used the most. Due to their performance benefits when compared to Global variables and their memory costs

Variables can have many values inside of them, some can have predetermined values such as when you are using **strict typing**

3.3 Strongly Strict typing

Strict typing or otherwise known as strong typing is a method of programming with strictly defined types for each value.

To use strict typing in Luau you have to declare it at the **top** of your script:

```
--!strict
Rest of the code
```

The syntax for defining a variables type is:

```lua
local MyNumber: Number = 0```

The syntax for defining a function type is:
```lua
local function RandomNumber(Min: Number, Max: Number): Number
    return math.random(Min * 100, Max * 100)/100
end
```

3.4 How_To WriteVariable NAMES

There are many ways of writing variable names such as: PascalCase, camelCase, snake_case

All of them are completely fine to use but should be used consistently with your coding accent

The Luau Styling guide that Roblox uses states that names should be spelt out fully, not abreviated to increase the readability of the code. Use PascalCase for Roblox API, since most camelCase API is deprecated. Use SCREAMING_SNAKE_CASE for constant variables (variables that do not change through the runtime of the script)
```