Best Practices

Sensible

Contents

[Git Procedure 3](#_Toc79832893)

[Structure of the project 3](#_Toc79832894)

[Process 3](#_Toc79832895)

[Committing on the wrong branch 3](#_Toc79832896)

[Data types 4](#_Toc79832897)

[Defining a variable 4](#_Toc79832898)

[Naming a variable 4](#_Toc79832899)

[Function names 4](#_Toc79832900)

[Variable names 4](#_Toc79832901)

[Where to use each data type 4](#_Toc79832902)

[Control Structures 5](#_Toc79832903)

[Implicit Documentation 6](#_Toc79832904)

[Commenting 6](#_Toc79832905)

[Functions 6](#_Toc79832906)

[Variables 6](#_Toc79832907)

[Complicated code 6](#_Toc79832908)

[Commit messages 6](#_Toc79832909)

[Debugging 7](#_Toc79832910)

[Running in debug mode 7](#_Toc79832911)

[Commenting to the console 7](#_Toc79832912)

[Asking the person who wrote it 7](#_Toc79832913)

# Git Procedure

<https://sharing.clickup.com/t/h/3cwtt8/9O0IDFM1WDMCXZ6?fbclid=IwAR12gLmhrc-_y5akmb0ZTeXdc9mh4jZPBuz5T80aOlwAAWVdZG6_xhzZivk>

## Structure of the project

Master. This branch will have a stable version from the last sprint. This branch should only be merged with the beta branch (consisting of work from the last sprint) at the start of each new sprint

Beta. This is the work of the current sprint. It doesn't necessarily have to be completely stable but each team member must only merge with this branch when they are sure the work they’ve done in their branch is relatively stable.

**Alpha branches. This is where each team member will be doing their work.** Team members can push to their own alpha branch as frequently as they want and it can be stable or unstable. However, it is important that this branch is stable before merging with the beta branch.

## Process

Every week, at sprint end, the stable *alpha* branches will be merged into *beta* by the Quality Assurance Manager. **You must make sure your branch is stable.** After the merges have been made, and any problems with *beta* have been fixed, all *alpha* branches will be updated with the new *beta* (which is the combined work of the sprint). After this, the *beta* version is merged with the master branch so that it also contains the most up to date work. Any unstable alpha branches can be merged at the next sprint.

## Committing on the wrong branch

Mistakes happen (like you) and if you realise you’ve done all your hard work and effort on the wrong branch, there is a way to fix this issue. Follow the instructions under “Transferring commits between branches” in this link: <https://sharing.clickup.com/t/h/3cwtt8/9O0IDFM1WDMCXZ6?fbclid=IwAR12gLmhrc-_y5akmb0ZTeXdc9mh4jZPBuz5T80aOlwAAWVdZG6_xhzZivk>

Or you can investigate it yourself here:

<https://gist.github.com/unbracketed/889c844473bcca1917e2>

# Data types

## Defining a variable

If the value of the variable is not going to change, use the **const** keyword before the variable name to define it as a value that does not change.

If the value of the variable is going to change, declare it as a **let** instead.

For variables that might be complex or hard to understand, you can comment above or after them to help people understand what they might be for.

## Naming a variable

### Function names

Function names should always begin with a lower case, and then every word in the title after that should have it’s first letter be uppercase (camelCase).

Function names should also be verbs, or actions, where possible.

Example: initialiseScreenData

### Variable names

Variable names should also be camelCase. Where possible, try to mimic the names of other variables around you, so that the application’s variables are all similar.

Also, check the spelling of words or phrases you are unfamiliar with when creating a variable.

## Where to use each data type

String -> It contains readable text, or characters.

Int -> A numeric value that you are planning to use for calculations or as an ID value.

Boolean -> There are only 2 possible states for the variable

Array -> You want to store a collection of similar variables

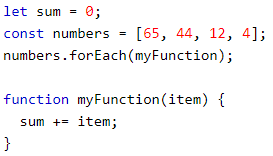
Object -> You want to store information about a variable that have different properties or types

Dictionary -> A paired storage method that you want to be able to search on

# Control Structures

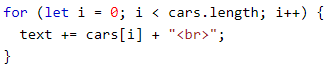
Iterating around an object/list/array -> Use **foreach**

Link to use: <https://www.w3schools.com/jsref/jsref_foreach.asp>

Example: (The function can be defined inside the foreach)

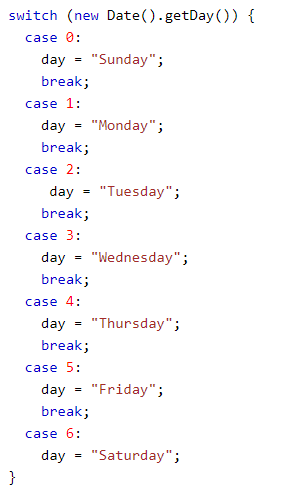
Iterating around an object where index needs to be used -> Use **for**

Link to use: <https://www.w3schools.com/js/js_loop_for.asp>

Example:

If there are a set number of possible options, and it’s not binary (> 2) -> Use **switch**

Link to use: <https://www.w3schools.com/js/js_switch.asp>

Example:

If there are a set number of possible options and it is binary -> Use **if**.

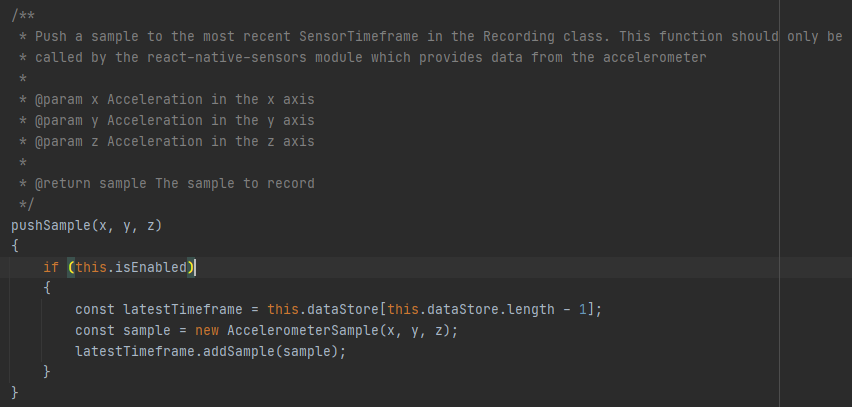
Additionally, custom functions should be kept to a maximum of 50 lines.

# Implicit Documentation

## Commenting

### Functions

A comment should be placed above every custom function. The comment should state the name, and creator of the function, as well as a short description. It should also have the inputs and outputs of the function (the parameter values and the return values), but if the function doesn’t have those it might be a good idea to define it anyway, as I’ve done below.



### Variables

Complex variables (Objects are a good example of this, as are dictionaries) should have some descriptive comments as to their contents and purpose.



### Complicated code

Comments don’t only need to be at the definition of things. They can be inside functions, explaining why you have a specific control structure or block of code, to make it easier for the next person. If you have written something that has a lot of lines, might be complicated or someone might have to work on next time, you should consider commenting it to make sure people know what to do.

## Commit messages

All commits should have a descriptive message, and if you were working on a specific task, you should include some indication for which task you were working on. This means it’s easy to see for everyone else what’s going on.

# Debugging

## Running in debug mode

You can run the application in debug mode, which allows you to stop and start it on breakpoints to check the values of variables. Since breakpoints don’t carry over onto other people, this is a good way to check the values of things mid-operation, and it’s very easy to see where you’ve logged them (however you should still take them out once the problem has been solved).

Commenting to the console  
Commenting to the console through the use of console.log is a good way to check a single variable at runtime, but the use of this should be reduced as much as possible. Additionally, removing these before you commit and push your final work is important, so if other people try to comment to the console, they don’t get all your debugging too.

## Asking the person who wrote it

The idea behind putting names to functions is that if you’re having problems with a complicated function, you can simply message them and ask. If there are no names on the function, then you can ask the group messaging and then hopefully get an answer that makes it a lot easier for you to do your thing or understand the code.