CustomJS primer for Calculatorians®

Written by:

Alvaro Diez

Not an Astrophysicist™

Dominik Czernia

Mr. CustomJS Wizzard himself

November 5, 2019

Omni Calculator Project

Contents

Chapter 1

What is this? Who am I? What is the meaning of life, the universe and everything?[Preface]

Let's start answering this questions in reverse order. Last question's answer is **42**. Regarding the one before it, I can't really answer for you, it takes a lifetime to discover. And for the first question's answer, I have bad news: It's long[1]

This *CustomJS primer* is meant as a quick-reference, or star-guide or user-manual for Calculatorians like you to (first) get started using Javascript in their calculators and (later) solve quick and simple doubts that you might have while using that knowledge.

This is NOT a formal, technical, precise, dense, boring, all-encompassing[2] Javascript book or a written lecture on programming. This document aims to provide understandable, applicable knowledge and will sacrifice technicality and precision if needed. If you have any programming experience you will find the first sections of the second chapter to be extremely basic and you might prefer to start reading from the chapter 3. If you already know how to program in Javascript, you might find most of it useless and some even probably offensive (if your style choices differ from ours) so we recommend to just read those sections that relate to the application of such knowledge to making OmniCalculators such as chapters and 4

Before we move on with the actual content let's faff around just a bit more and take a look at the different parts of this document and what to expect from them:

This document is divided into 3 different chapters of increasing length. The first chapter proper (2) is an overview of why and when to use CustomJS additions in your calculator and when it is not needed. This chapter also includes a short list of "best practices" regarding the technical side of CustomJS and an explanation of why they are like they are. To finish it off and to properly get you prepared for the second chapter, there is an overview of the most basic principles of programming with specific focus on examples in Javascript. Once finished with it, you should be able to understand the Javascript code that will be presented in the following chapters as well as understand most of the Javascript code you will ever find.

[1] That's what she said

[2] Thanks for the word Jack

The second chapter is a rundown of all the custom functions available at Omni. This is a collection of calculator-specific functions that let you modify the behaviour of each component of the calculator as well as add new functionalities. The aim of this section is to replace the old gist that was in Polish. Feel free to use this section as your main reference to understand, use and solve any basic problems regarding Omni's own functions.

The third and last chapter is focused on applying the previous knowledge in an effective and efficient manner. It is composed of a collection of common uses, typical combinations and behaviours implemented in the calculators we've made so far an includes a list of reference calculators where you can check these principles being applied (list on Trello). Then we included a collections of tips and tricks to prevent, diagnose and fix problems as well as a list of typical error behaviours and typical solutions. To finish it off, the last section of this chapter is devoted to setting some guidelines to make the process of creating, fixing, editing and improving customJS calculators as simple and painless as possible. Most of these guidelines have been set by the group of calculatorians and might deviate from the traditional "programming best practices" so, yes, they might not be the most generalisable rules, but just follow them and don't hate.

Chapter 2

Before you start coding

So now it is time to jump into the actual content. We will start by taking a look at what is customJS and what we use it for, so that you will know if it fits your purpose or not. We will also help you make such decision by presenting you with clear scenarios where customJS is not needed, could be needed and is compulsory to use.

After learning about customJS we will have a quick crash-course on programming so that no matter your starting knowledge you will be able to write and understand javascript code and make your calculator truly *Omni-Awesome*®. If you already have experience in programming or have taken any kind of programming course, this section (2.2) will likely seem redundant and won't teach you much, so feel free to skip it and move on to the next chatper: Chapter 3, were we dive into the custom functions that we have available at Omni but are not part of regular javascript.

2.1 Who is this CustomJS guy?[An introduction to customJS]

Let's start with the basics, CustomJS is one of the many options available for calculatorians when making a calculator. This option consists on a text window where you can write your own (Custom) javascript (JS) code to be executed when the calculator is loaded and during subsequent calculations. As a calculatorian you can chose to run plain Javascript code (though it's functionality is capped) or use some of the omni functions that are at your disposal and help you interact with the elements of the calculator as well as to run commands and different moments in the loading-calculating process. We will see more about this functions in the next chapter (3) but first we need to decide when it is required to use CustomJS in a calculator, when it is optional and when it is not a good idea.

2.1.1 You only need food, water and sleep

In the beginning there was Mateusz. He started making calculators at speeds never known to humankind before. But he couldn't make them perfect for he was only (partially) human. This meant prioritizing speed and throughput over quality and visuals. This was good in the pre-calculatorian times at Omni.

However, with time, Omni started to produce a rare breed of humans, humans with the ability of empowering other humans to calculate anything and, impossible as it seemed at the time, do so while having fun. These highly evolved calculatorians grew in size quickly allowing for priorities to be shifted from throughput to quality; Omni didn't need to just exist, it needed to be awesome, the best.

You see, when Omni was created customJS was nothing more than a time saver, a workaround, an 'unlocking' feature. However, with the advent of the calculatorians-age customJS is being more and more widely use in calculators, which is reflected in more beautiful calculators, more user friendly tools and overall better quality products.

It is for this reason that writting customJS has gone from a dark art to a truly useful skill that every calculatorian should know. From adding images, to making interactive calculators that show/hide variables depending on the user input, to graphs or even having different calculators in one, customJS can take your calculator from good to awesome!

That said, you shouldn't just use customJS for the sake of it. Omni calculators have to be as simple and easy to use as possible (without compromising functionality), so some times it is better to stick to the standard procedure than sacrifice user experience.

To assess better if your calculator could benefit from some customJS goodness, the best way is to look at some examples and understand what is possible, what is recommended and what is not recommended or just plain impossible to do; so let's do that!

2.1.2 Do what you can because you must

We will now show you the functionalities that customJS provides and its (dis)advantages by looking at 3 different scenarios. One where customJS is not needed or useful, one where customJS is not needed but can add extra functionality (so customJS would be "desirable") and one where customJS is needed.

CustomJS is uncalled for

Example: Simple Percentage Conveniently, this example is actually Omni's #first ever calculator and shows clearly when CustomJS might be an unnecessary addition. This is a very simple calculator that most people will use to quickly calculate the value they need. In this case we want to keep calculators as simple and easy to use as possible.

Another important reason why customJS is probably not a good addition is the fact that the concepts calculated and shown are simple enough that they don't require any visual aid, and there is also not clear visualization of the topics at hand that would greatly improve the usability or explanations that accompany the calculations.

In cases like this, one should just make a simple calculator and focus on making a kick-ass text to go with it where the user can learn more about the topic.

With all this said, is important to note that it is never cut-and-dry, a pie chart would help the user visualise what a given percentage means. It could be a decent addition depending on who visits the webpage, nevertheless we really think that in this particular case your time is be better invested elsewhere.

More Examples of unneeded CustomJS

- Percentage Increase Calculator
- Percentage of a Percentage Calculator
- Calculator

CustomJS is desirable

Example: Pythogorean Theorem This is also a very simple calculator, that doesn't need to include customJS to be a decent calculator.

However, a simple picture makes everything much easier to understand and simple to explain. You don't need to add help text to the variables and you can use the standard mathematical notation for each of the sides of the triangle. Which is a better approach than using "First/Second Cathetus" and "Hypotenuse" which assume some pre-existing.

A picture and some added text with the equation take this calculator from "usable" or "decent" to "very good" and that's something we want to do as often as we can.

This is by far the simplest example of customJS usage, but exemplifies very well the benefits of adding some customJS in the right places. Some other, more complex but still useful additions of customJS can be seen here. Note that none of this calculators "require" the use of customJS but greatly benefit from it.

More Examples of desireable CustomJS

- Calc

CustomJS is compulsory

And now we get to the interesting part of this section: the calculators for which you must use customJS. These fall in one of three categories:

- Calculators that need pre-sets
- Calculators with non-standard calculations
- Calculators suffering from Dissociative identity disorder

Pre-sets
An example of a calculator that requires pre-sets is the Chocolate calculator [id:941]. In this calculator there are predefined values that are not variables and are only shown in a table. These values can only be edited using customJS and are the main output of the calculator, making it compulsory to use customJS. Another, less standard example of a calculator using customJS as a way to offer pre-sets is the Bike Size Calculator[id:1629] where *type of bike* changes slightly the behaviour of the calculator. As a third and final example we have the most standard of all, which is the Screen Size Calculator[id:832] the screen size is controlled by a simple `omni.valueSetter`.

We haven't yet talked about them, but the customJS function commonly used in this type of calculator is the `omni.valueSetter`. We will talk about it in more detail in Section **??** but for now it's enough to know it give the user the option to changes the values of multiple variables at the same time via Calculatorian-defined tables.

Weird formulas
These calculators are those for which the calculation requires formulas or procedures that cannot be implemented via the *Equations* tab on the calculator editor. The simplest example of such a calculator is the Factorial Calculator [id:395] that requires the input to be integer before it can output the result. Other examples along the same lines are:

- Prime Factorization Calculator [id:143]
- GCF and LCM Calculator [id:171]

You can clearly see a common theme where the calculation processe requires some extra steps that are not provided by any standard function that you can simple input in the calculator editor. Once again, we will see in more detail what kind of equations and formulas we actually have access to when we talk about Javascript (Section 2.2) and the functions available at Omni (Chapter 3)

Multiple personality disorder
Some times when you make a calculator you want to add different options and behaviours so that you effectively have many calculator in one and the user simply changes between them by selecting from a drop-down menu. This kind of multiple personalty calculators are not always the best options but we all know that SEO works in mysterious ways, so at times is the best option, just make sure to confirm before you build.

Examples of these types of calculators are:

- Distance Calculator [id:144]
- Area Calculator [id:1569]

There is also a fourth type of calculator that is feared by calculatorians for its tough requirements, weird calculations and compulsory, unavoidable use of customJS: the marketing calculators. We will talk about them in the following section

2.1.3 when freedom is subjugated to the marketing needs

There are times in life when one needs must surrender its own needs and desires for a greater good. For calculatorians this time has been given the name of ***Marketing Calculators*** . And just as a good soldier must follow orders even against their own interest, a calculatorian must follow the guidance of the Marketing Team when the time comes.

In all seriousness, though, marketing calculators are a special breeding for which many of the ordinary rules and guidelines bust be ignored or at least relegated to a secondary role. Marketing calculators have different goals than regular calculators and therefore their requirements are different. One of these differences is the fact that customJS is a must.

In these instances customJs is not necessarily used to improve the functionality of the calculator but, mainly, for improving the user experience and fun-factor of the calculator, some times actively reducing the capabilities of the calculator since the main aim of it is not to solve a problem but to engage and entertain the user.

There are countless examples of these calculators but have curated some of the (subjectively[1]) best marketing calculators of all time:

- Chilled Drink Calculator [id:1556]
- Christmas Tree Calculator [id:1240]
- Exoplanet Discovery Calculator [id:1825]

[1]Possibly unrelated fact: all these examples were created by the author of this document

All these calculators tend to use customJs for some or all the reasons listed above, but also for reasons directly related to user experience. For example, the Chill Drink Calculator hides most of the technical variables behind user-selectable options that are easy to understand by laypeople. The Christmas tree calculator also does that and adds interactive graphics that help the user preview the results of the calculations. The exoplanet calculator is a perfect example of breaking almost every single rule about creating calculators heavily including html code to help the user understand and visualise the results.

For marketing calculators the **tl;dr** is that calculators should be entertaining and simple, they should include customJS and EVERYTHING is overridden by whatever the Natalia says at the time of building the calculator.

2.2 Programming vs witchcraft spells [Fundamentals of coding]

So you have evaluated the requirements of your calculator and you want to use customJS to make the best calculator ever created (or at least something cool). If you already know how a program works or have coded in any language before, you just got yourself a free upgrade to Section 2.3. In this section we will briefly go over the very basics of programming with focus on javascript. We will start from the very beginning and will teach you in simple terms the most important things you have to know to understand and write customJS. It is not intended as a full course or any kind of rigurous learning material, so feel free to take an online course or get yourself a decent javascript book if you really want to become a programmer. For the rest of you, Calculatorians, let's get started with the basics of programming.

2.2.1 What is a program?

We promised these will be a very brief and simple overview, so let us deliver just that:

A program is a text file that contains precise instructions understandable by a computer. The computer will perform this instructions in order as they are written (from top to bottom).

For a computer to understand a program you need to follow a certain convention, which is often called the syntax, and that is specific to each programming language. Here we will only talk about the javascript syntax (and briefly html). Bare in mind that even if the syntax is language-dependent, most of the concepts are the same across different programming languages[2].

For us calculatorians javascript is the main language we will be using, but sometimes we might come across some *HTML*, you need not worry about it, since it's recommended not to use it directly, but for completeness we will take a look at what it is and how it compares to javascript.

I have mentioned above that the code gets executed by the computer from top to bottom, but I did lie a little. First of all there are ways to control the execution order of the code, like the ones we will learn in Section 2.2.4 and 2.2.5. Secondly, remember the *syntax* thing I mentioned before briefly? Well that was a teaser, and even at the most basic level there are some things you

[2]This is almost always the case when we are talking about the most basic concepts such as the ones we will mention in this document

absolutely MUST know. First of all are **reserved words** which are those words that have special meaning in a given programming language and therefore cannot me used for other purposes. For example, you cannot used the as the name of a variable (more about what this mean can be found on Section 2.2.3). These words tell the computer clear commands regarding the next or previous words. Some examples are `var, for, if, NaN, continue, arguments, break, undefined,...` and you can find the full list on **??**W3 Schools or if you gently ask ~~God~~ Google.

Another crucial part of a language's syntax is how one conveys the end of an instruction. In many languages it is done by simply jumping to the next line to write the next command, while others, like Javascript, do this with a semicolon (**;**). This means that technically you could write all your javascript code in just one line. But you will not do, ever, cause you are a nice person and reasonable person that ~~don't want to have an accident happen to him/her~~ cares about others and like to do things in the best way possible.

We will talk later about other good practices that you should follow to make your code easily understandable. But for now we should leave it here with the most basic of the syntax rules in Javascript.

2.2.2 Javascript vs HTML

Picture this: You are happily looking at someone's calculator looking for inspiration when suddenly weird characters appear on the screen and threaten you with their 'totally not javascript' looks, or how you call it now:*syntax*. They probably look like this: `<a href='http...'>`. Fear not! This a friendly characters that have been relegated to the darkest parts of BB, but that love showing up here and there to help you when you've already lost all hope or javascript is simply not having a good day.

HTML's are not the same spices as Javascript, though. But they are the perfect companion for javascript and a loyal friend that help your javascript actually be used by people. Let me explain:

Javascript is similar to general purpose programming languages like Python, C or Java[3] and hence it is used to perform calculations, run algorithms and make webpages dynamic. On the other hand HTML is closer to LaTeX, or markdown (what we use to write the text of the calculators) and it's main purpose is to display things and make them look pretty. When people talk about HTML they usually mention also CSS which is (for our level, anyway) just an ordered way to create and store visual styles and desgin rules that HTML will follow, kind of like a configuration file for CSS.

There are many more differences between Javascript and HTML, like for example the fact that Javascript is generally ran on the user's computer (which is important to keep in mind when we build complex calculators) while HTML is ran on our server and only the results are shown in the user's computer.

[3]It's similar but DEFINITELY different

Compatibility is also an important factor to take into account when using Javascript and HTML. HTML tends to be much more widely supported across all browsers (nothing is perfect, though), while Javascript does present significant compatibility concerns for those who program websites from the ground up, which is not us[4]. Luckily we have nice developers that deal with those problems for us and don't allow us to use incompatible functions, which can be a bit frustrating when making very complex calculators, but it ensures that every user that visit our webpage will get a very nice calculator that works as intended.

tl;dr Javascript is for doing and HTML is for showing it to people. Javascript is run by the computer of the user, and our server does the HTML

2.2.3 Variables, functions, operations...

And now into coding proper. Just a small side note before we dive into it: ***Don't Panic!*** Programming does seem like witchcraft but unlike witchcraft it (1) really actually works and (2) if you make a mistake you don't actually risk the fate of the universe or anyone's life.

I once told my father *"Don't you worry when using a computer, you don't know enough about it to break anything serious"* and exactly this philosophy applies to you: if youare reading this guide you probably don't have the knowledge to truly break anything. With that said, $Don'tBeaDick^{TM}$don't try to prove me wrong, don't do things that you are not sure about and if in doubt don't ask yourself "what would this do?", but rather "who can I ask about it?". As long as you don't try to kill flies with bazookas[5] you will be fine, and you should not worry at all.

Variables - Primitives

First of all we need to start with the idea of a variable. A variable is a reference, a name, a shorthand or nickname to refer to something in a much more clear and efficient way. We do this in $RealLife^{TM}$ when we talk to people. For example, you would never say: "super enthusiastic, very demaning, tall, hard working, really nice boss/colleague, and the reason any of us finish a difficult calculator *on time*" and instead you would simply say **Bogna**, because it's simply quicker and conveys the same meaning. We can do the same thing in any programming language by defining variables and assigning them some values or properties. In Javascript all variables are defined in the same way:

```
        var Bogna = "super enthusiastic, very demaning, tall, hard working, really
            nice boss/colleague, and the reason any of us finish a difficult
            calculator on time"
```

In Javascript all variabels can be defined in the same way `var nameOfVariable = [expresion]` but it's important to understand the basic types of variables that exist, becuase each behaves differently and have different properties. Most variables cannot be mixed together and they require some kind of 'translation' from one type to another. Javascript does a good

[4]Sorry Darek and Daniel ^^ <3

[5]Spanish saying

job to automatically convert but sometimes you might find weird errors popping up due to incompatible types being mixed up. Here is a list of the most basic variable types:

- **int**: Integer number
- **float**: Decimal number
- **char**: One letter or character
- **string**: List of characters or piece of text. Think about it as a sentence
- **bool**: Binary logical variable which can be set to True or False

You will always use `var` before the variable name to declare a variable (declaring means creating for us) independent of the type of variable. When declaring a string you need to put quotes at the begining and at the end of the text, but it doesn't matter if you use sing (′) or double quotes ("). It's important to keep in mind what type we're working on. Specially important is when we want to operate with them since depending on the types we are mixing we will get different results, or even crash errors. Let's see more about this in the next subsection.

tl;dr Variables are a nickname for any expression we want and help us write less and same more. Variables can have many types and we mostly never care about them except when we do.

Operations

Operations are actions that you can perform with one or more variables to modify their value or create a new variable with some combination of the previous values. The most commonly used and notably simple are the mathematical operations such as **+**,**-**, *****(multiplication), **/**(division). Which should be self-evident. A more uncommon operator that is also very useful is the modulo operator (**%**) which returns the remainder of the first variable devided by the second one. The usage is very simple: `remainder = numerator%denominator`, and a practical example using number would be: `37%5` which would return the value `2` since 37/5 = 7 with a remainder of 2, which is the output of the modulo function.

Just remember that this is not an extensive documentation for javascript, so if you ever need any functionality or operation that we haven't mentioned, ask Google before you give up and look for a workaround. Speaking of things that are by no means exhaustively covered in this document: let's talk about operation on strings.

Strings are special variables in the sense that they act in many occasions as a single variable but can be also used as a collection of characters, kindof like a list of characters. The fact that the also carry information in natural language makes them more likely to be searched, splitted, slightly modified or even reformated. The gods of Javascript (yes, javascript is a revealed truth, not the creation of humans) though of these and gave us multiple operations that relate only to strings. The call them methods for reasons that will become apparent when we talk about objects in section 2.2.3.

We will mention here the (subjectively) most useful ones but make sure to check the reference for a complete list of all the methods available and a description of what they do and how to call them (that is, use them). Here we will only mention the three that we**??** feel are the most useful. That said, in Omni Calculators strings are mostly just concatenated using the + operator, so you might never use any of these three.

- `search()` Searches a string for a specified substring and returns the position of the match
- `replace()` Searches a string for a specified value and returns a new string where the specified values are replaced
- `split()` Splits a string into an array of substrings
- `length()` Returns the length of a string (number of characters)

An more complete list of all string methods can be found on W3Schools do check it out whenever you want to play around with strings, there are many options that can save you lots of time.

textbftl;dr Operations are sum (+), substract (-), multiply (*) and devide (/) but there are others we barely use. For strings no all this functions work but sum just add one string at the end of another. For all the string-specific operations, just look it up online (in the text above are some decent links). Just know that you can split, find characters, and other cool stuff.

Variables - Compounding like I have interest

So this is all well and good, but what happens when you have to mange hundreds or thousands of variables that are very similar to each other, or very closely related? ~~You give up.~~ You use compounded variables! (as I have taken the liberty of calling them for the sake of a pun) What we mean here by compounded variables is having one variable that represents a collection of variables. This will allow to organize and operate with these multiple variables in a much more organized and efficient manner. For accessing (reading) and working with each variable we will need to use loops, which you will learn more about in Section 2.2.4, in particular you will most commonly use `for` loop (Section 2.2.4) or a `while` loop (Section 2.2.4), at least in the most basic cases. There are several types of compounded variables, but here we will only take a look at two of them (again, (Google is your friend if you want to learn more): the **array** and the **dictionary**. The most intuitive to understand is the Dictionary, which technically is a Javascript object, but we don't need to know what that means for now. For us a dictionary is simply a collection of pairs of keys and values. Using the dictionary metaphor, the key would be the word and the value would be the definition. The dictionary is also a variables (I know it can get confusing but just look at the example) so it needs to be defined using the keyword `var`. Let's look at a example in which we set a dictionary with keys corresponding to the two letter abbreviation of a country and the value being the full name of that country; we have limited the number of countries in the example for simplicity and political reasons.

```
 var countryDictionary = {ES: "Spain", PL:"Poland", HU:"Hungary", IN:"India", PH:
    "Philippines", UK:"United Kingdom"};
```

As you might have notice the syntax is very simple, the dictionary is defined between curly brackets () and each key-value pair is defined as `key:value` (Notice that here our values are strings so the go in quotes). All key-value pairs, which are considered a single element of the dictionary, are separated by commas. In this example we have used values of type `string` but we could have used any other type including arrays (which we will see momentarily) or even dictionaries.

To access any element of the dictionary you can simply type the name of the dictionary followed by the key corresponding to the value we are interested in, enclosed in brackets. Let me show you, for example, how to get *"India"* using the dictionary from the example above: `countryDictionaty["IN"]`. If you can't wrap your head around it, you can think of `countryDictionary["IN"]` as the name of a variable that contains the string `"India"`.

This type of variable is very useful for *"translations"* and *"conversion"* and it's what underlies the creation of an Omni `valueSelect` (see Section 3.1.1) and it's also closely related to the Omni `valueSetter` (Section 3.1.2). There is also another keyword you might want to do some research on Google about: Object Literals. But you seriously don't need it to make calculators, so unless you like to complicate your life in the name of knowledge, don't do it.

Now we swiftly move onto **Arrays** probably the most useful of compounded variables for a calculatorian. You can think of them as a list of variables grouped together. If you are mathematically inclined you are better off thinking about them in terms of vectors or matrices, but if your head hurts from just reading those words, don't worry, ignore that advice.

Let's see how an array is defined in Javascript. This, again is a type of variable as you can see in the example.

```
 var daysInMonth = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31];
```

As you can see we have made a list that contains the number of days in each month (in regular year). Once again the elements of the array could've been anything, from numbers to strings to dictionaries to even lists. To access each element you need to remember the position of said element in the list. This shouldn't be hard if you have made a list that makes sense. For example, in this case the months are ordered as they appear in the year from first to last (Jan, Feb, Mar,...). Now that we now the position I want to access (let's say March, third position in the list) we simply use it to obtain the value by calling `daysInMonth[2]` and we obtain the number of days in March.

What? You are confused because I said **3th** position and wrote a **2**? You think is a typo? **NOPE!**[6]. This is **VERY IMPORTANT** so pay attention and don't forget: (like all sensible programming languages) **javascript starts counting the elements of a list at *0* (zero)**. So the

[6]Chuck Testa

first element is accessed by adding `[0]` to the name of the array. You will probably forget about this (cause we all have at some point) so at least try to remember that the order of elements in an array is something weird.

As for the main advantages of an array, there are many. First you can simply and orderly store multiply values in one convenient variable. The fact that is is ordered means you can access its values in a consisten way everytime without having to know anything about its contents. Second advantage is that since it can store any type of variables, it is very flexible and lets us use one type of variable for all our mass storage needs. On top of that, this is where the mathematiacally inclined rejoice, Arrays can be treated as multidimensional vectors or matrices.To do that you simple create an array of arrays to form what is called multidimensional arrays in which the value of the array is another array (and we can extend more, but once again we don't want to cause headaches). To access this arrays you simply follow your intuition. In this case an element of a multidimensional array called `myArray` is accessed by using `myArray[i]`. This is in itself and array which elements can be accessed as: `myArray[i][j]` where `i` is the position in the first (outter) array and `j` is the position in the inner array.

As a quick **tl;dr** just remember that Arrays (and to a certain exten dictionaries too) are you friends, they help you write less and do more while keeping everything organised. This means no more long hours spent writting thousand of lines of code that look almost the same[7] Arrays, lists, dictionaries, objects...

Functions

And we finish our basic collection of variables with NOT a variable. In technical terms a function is not a variable because it doesn't hold any actual data, but rather it holds information about processing data. A function can be though of as a dumb calculator, like those ones you see on the internet that are not made by Omni. Put simply a function has an input and an output. Both the input and output are comprised of none, one or many variables. Functions that don't have an output are typically called `void`, not that it matters much for us, calculatorians.

To simplify the concept you can think of a function as a variable that holds instructions instead of data, it's an algorithm. Typically we will use the to transform some variable or its values into another variable following a precise set of instructions. Functions, you might be thinking, are not technically necessary, but neither are you at Omni. Sorry, in all seriousness the analogy is not so bad, just as Omni could technically still work without one of us, or maybe two, or even just with Mat, your code can also work without any functions. However, the moment the workload gets too complex or too large, it becomes so painful and slow to do the work that you cannot deliver in time. Besides, just like Omni strives to be the best company possible (and for that calculatorians are a REQUIREMENT), you should also strive to create the possible calculators and, guess what: in many cases that means using functions. So just $Don'tBeaDick^{\text{TM}}$ and use functions :_)

[7] I'm really sorry Julia, we could've save you so much time with the Pomodoro... Hopefully this will still help you in the future

Let's see now how to define/declare a function and how to later use it.

```
function sumation(input1, input2){
    var sum = input1 + input2;
    return sum;
}
```

This function `sumation` does what it says on the tin: performs the sum of two numbers given as input. The way to define a function is using the keyword `function` followed by the name of the function, followed by the inpute variables in parenthesis (each of them separated by commas). If your function doesn't take any input just leave the parenthesis empty like this `nameOfFunction()`. After the parenthesis comes the code to be executed between curly brackets. A key component of a function with an output is the `return` statement. Whatever comes after said keyword will be the output of your function. If your function doesn't have any output (strange in calculators but can happen) just don't use the `return` keyword. You must keep in mind, that the moment a `return` operation is performed inside of a function, the execution path leaves the function and nothing else gets executed afterwards.

Functions can call other functions inside of them, and there are examples of that in our repository. Just remember that when you define a function in a calculator you don't have access to the calculator variables using `ctx.getNumberValues('variable')` and you would have to pass that value to your function as a regular input. Don't worry if this doesn't make sense yet, it will once we talk about omni's internal functions (Section 3).

To use a function you simply need to call using its name and providing the inputs required. 99.9% of the time you will want to store the output of the function in a variable, so here is the example to do so, using the function defined above:

```
var firstNumber = 5;
var secondNumber = 2;
var sumOfBoth = sumation(firstNumber, secondNumber);
```

The value of `sumOfBoth` will now be 7 (5+2) since that is the output of the function.

Functions are useful for two main reasons. On the one hand, the let you organize the code in a more understandable way and make complex operations more readable. And on the other hand, the let you reuse code without having to rewrite it; once you have created a function, no matter how long it is, it will take you one line of code to run that code again anywhere you want.

Another advantage, derived from reusability of code, is that you can actually search the internet for functions created by others that do exactly what you want to do. By simply copying and pasting them in your code, you will have all the functionality with non of the hassle. We have a library of useful function for Omni calculatorians in our Github repository, feel free to search use and contribute. More on how to do that in section 5.1.

2.2.4 Order of execution and loops - Basics

Bla bla bla up to down unless modifiers or functions.

if (if-else)

don’t over use them

for

the protytpe loop 4.4.2

while

for’s brother

break

DENIED!

switch...case

A fancy if, technically faster, only use for clarity

2.2.5 Order of execution and loops - Advanced

Don’t use, but they are cool, so maybe use?

do-while

for’s weird cousin

labeled

Make it your own!

continue

if you need help: click here

for...in

for’s weird cousing from Alabama

for...of

for’s weird-cousin-from-Alabama’s normal son

2.2.6 The laziness principle

If it takes more than 5min to do think if someone might have done it before and look for it (or ask politely) If you're doing the same thing more than 3 times, it can probably be automated. Never write the same thing (or almost the same thing) more than 5 times, there's surely a more efficient way[8]

-

2.3 A short list of strong suggestions [Do's and Don'ts]

2.3.1 Do's

follow the rules

2.3.2 Don'ts

follow the rules ALWAYS

-

[8]Exceptions might apply

Chapter 3

CustomJS at Omni [Built-in functions]

3.1 Omni special objects

3.1.1 valueSelect

3.1.2 valueSetter

- Pepe

3.2 onInit

3.2.1 `one for each function`

and its functions (and shortcomings) here

-

3.3 onResult

3.3.1 `one for each function`

and its functions (and shortcomings) here

-

Chapter 4

Okay, so you are already coding...

4.1 What can you do [typical additions in customJS]

4.1.1 The obvious answer

omni.functions && his friend

4.1.2 Advance uses for novices (a.k.a. basic uses for advance users)

4.1.3 Mix it up, spice it up!

you are free[1]

4.1.4 Useful examples

source from trello

4.1.5 How to memorize everything

Don't!

-

4.2 Sh*t! Why is this not working! [debugging for normies]

4.2.1 The disappearing calculator

Maybe you've triggered some unexpected behaviour, check how you are using omni functions and ctx functions

4.2.2 The error message in place of the calculator

You made a mistake, find it using this hints (google = friend)

[1]restrictions may apply. Free will is not guaranteed by Omni or the Universe

4.2.3 The "everything works but the result is wrong"

You made a mistake, find it

4.2.4 developer options, call html for help and other tricks

plan ahead and your life will be easier. Then again, where is the fun in that?

4.3 Do yourself a favor, do your colleagues a favor [Style guide]

4.3.1 I don't like rules, why do we have them?

Because rule number 1 is "You must have rules"

4.3.2 Organizing the code

Omni.defines
functions
general variables
onInit
OnResult
[final convention to be determined by democratic voting cause idc enough to be a dictator]

4.3.3 Formal style conventions

Bracket positioning, indentations, truncation of lines, spaces...
[final convention to be determined by democratic voting cause idc enough to be a dictator]

Naming conventions

thisIsAVariableNameThatLooksGood
this_i_do_not_like_but_is_alright_i_guess
this_ISHorrendous
DontDoThis
andneitherdothispls

Commenting

Comment the weird bits, comment for visual aid comment as much as needed and as little as possible

Space vs Tabs: the age old debate nobody should've had :(_)

Tabs are for losers, end of the story

-

4.4 The artform of asking for help and not being a total dick. [Give help, get help]

4.4.1 When to ask and when not to ask

4.4.2 How to ask and who to ask

test1: 2.2.4
test2: 4.2
test3: 2.2.1
test4: 4.2.1

4.4.3 Give back, everyone needs help some day

-

Chapter 5

Okay, but how can I...? [additional resources]

5.1 Omni (not so) exclusive resources

Appendix A

To infinity and beyond!

A.1 A collection of helpful resources

•

Bibliography