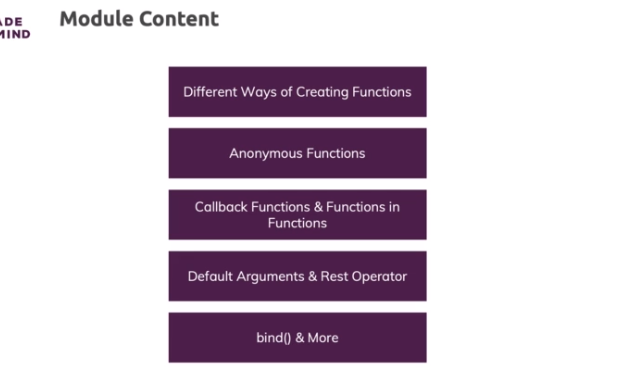
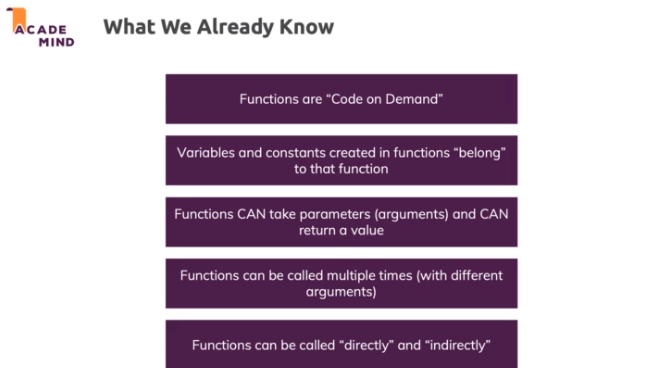
**Functions**



**Recapping Functions Knowledge - What We Know Thus Far**



**Parameters vs Arguments**

**Parameters** are these variables which you **specify between parentheses** when defining a function.

    1. function sayHi(name) { ... }

In this example, name is a parameter.

**Arguments** then are the **concrete values** you pass to a function when calling that function:

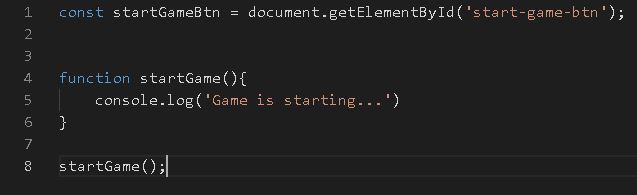
    1. sayHi('Max');

'Max' is an **argument** of the function therefore - for the name parameter to be precise.

Since both concepts obviously are extremely close connected, I will often say *"let's define which arguments a function receives"* or something comparable, since defining the arguments of a function in the end means that you set up its parameters (and vice-versa).

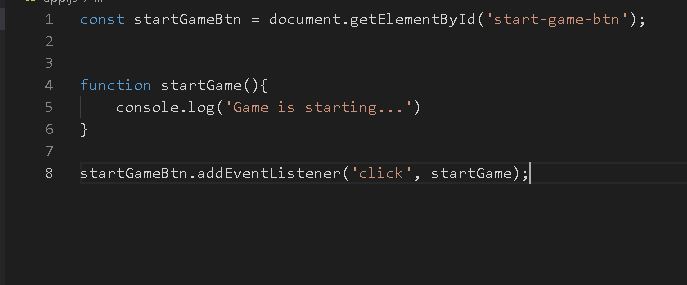
**Functions vs Methods**

Direct execution function



https://paper-attachments.dropbox.com/s_496624715D96282E93B2F05246D0B693ACB75D6EF37A40C2C0B2D4732AD245DA_1632378694817_vsdsd.JPG

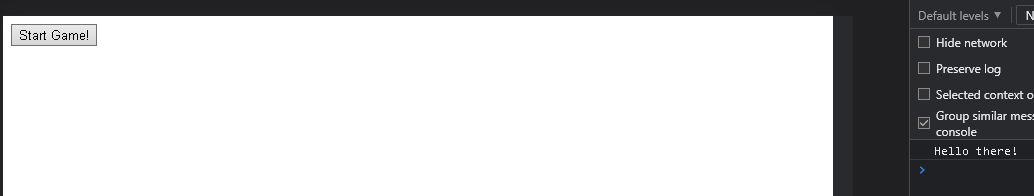
Indirect execution function





* Functions can be stored inside of objects as well
* Dot representation not only for declaring objects but for functions too





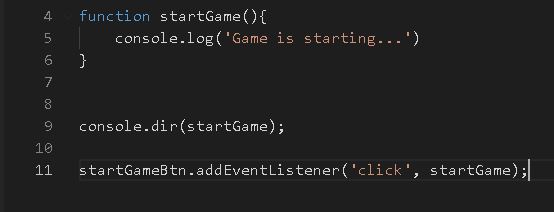
* We can store a function in an object and that is then called a method
* A method is nothing else than a function attached to an object

**Functions are Objects!**





https://paper-attachments.dropbox.com/s_496624715D96282E93B2F05246D0B693ACB75D6EF37A40C2C0B2D4732AD245DA_1632379792390_image.png

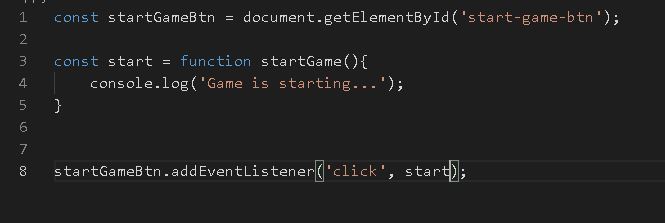




* Functions are basically also an object, a special type of object if you will with special pre-configured properties
* It is typically stored in the heap memory

**Function Expressions: Storing Functions in Variables**

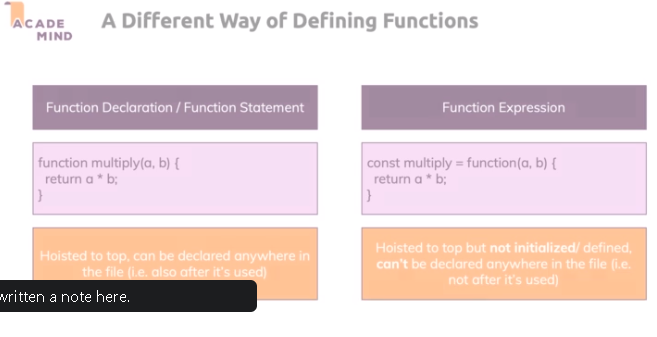
* So you can store functions in variables or constants



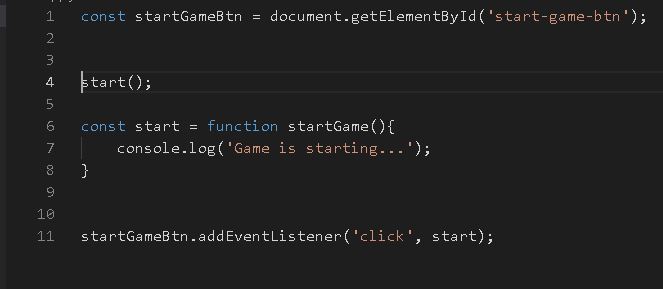


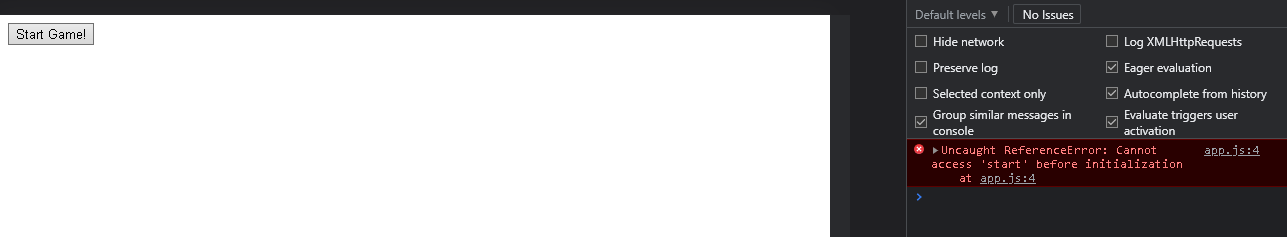
* So now we're creating a function in this expression mode and therefore it generates a function object
* we just store it in this variable, that variable or that constant here is then stored in a global scope but we can therefore now only reference this function by using that variable or constant name

**Function Expressions vs Function Declarations**

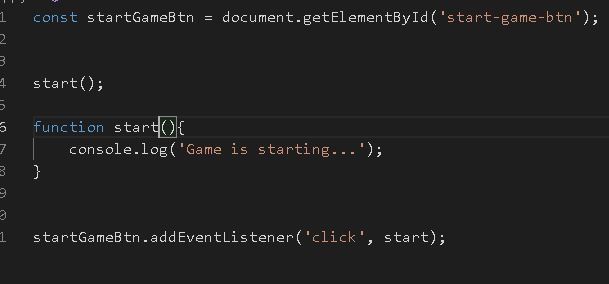


* Declaration can’t be done after calling the function, it will show error



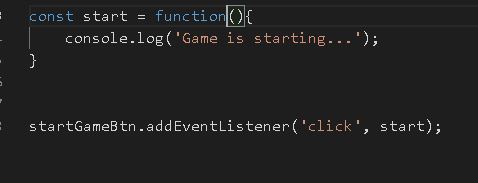


* Javascript reads the entire file and if it finds function declarations like this, it kind of acts like they were at the top of the file.

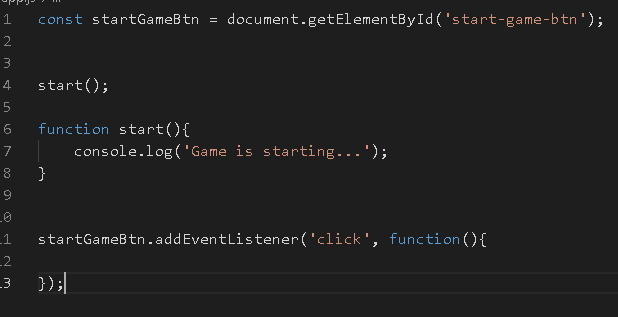


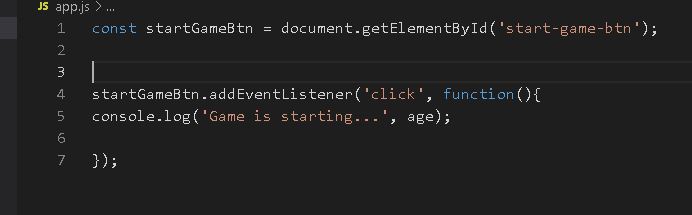


**Anonymous Functions**



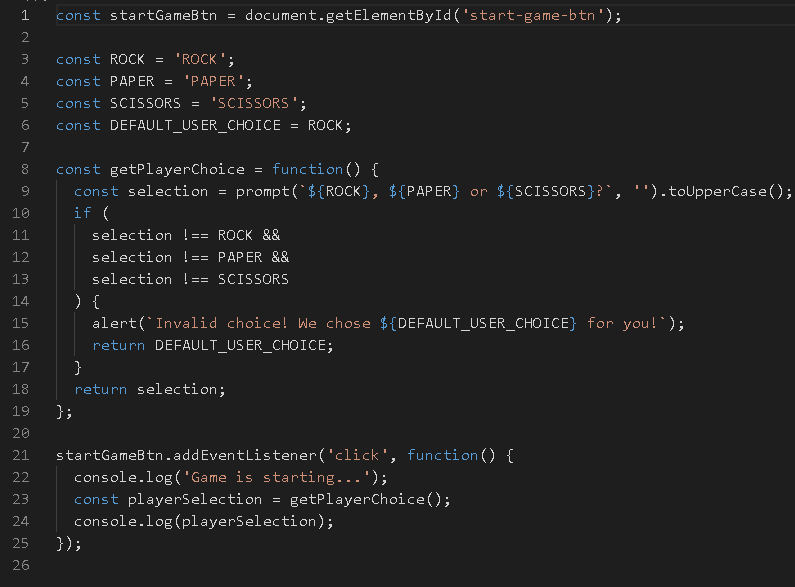
* This now is a so-called anonymous function and typically you then place the parentheses
* you can also use anonymous functions in places where you don't even store them in variables

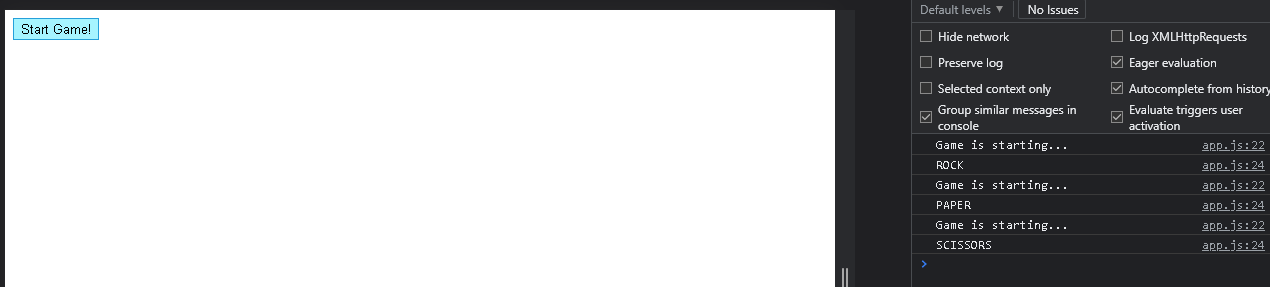




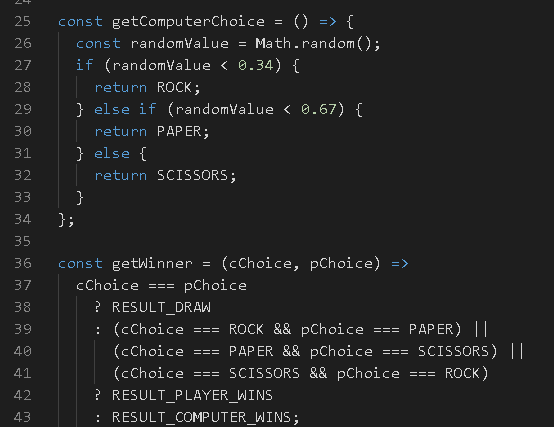


**Working on the Project: Adding User Choices to the Game**





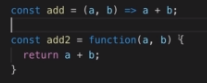
**Implementing the Core Game Logic**

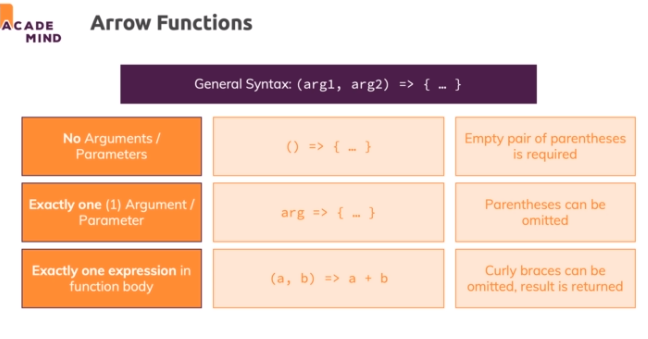


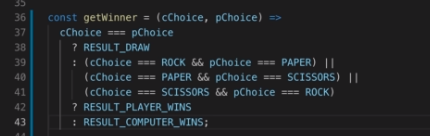


**Introducing Arrow Functions**

* you always have to store it in some variable or use it in a place where you would use an anonymous function
* it's an anonymous function where you can't assign a name in any way
* you always use it on the right side of the equal sign or in places like down here with the event
* arrow function don’t need curly braces







**Different Arrow Function Syntaxes**

**1) Default syntax:**

    1. const add = (a, b) => {

    2.     const result = a + b;

    3.     return result;

    4. };

**2) Shorter parameter syntax, if exactly one parameter is received:**

    1. const log = message => {

    2.     console.log(message);

    3. };

**3) Empty parameter parentheses if NO arguments are received:**

    1. const greet = () => {

    2.     console.log('Hi there!');

    3. };

**4) Shorter function body, if exactly one expression is used:**

    1. const add = (a, b) => a + b;

**5) Function returns an object (with shortened syntax as shown in 4)):**

    1. const loadPerson = pName => ({name: pName });

That last case can be confusing: Normally, in JavaScript, curly braces always can have exactly one meaning.

    1. const person = { name: 'Max' }; // Clearly creates an object

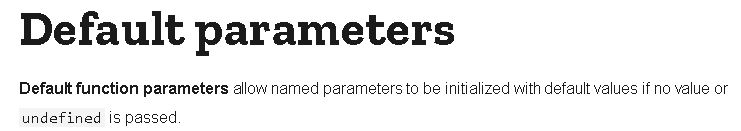
    2. if (something) { ... } // Clearly used to mark the if statement block

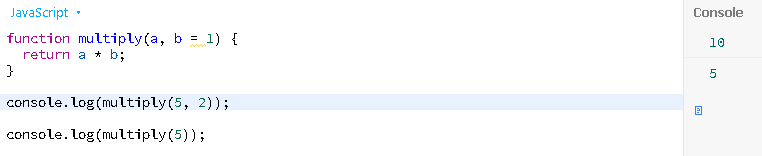
But when using arrow functions, curly braces can have two meanings:

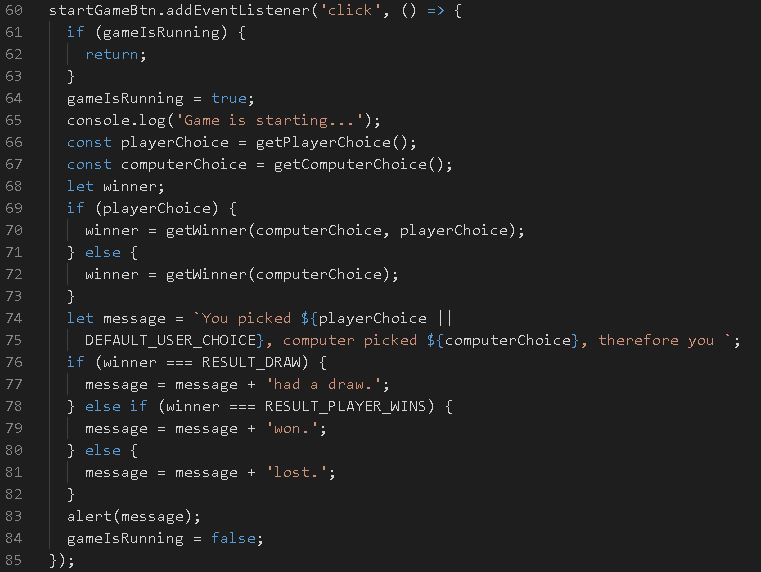
1) Mark the function body (in default form)

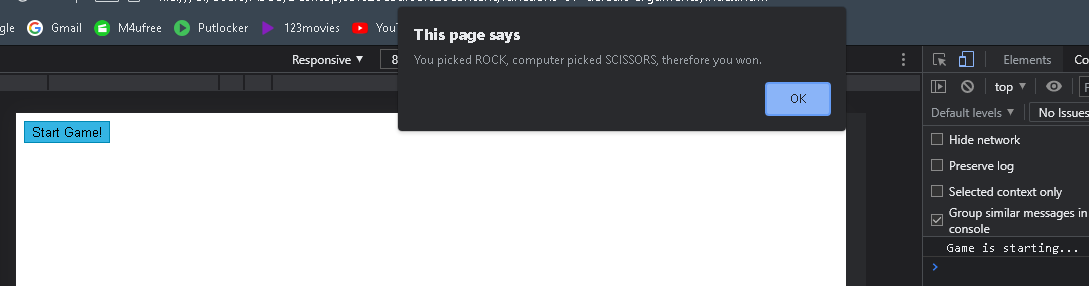
2) Create an object which you want to return (in shorter function body form)

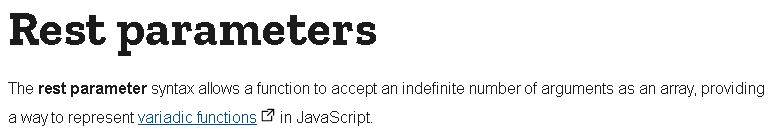
To "tell" JavaScript what you want to do, wrap the expression (e.g. object creation) in parentheses like shown above

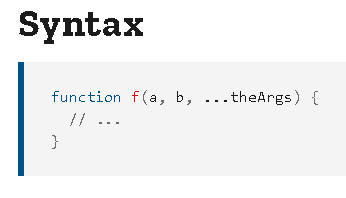


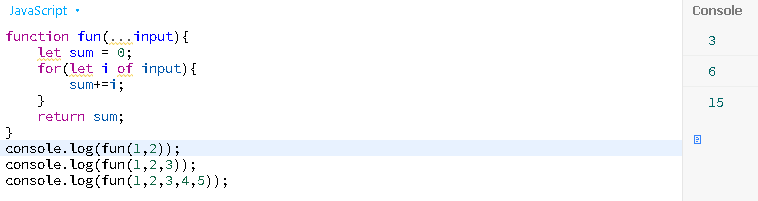






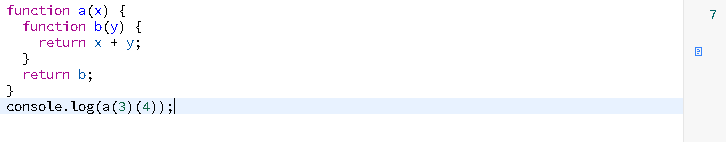


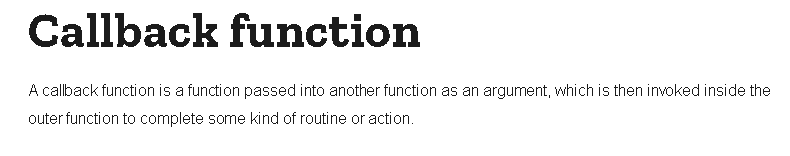


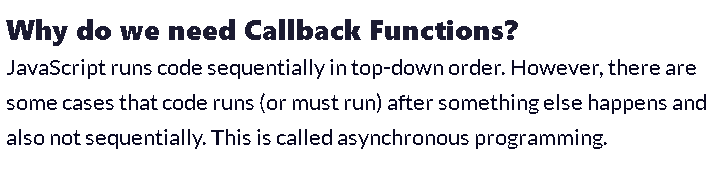


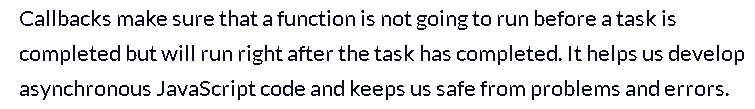
**Nested functions**

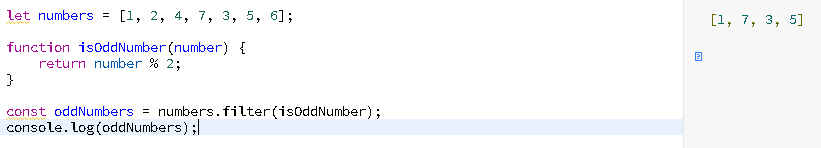
* The nested (inner) function is private to its containing (outer) function
* The inner function can be accessed only from statements in the outer function.
* The inner function forms a closure: the inner function can use the arguments and variables of the outer function, while the outer function cannot use the arguments and variables of the inner function.

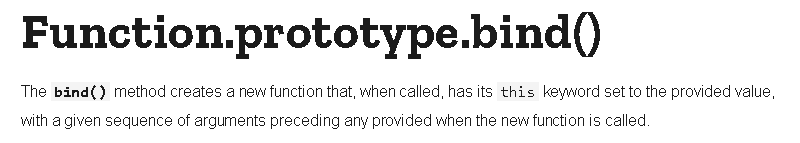




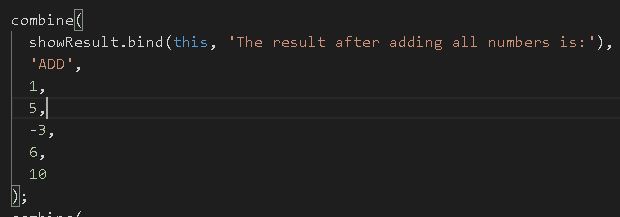


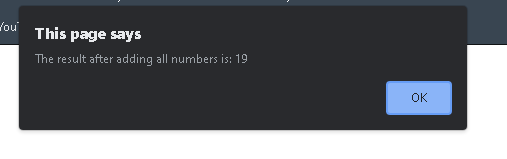






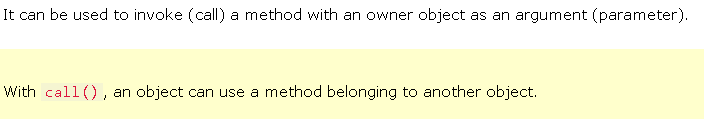
* it will create a new function, a new reference at a function which it returns to you which will be preconfigured regarding the arguments it receives
* bind takes at least two arguments

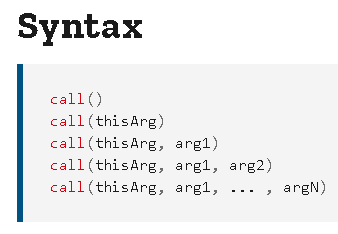






https://paper-attachments.dropbox.com/s_496624715D96282E93B2F05246D0B693ACB75D6EF37A40C2C0B2D4732AD245DA_1632388774384_image_2021-09-23_144934.png







https://paper-attachments.dropbox.com/s_496624715D96282E93B2F05246D0B693ACB75D6EF37A40C2C0B2D4732AD245DA_1632389796367_qweqwdd.JPG

https://paper-attachments.dropbox.com/s_496624715D96282E93B2F05246D0B693ACB75D6EF37A40C2C0B2D4732AD245DA_1632389805951_image_2021-09-23_150645.png

