string methods

# design

I am going to approach String Methods in a very similar style to how function and array calls are implemented. That being, after each factor, the interpreter should check if a property is present, alongside checking for the brackets which indicate a call. Then, the old factor will become a child of the property, and the process will iteratively repeat to construct the tree.

### property and parsing

Property will be the new class, and itself will mainly be a placeholder for other classes to extend it, and so that the parser can interact with them. It will have a callee class, to allow for polymorphism with the Call and ArrayCall classes, and the item that is will be the property of will be that callee.

A diagram of a process

Description automatically generated with medium confidenceTo avoid the factor() method being huge, I will be adding a new calls\_or\_property() class which will handle adding these to the factors. At the end of factor, there will be a while loop to repeatedly call this method while the current token is a Property, ‘(‘ or ‘[‘, and the current result will be passed to it, and result will be set to its return value .

**🡺 calls\_or\_property()**

This method will accept self, as expected, but also the previous factor. If the current token is a property, then the method will set the property’s calle to the old factor, continue() and then return the property to become the new factor. If not, then the old code for calls and array calls will make up the rest of the method, so that the Call or ArrayCall will be returned instead.

This implementation is very simple, but now properties should be able to be correctly parsed and built into the AST for evaluation.

### lexing

Whenever a full stop character is detected in make\_tokens\_line(), the new make\_property() method will be called. It will operate very similarly to make\_identifier() and will get the name of the property by continually iterating through the characters until a non-letter is returned. The position will be the original position of the full stop for consistency.

At the end, a switch statement will occur based on this name, and the corresponding property will be returned, and if it does not exist then an error will be returned, which is different to make\_identifier() as only the pre-defined OCR properties will be allowed.

## the property subclasses

Now, I need to implement a different subclass for each type of property. These will all either have an evaluate() method or a call() method, depending on how they are used. For example,

“example”.length will have an evaluate() method, because it is not used with brackets.  
“example”.substring(1, 1) will have a call() method, because it is used with brackets.

So, depending on if the property will be called or not, then the class will have either an evaluate() or call() method, and the wrong option will return an error message to specify that it was used incorrectly.

Now, I will plan and explain the process of every single property.

### length

The Length class will contain an evaluate() method.

In evaluate(), the callee will be evaluated. Then depending on the type:

🡺 StringType: An IntegerType with the value of the length of the string’s value will be returned.  
🡺 ArrayType: An IntegerType with the value of the Array’s arrayLength will be returned.

The specification does not explicitly say that the length property can be used on arrays, however I feel like it is suitable to include as it will make a lot of programs a lot easier to implement, such as all sorting algorithms, so I feel like it should be included as it is very core to a lot of languages.

If the callee is not one of these types, then an error will be returned.

It will be created by the “length” name in make\_property()

### substring

A close up of a word

Description automatically generated The Substring class will have a call() method because it accepts arguments.

The callee and call arguments will be evaluated and stored in variables to distinguish them with clearer names: string, index, and length. The callee will be the string, and the call will contain the parameters.

As written in the spec, the first argument will be the starting index of the substring, and the second will be the length of the substring. This is different to a lot of programming languages, which have the start and end index, so the implementation will need to be different. Overall, the steps in call will be:

🡺 Ensure that the callee is a StringType  
🡺 Ensure the index is an IntegerType and has a value from 0 to the length of the string – 1  
🡺 Ensure the length is an IntegerType and has a value from 1 to a valid value for the given index  
🡺 Return a new StringType with the value of the corresponding substring

It will be created by the “substring” name in make\_property()

### left and right

Because they will operate so similarly, I am going to combine these into a single class called LeftOrRight.

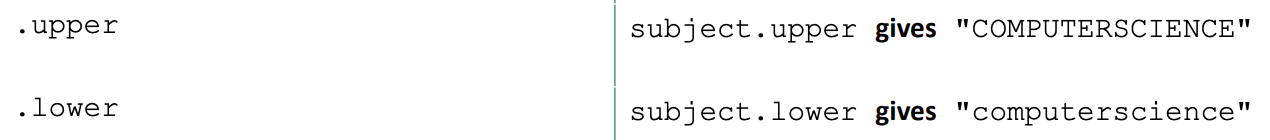
This is because they both take the same steps until the final return, as they both require a call(), need the callee to be a StringType, and need a single IntegerType argument with a value between 1 and the length of the file, so they will both do these checks.

The way this class will distinguish between the types will be a tag property, being either “left” or “right”, that will be assigned in the constructor, and given to it in make\_property() when “left” or “right” are the names.

Then, at the end of call(), based off the tag, the corresponding substring will be returned. If the tag is “left”, then the substring from position 0 to the length - 1 will be returned, and otherwise, if the tag is “right”, the substring from the string’s length – the length to the string’s length - 1 will be returned.

This therefore allows the two string methods to be combined into one, due to all the similarities between the two. Additional cases will probably have to be added to ensure that the string that is being used is not empty, as it could cause some issues, but that can be solved in testing.

### upper and lower

Like Left and Right, I will be merging Upper and Lower into a single class called UpperOrLower due to their similarities.

Again, the system for selecting this will be the same as LeftOrRight, with a tag given in make\_property().

The class will have an evaluate() method because it is not followed by brackets. After ensuring that the callee is a StringType, it will return a new StringType with the value being uppercase or lowercase, depending on the tag and using the JavaScript built-in functions.

This method is a lot simpler, and after implementation then all of the string methods should be complete.

# development

## lexing and parsing

A computer screen with text on it

Description automatically generatedI’ve decided to add evaluate() and call() methods that return Errors to the main new Property method.

This therefore means that I do not need to write a custom error message for every single property that is incorrectly called.

A computer screen with text on it

Description automatically generatedA screen shot of a computer code

Description automatically generatedThe new factor changes are left.

This is mainly to reduce the size of factor, as the decomposition makes it easier to manage.

A black background with white text

Description automatically generatedMost of the new method is copied and pasted, and the Property addition is very simple.   
As a test, I ran a simple print message to ensure that function calls were being correctly parsed, and it worked.

### A black screen with text Description automatically generatedlexing

A computer screen with text

Description automatically generatedA computer screen with text

Description automatically generatedThis was not on the plan, but I moved the code that parses identifier names into a separate method so that it can be reused for the new make\_property().

The new method is very simple, almost identical to the make\_identifier() method, except it will store the position of the full stop for the position. For now, the switch is empty, but it will be expanded, and a check in make\_tokens\_line() for the full stop is added.

## A screen shot of a computer program Description automatically generatedlength

A black background with blue and yellow text

Description automatically generatedThis will be the easiest property to implement.

A black screen with white text

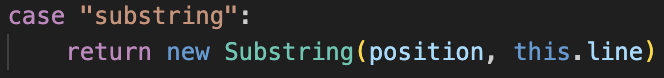
Description automatically generatedA screenshot of a phone

Description automatically generatedThe implementation is very clear, and it has been added to make\_property()

## substring

A screen shot of a computer program

Description automatically generatedThis property is a lot longer, but overall, it is just a lot of checks as discussed in the plan.

It ensures that there are no errors in evaluation, that the data types are all correct and that the index ranges are valid.

The substring check is also added to the new method.

### A black background with white text Description automatically generatedA screen shot of a computer Description automatically generatedissues

A black screen with white text

Description automatically generatedWhen trying to run a test case, a strange error message was produced.

When checking the ast, the structure was as expected, so was not the issue.

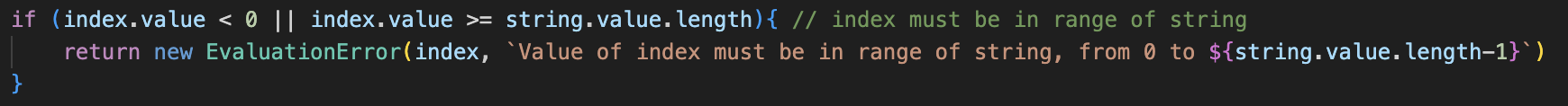
When realising where the error was coming from, it was the incorrectly formatted error messages in the call class. I fixed this issue, by adding the additional arguments for the error token, alongside giving the call() method the additional argument.

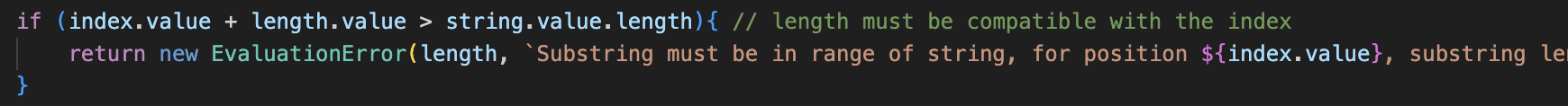
A black background with colorful text

Description automatically generatedHowever, there was still an issue: the property should not have been evaluated in the first place, and it is an issue if evaluate() is called. When looking into Call’s evaluate method, because it previously was only used for functions, which are stored within variables so need to be evaluated to access them.

Now, Call’s evaluate() method is updated to check if the callee is initially a property, and if it is it will directly call it without evaluating first.

Now the substring works as expected for normal cases. I also checked each of the other tests accounted for and had to make two small changes:

The first was that the error message for invalid indexes needed to have the upper bound for indexes to be one less than the length of the list, needing a -1:

The second was that the check that the length corresponds with the value needed to be exclusive, instead of using a greater than or equal sign, because when trying to access the last character, it would incorrectly return the error. Again, this is a very easy change:

For now, substrings are complete, but I will do more formal testing later.

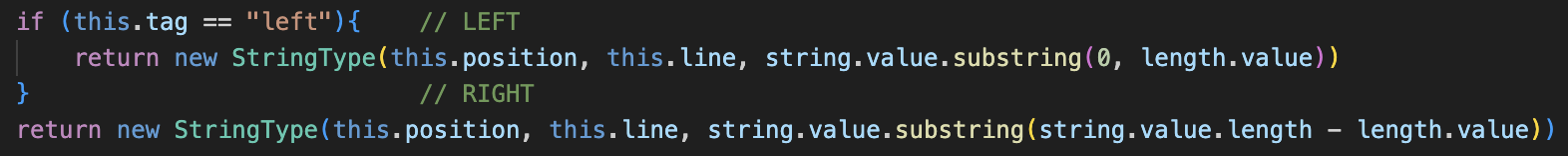
## A screen shot of a computer code Description automatically generatedleft and right

Because the left and right methods are almost identical, they use the same class, with a tag to determine the value returned in the final step.

A black background with white text

Description automatically generatedA screen shot of a computer code

Description automatically generatedAfter being added to the Lexer, I ran the test case:  
  
Initially this didn’t work, but the changes was to no longer add 1 to the length for the left result, and to take the length of the string’s value rather than itself, as shown:

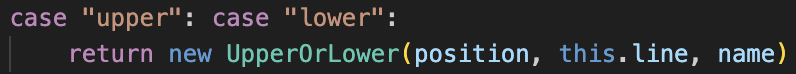


A black background with white text

Description automatically generatedNow the correct values were returned. When testing each case, the error when a non-Integer is passed to the call crashes the program, but it was fixed as shown in the main code.

## upper and lower

A screen shot of a computer code

Description automatically generatedA computer screen shot of a program code

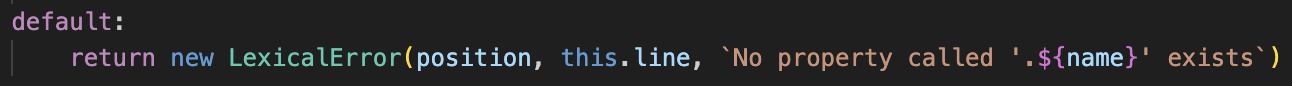
Description automatically generatedThe code for this property is very simple, as shown:

A black background with white text

Description automatically generatedAs shown, the two test cases for this instantly worked, and testing the method on non-strings also functioned as expected.

## errors

**🡺 Non-existent properties**

A black background with white text

Description automatically generatedThe main issue is that currently, if a property is references that don’t exist do not flag any errors, so a default clause needs to be added to make\_property(), which will then catch any non-existent properties instead of ignoring them.

**A black background with white text

Description automatically generated🡺 Integer and float with properties**

**A screen shot of a computer screen

Description automatically generated**Currently, when trying to use a property on a number that is not stored in a variable, it breaks how the Lexer works and produces a weird syntax error. This is because the input on the left translates to an Integer and an Identifier.

Instead, make\_number() now makes additional check after each full stop, to determine whether the character after is another number or a letter.

If it is not a number, then it will exit out of the method, returning the number before the full stop so that make\_tokens\_line() can then parse the property.

A black background with blue text

Description automatically generatedBut there is a problem that it exits at the character after the full stop.

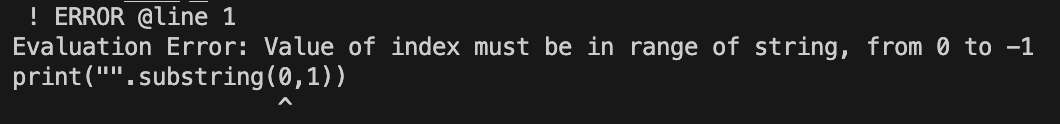
This is a very simple fix, adding a reverse() method, which simply decreases the position to the previous character, so reverse() can be called before exiting.

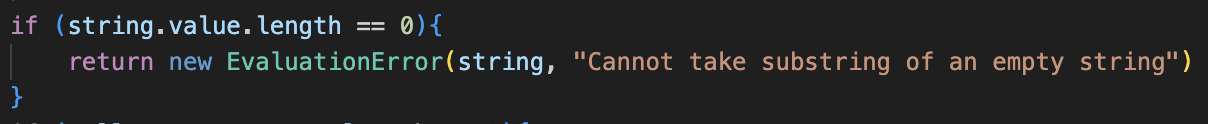
A screenshot of a computer program

Description automatically generatedNow, as shown, the proper error messages are returned. Even though there are no properties that work on numbers, anyway, ensuring that is correctly Lexes is still important as there could have been some weird cases which really broke the system.

A screenshot of a computer program

Description automatically generatedNormal floats still work as well and can still be created as usual.

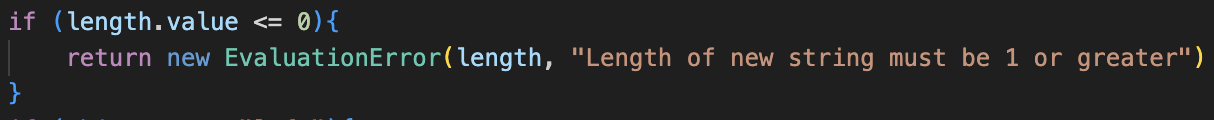
**🡺 Empty substrings**

****Currently, the error messages when using substrings are slightly broken.

A black background with white text

Description automatically generatedBy adding an extra check in Substring’s call(), for if the string has a length of 0, then it will return a custom error to avoid issues with the suggested values.

A black background with white text

Description automatically generated🡺 Negative values with left and right.

Currently, negative values will end up with nothing being printed, but I would rather an error be returned, so I added an additional check in LeftOrRight.

Now, this new error message will be displayed in this case.

This module does not have loads of places for errors to occur, so all of the test cases that I’ve tried have succeeded, except for the ones shown which I’ve now fixed.

### examples

A screen shot of a computer program

Description automatically generatedA screenshot of a computer program

Description automatically generatedNow with all these features, some quite complicated algorithms can be written in ERL and run, and the properties allow for more dynamic inputs to be used, which is very helpful for making it more suitable to being used instead of an actual high-level language.

A screen shot of a computer

Description automatically generatedA black background with white text

Description automatically generatedThe two examples here make the font irregular and convert inputs into their binary forms.