

**CSCI 468: Compilers**

**Capstone Documentation**

**Spring 2023**

# Section 1: Program

Please include a zip file of the final repository in this directory.

## Section 2: Teamwork

*Describe how your team worked on this capstone project. List each team member's primary contributions and estimate the percentage of time that was spent by each team member on the project. Identify team members generically as team member 1, team member 2, etc.*

The team divided work roles into primary engineer (team member 1) and documentation/testing engineer (team member 2). Team member 1 implemented the Catscript tokenizer, parser, language evaluations, and bytecode compilation based on the tests provided by team member 2. Team member 2 was also in charge of the documenting the features of Catscript.

## Section 3: Design pattern

*Identify one design pattern that was used in your capstone project and describe exactly where in the code it is located.*

One design pattern that was used in our capstone project was the memoization pattern. It is located in `edu/montana/csci/csci468/parser/CatscriptType.java`.

*Highlight the design pattern in yellow. Explain why you used the pattern and didn't just code directly.*

```
private static final Map<CatscriptType, CatscriptType> LIST_TYPES = new HashMap<>();
public static CatscriptType getListType(CatscriptType type) {
    CatscriptType listType = LIST_TYPES.get(type);
    if (listType == null) {
        listType = new ListType(type);
        LIST_TYPES.put(type, listType);
    }
    return listType;
}
```

We used the pattern to improve the performance of getting the list type in Catscript. If `getListType()` is called often, then we don't want to keep newing up a `ListType`. We can store any first time types in a hashmap and just do a quick look up if that type is asked for again.

**Section 4: Technical writing. Include the technical document that accompanied your capstone project.**

# Catscript Guide

---

## Introduction

Catscript is a simple scripting language. Here is an example:

```
var x = "foo"  
print(x)
```

---

## Features

### Print

In the context of the custom-made programming language "catscript," a print statement is used to output a value or message to the console. The basic syntax of a print statement in catscript is as follows:

```
print(x)
```

Where x is the value or message that is intended to be sent to the console.

For example, the following catscript code demonstrates an print statement on a saved string:

```
String x = "I want this to be printed"  
print(x)  
Output: "I want this to be printed"
```

### Variable Assignment

A variable assignment statement is used to assign a value to a variable. The basic syntax of a variable assignment statement in catscript is as follows:

```
var x = expression
```

Where the var x represents the identifier being instantiated and expression is the value being assigned to x

For example, the following catscript code demonstrates an identifier being instantiated and then used:

```
var answer = 42  
print(answer + 5)  
Output: 47
```

### If Conditional

An if statement is used to conditionally execute a block of code based on a specified condition. The basic syntax of an if statement in catscript is as follows:

```
if(conditional){
```

```

        statement
    } Optional else if (conditional){
        statement
    }else{
        statement
    }

```

Where if and else are mandatory keywords used to define the else branch and if branch for the statement, each conditional represents the necessary truth value to follow that branch, and each statement represents what is executed if the conditional above is true. There is also the option for an if else branch, where a conditional is considered in order to enter the branch.

For example, the following catscript code demonstrates an if conditional with an else if branch:

```

int x = input("Enter a number: ")
if(x < 5){
    print("x is smaller than 5"
} else if(x > 10){
    print("x is bigger than 5"
} else{
    print("x is between 5 and 10"
}

```

## For Loops

A for loop is a type of control flow statement used for repeatedly executing a block of code for a fixed number of times or iterating over a sequence of values. The basic syntax of a for loop in catscript is as follows:

```

for(var x in expression){
    statement
}

```

Where the var x represents a local identifier to be used within the statement body as the expression is iterated through.

For example, the following catscript code demonstrates a for loop that iterates over a list of names and prints out each name:

```

names = ['Alice', 'Bob', 'Charlie']
for(String x in names){
    print(x)
}

```

```

Output: Alice
        Bob
        Charlie

```

## Return

A return statement is used to exit a method and return a value to the calling code. The basic syntax of a return statement in catscript is as follows:

```
return x
```

Where x is the value that is intended to be returned to the calling code.

For example, the following catscript code demonstrates a function call that assigns a returned value to the calling code:

```
function foo(int x) : int {  
    return x + 1  
}  
int z = foo(5)  
print(z)  
Output: 6
```

## Method Instantiation

A method Instantiation is used to define a named block of code that can be called and executed repeatedly throughout a program. The basic syntax of a method declaration in catscript is as follows:

```
function methodName(parameter_list) Optional ': type' {  
    statement  
    Optional return (if required type)  
}
```

Where methodName is the name of the defined method, parameter\_list is the optional list of required method inputs, and statement is the saved functionality of the named method. If a required type is requested by the method signature, there must be a variable of that type returned by the end of the method execution.

For example, the following catscript code demonstrates a method (with 1 integer parameter and required type int) being instantiated and then used in context.

```
function foo(int x) :int {  
    int y = x + 1  
    return y  
}  
foo(3)  
Output: 4
```

## Method Invocation

A method call is used to execute a named block of code that has been defined elsewhere in the program. The basic syntax of a method call in catscript is as follows:

```
methodName(argumentList)
```

Where methodName is the name of the method that was defined elsewhere and argumentList is the list of parameters needed to invoke the method (based on specifications during instantiation)

For example, the following catscript code demonstrates a method definition and its subsequent use.

```
function foo(int x):String {  
    String y = x + " is a number"  
    return y  
}  
foo(3)  
Output: 3 is a number
```

## Types

The available types within Catscript include:

- int – 32-bit integer
- string - a java-style string
- bool - a true/false boolean value
- list<x> - a list of value with the type 'x'
- null - the null type
- object – generic parent type

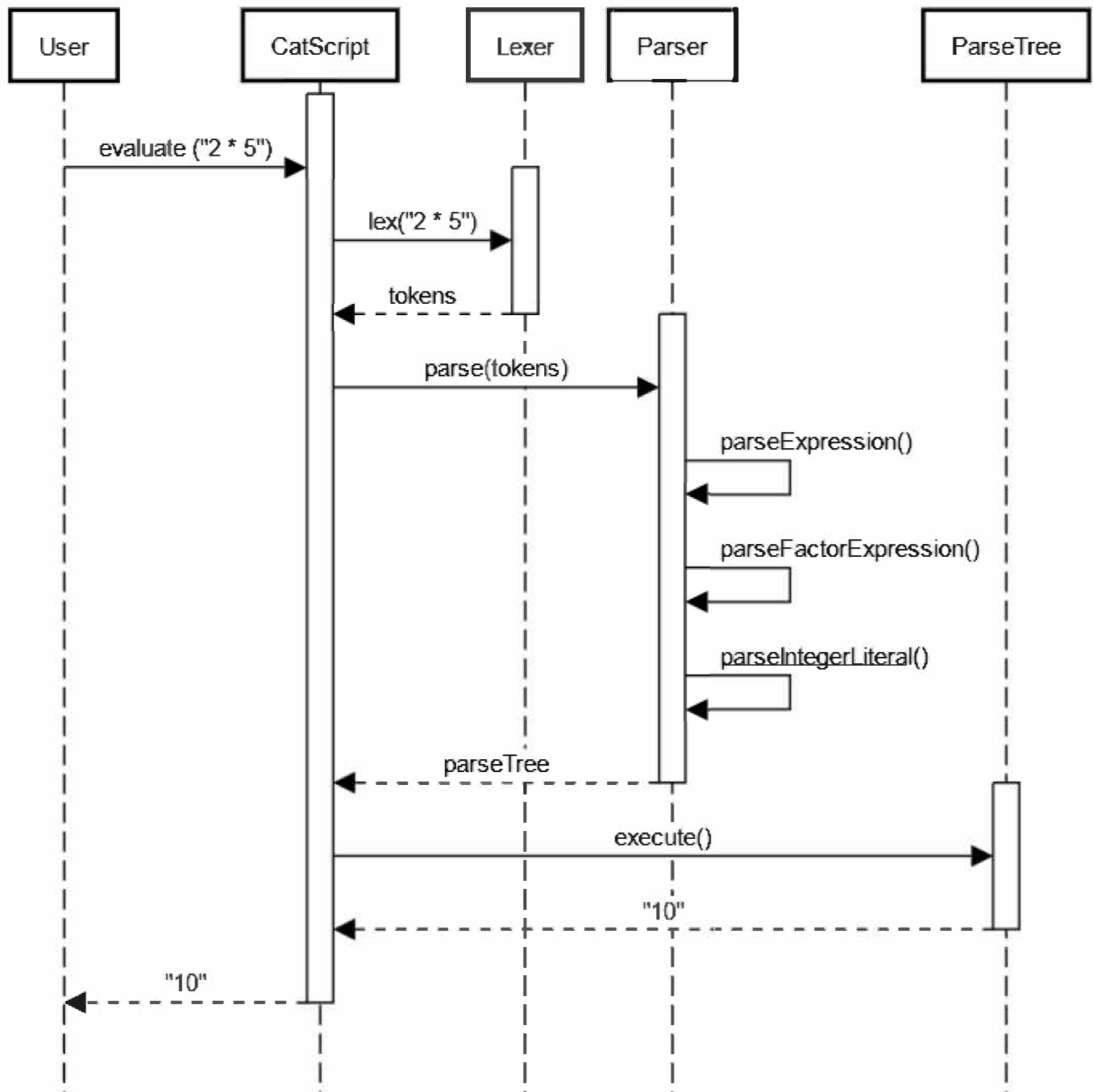
## Operators

- +
- -
- \*
- /
- ==
- !=
- <=
- >=
- >
- <
- not

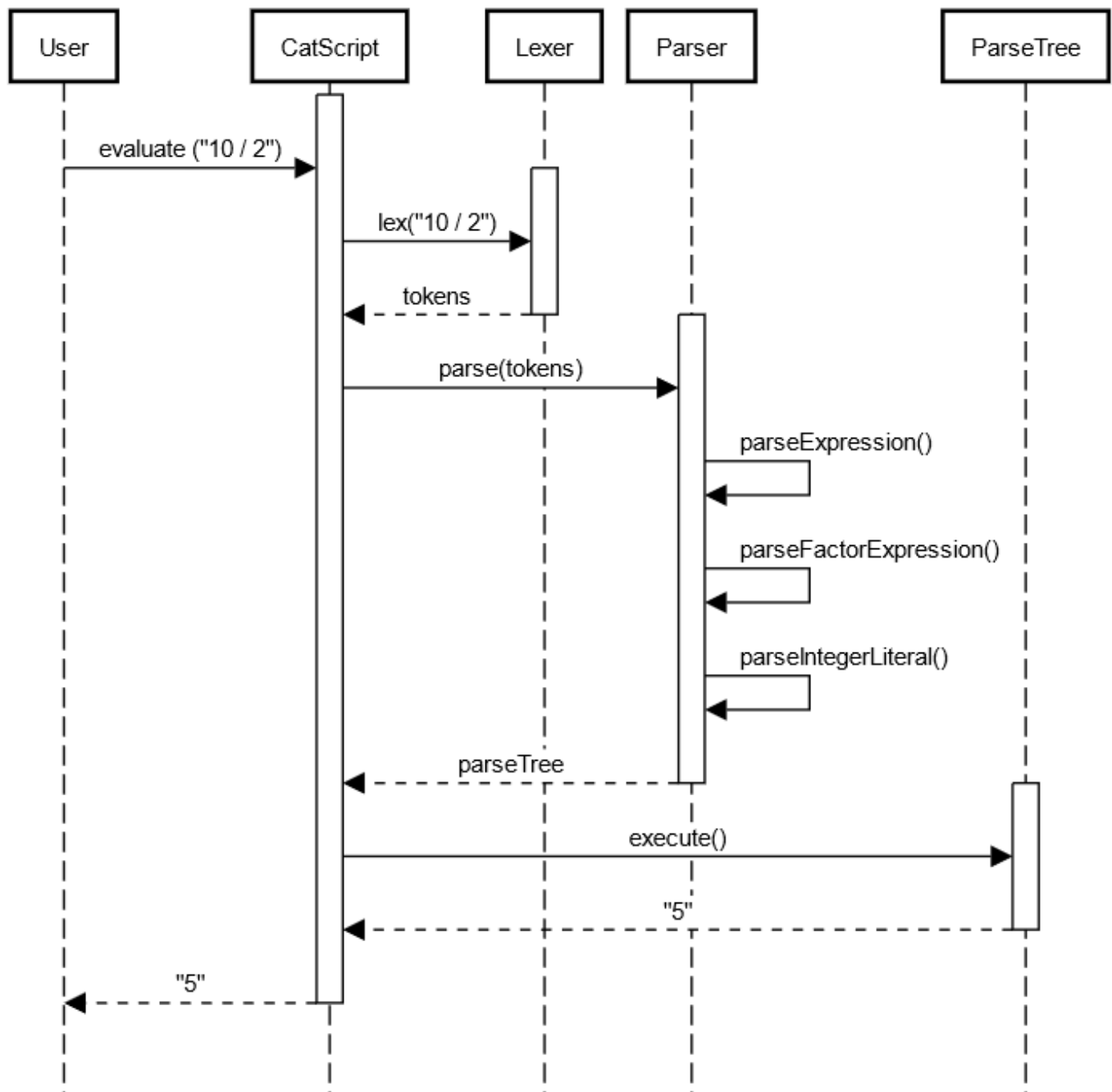
## Section 5: UML.



## Catscript Multiplication Sequence Diagram



## Catscript Division Sequence Diagram



## Section 6: Design trade-offs

We implemented a parser by hand using recursive descent rather than using a parser generator. The benefit of this is that we are able to understand the Catscript grammar far better than we would have if we generated it. Debugging makes much more sense as the functions have sensible names, are in logical places, and overall are more human-readable. Furthermore, the amount of code necessary is far less when implementing recursive descent by hand. The only real down side to this manual development is that we spend more time up front creating the parser. While we initially save time using a parser generator, we would eventually lose

this benefit when we inevitably need to debug. Implementing a recursive descent parser by hand allowed us to get familiar with the grammar and with the system in general. This ultimately makes debugging much easier and saves us time in the long run.

## Section 7: Software development life cycle model

*Describe the model that you used to develop your capstone project. How did this model help and/or hinder your team?*

We used Test Driven Development (TDD) for this project, meaning that the software was developed by creating and utilizing tests to produce quality software. This model was extremely successful in the fact that we understood what needed to be implemented at all times and we could be constantly checking to see if our software was working as intended. This project definitely wouldn't have been as successful if development wasn't driven by tests.