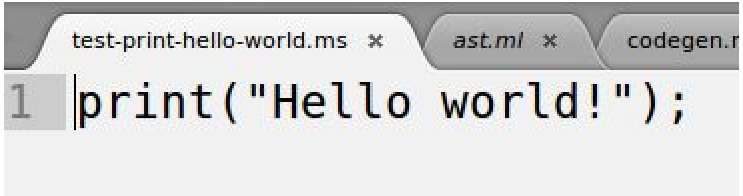
MatchaScript

“Like JavaScript, but better for you.”

Language Guru: Kimberly Hou - kjh2146
Systems Architect: Rebecca Mahany - rlm2175
Manager: Jorge Orbay - jao2154
Tester: Rachel Yang - ry2277

Overview on MatchaScript: motivations

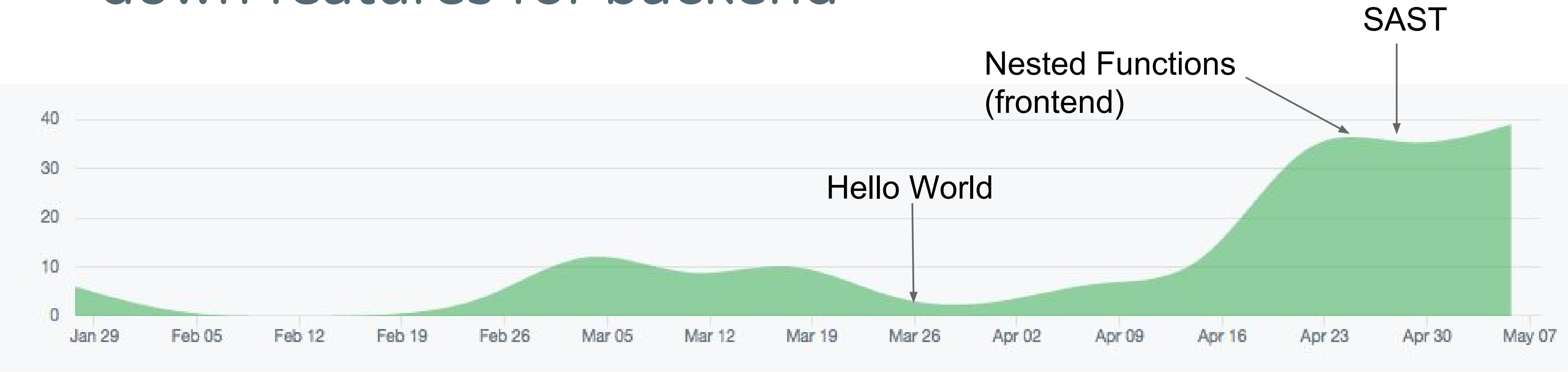
- MatchaScript is a general-purpose statically typed programming language that is convenient for both imperative and functional programming
- The syntax of MatchaScript can be described as “JavaScript, but with type specifications”
- No main method required



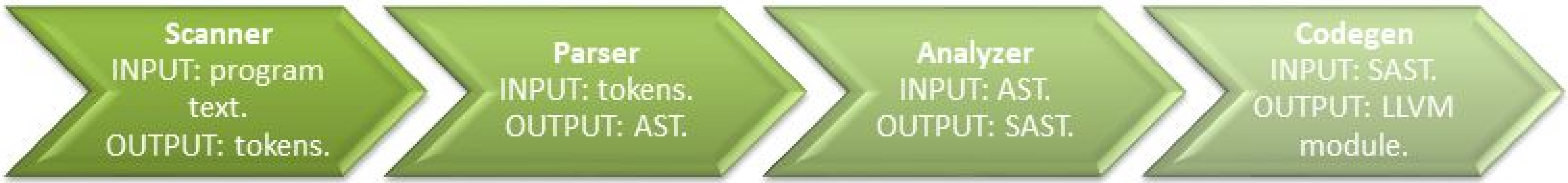
```
print("Hello world!");
```

MatchaScript on GitHub

- https://github.com/RebeccaMahany/MatchaScript
- 220+ commits to master
- Process: Hello World, full-features front-end, pare down features for backend



Architecture overview



Scanner
INPUT: program text.
OUTPUT: tokens.

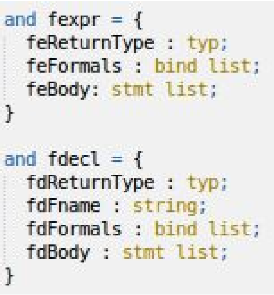
Parser
INPUT: tokens.
OUTPUT: AST.

Analyzer
INPUT: AST.
OUTPUT: SAST.

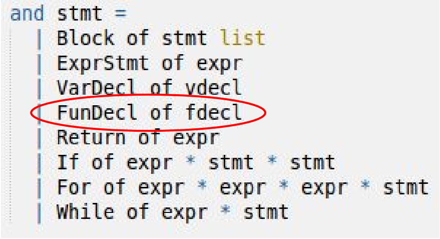
Codegen
INPUT: SAST.
OUTPUT: LLVM module.

Interesting features: Nested functions

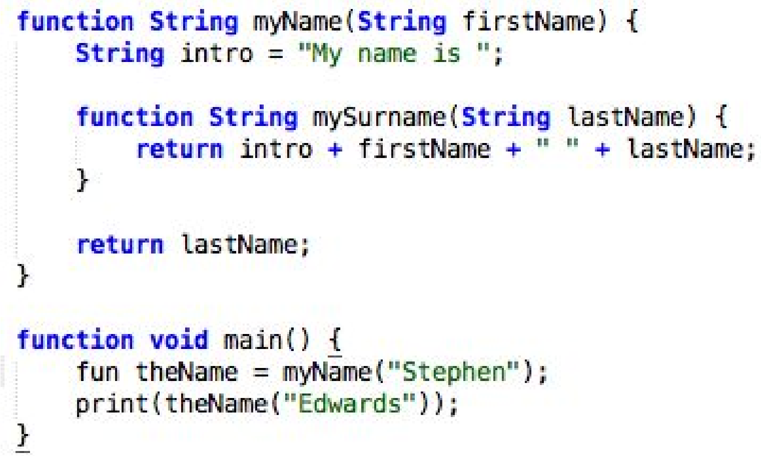
- Currently implemented through scanner, parser, AST, and semantic checking; codegen in progress
- Based off of JavaScript’s use of closures, where inner functions can access their parent and ancestors’ scope



```
and fexpr = {
  feReturnType : typ;
  feFormals : bind list;
  feBody: stmt list;
}

and fdecl = {
  fdReturnType : typ;
  fdFname : string;
  fdFormals : bind list;
  fdBody : stmt list;
}
```



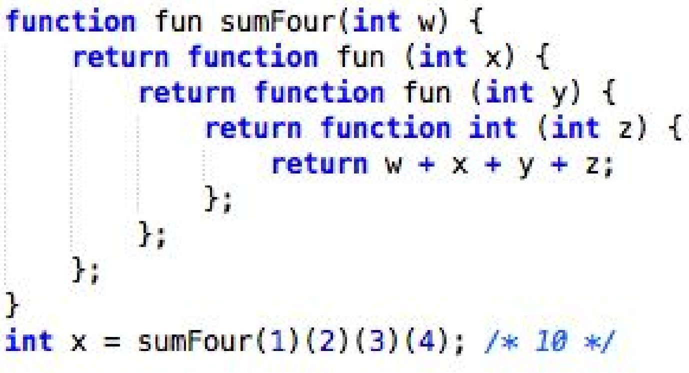
```
and stmt =
  | Block of stmt list
  | ExprStmt of expr
  | VarDecl of vdecl
  | FunDecl of fdecl
  | Return of expr
  | If of expr * stmt * stmt
  | For of expr * expr * expr * stmt
  | While of expr * stmt
```



```
function String myName(String firstName) {
    String intro = "My name is ";

    function String mySurname(String lastName) {
        return intro + firstName + " " + lastName;
    }

    return lastName;
}

function void main() {
    fun theName = myName("Stephen");
    print(theName("Edwards"));
}
```

Interesting features: Currying

- Currently implemented through scanner, parser, AST, and semantic checking
- As part of currying, use of anonymous functions also supported



```
function fun sumFour(int w) {
    return function fun (int x) {
        return function fun (int y) {
            return function int (int z) {
                return w + x + y + z;
            };
        };
    };
}
int x = sumFour(1)(2)(3)(4); /* 10 */
```

Standard Library

- We implemented a basic standard library based on common functions available in JavaScript and other object-oriented languages
- Right now, mostly math functions for both floats and integers: pow, ceil, floor, round, min, max, abs
- Automatically included in all .ms files during code generation

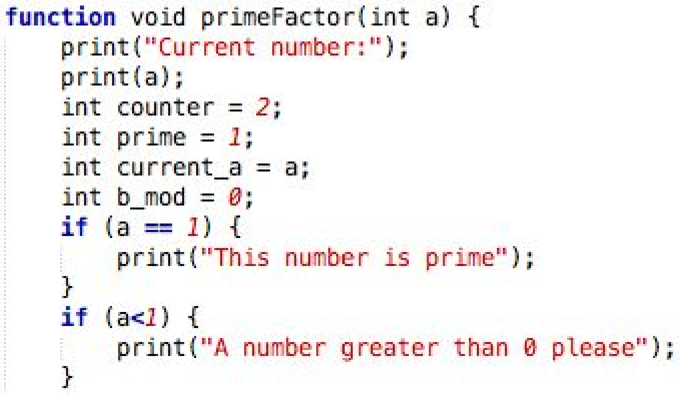
Test Suite

- **test-frontend.sh**: For each test case:
  - Pretty-print the AST generated for a tests/test-<filename>.ms file
  - Run scannerprint.ml (generate tokens from program text) on both:
    - The original tests/test-<filename>.ms file
    - The pretty-printed AST
  - If the two token files match, the AST was generated properly and the AST pretty-printer works
- **test-all.sh**
  - Fail tests
    - Tests error-identification in analyzer.ml
  - Pass tests
    - Tests proper code generation

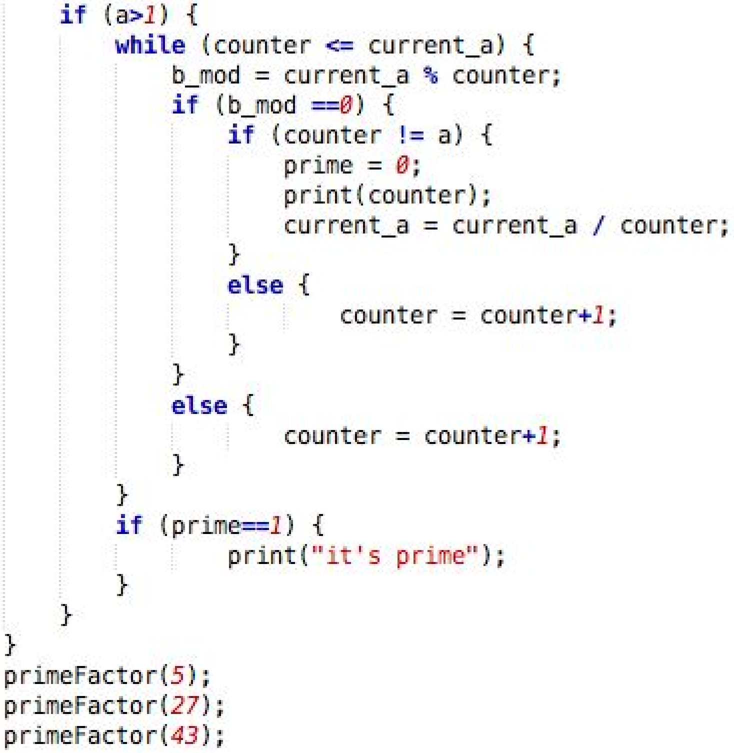
Lessons learned

- Kimberly: Listen to Prof. Edwards and focus on building the entire compiler at the same time, even if it means re-doing some parts when adding in the next feature.
- Becca: Pick realistic goals and start early.
- Jordi: Have a flexible battle plan.
- Rachel: Write a **good** outline of the code components, and specify interfaces (AST, SAST) **early**. (e.g. by specifying SAST, one group member can work on Analyzer and another can work on Codegen at the same time). Also, don't get hung up on one feature (nested functions).

Demo of MatchaScript: Prime Factorization

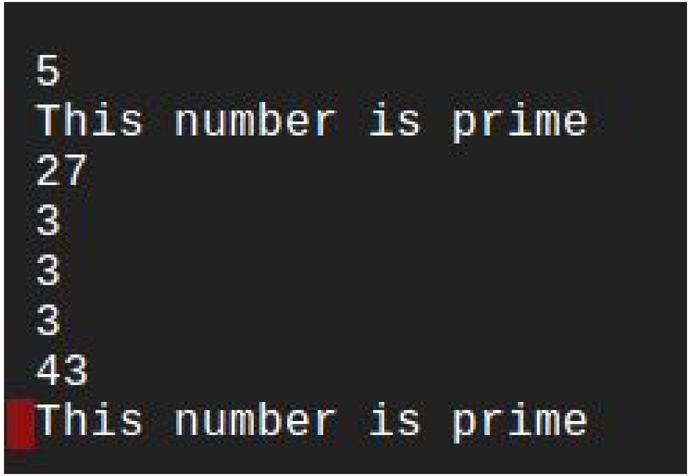


```
function void primeFactor(int a) {
    print("Current number:");
    print(a);
    int counter = 2;
    int prime = 1;
    int current_a = a;
    int b_mod = 0;
    if (a == 1) {
        print("This number is prime");
    }
    if (a<1) {
        print("A number greater than 0 please");
    }
```



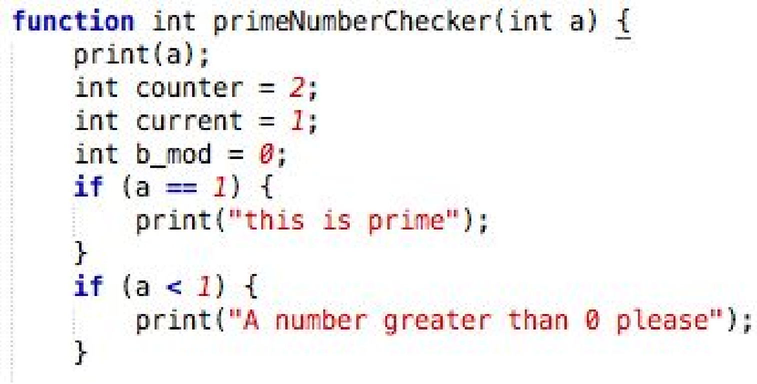
```
    if (a>1) {
        while (counter <= current_a) {
            b_mod = current_a % counter;
            if (b_mod ==0) {
                if (counter != a) {
                    prime = 0;
                    print(counter);
                    current_a = current_a / counter;
                }
                else {
                        counter = counter+1;
                }
            }
            else {
                    counter = counter+1;
            }
        }
        if (prime==1) {
                print("it's prime");
        }
    }
}
primeFactor(5);
primeFactor(27);
primeFactor(43);
```

Demo of MatchaScript: Prime Factorization Results

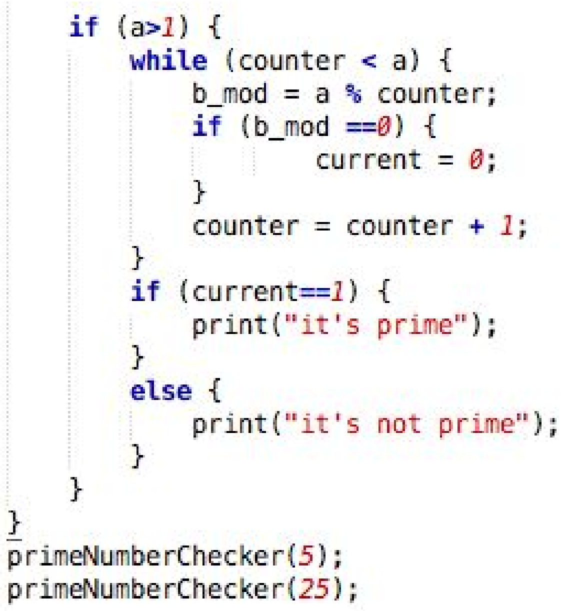


```
5
This number is prime
27
3
3
3
43
This number is prime
```

Demo: Prime Number Checker



```
function int primeNumberChecker(int a) {
    print(a);
    int counter = 2;
    int current = 1;
    int b_mod = 0;
    if (a == 1) {
        print("this is prime");
    }
    if (a < 1) {
        print("A number greater than 0 please");
    }
```



```
    if (a>1) {
        while (counter < a) {
            b_mod = a % counter;
            if (b_mod ==0) {
                current = 0;
            }
            counter = counter + 1;
        }
        if (current==1) {
            print("it's prime");
        }
        else {
            print("it's not prime");
        }
    }
}
primeNumberChecker(5);
primeNumberChecker(25);
```