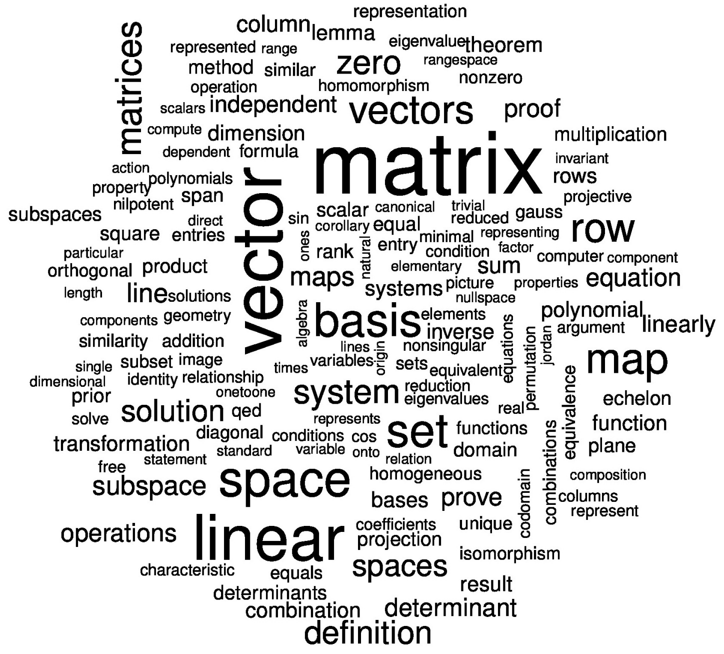
LFLA

LANGUAGE FOR LINEAR ALGEBRA



TEAM OF GEEKS

- **Zhiyuan Guo (Architect)**
  - Compiler, Code generation, Semantics
- **Guitang Lan (Language Guru)**
  - Compiler, Semantic validation, Test case creation
- **Jin Liang (Tester)**
  - Test case creation, Testing automation, Documentation
- **Chenzhe Qian (Manager)**
  - Python libraries, Code generation, Documentation

INTRODUCTION

- Is Vector same as Matrix?
- What is Vector Space?
- Why on earth need Matrix?

Why not MATLAB?

How about Python and others?

- Set vector EQUALS matrix
- Mixed math concepts with data structures

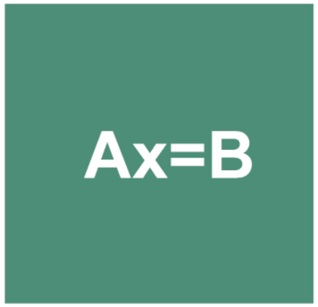
GOAL

Math education

Linear algebra programming

Solve X with real math language

in computer!



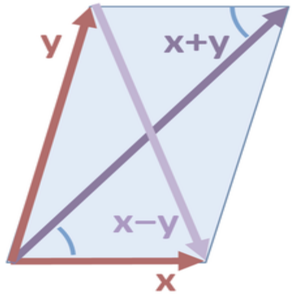
TYPES

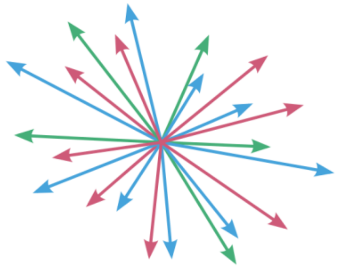
Featured Primitive Types

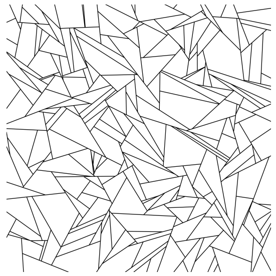
- **vector**
- **matrix**
- **vecspace**
- **inspace**
- **affspace**

Common Primitive Type

- **var**







DECLARATIONS

```
var a  = 1.2

vector b  = [1,2]

matrix  c =  [ 1,2;2,8;]

vecspace d = L( [1,2],[3,4])

inspace e = inspace( {[1,0],[0,1]},  c)

affspace  f =  affspace( b, d)
```

FEATURED OPERATORS

- **Belongs @:**
  - vector@ vecspace (affspace)
- **LieBracket [[,]]**
  - [[matrix, matrix]]
- **Innerproduct << , >>**
  - id<<vector,vector>>
- **Matrix action &**
  - matrix & vector

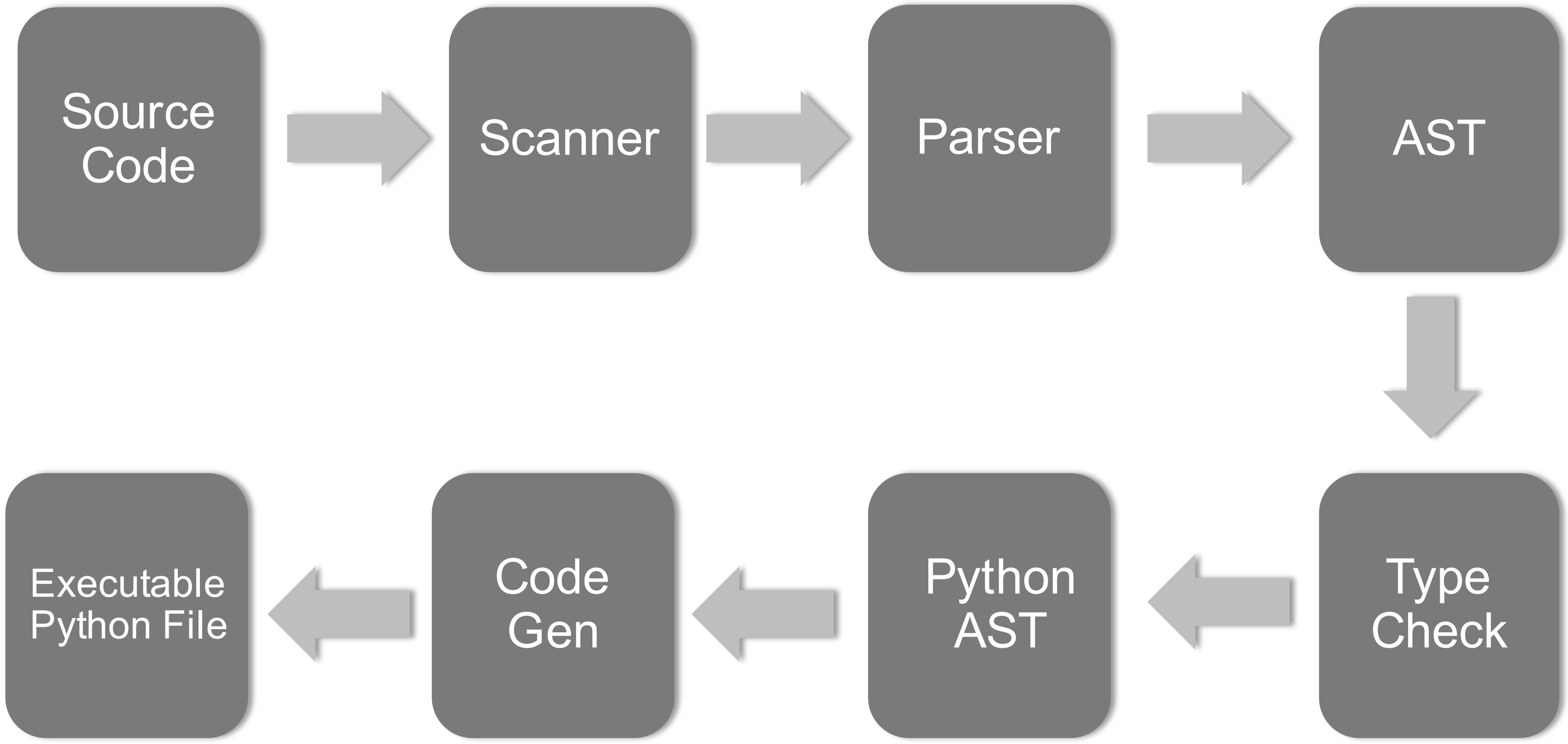
FEATURED BUILT-IN FUNCTION

- **dim(argument)**
  - argument is vector, vecspace, inspace or affspace

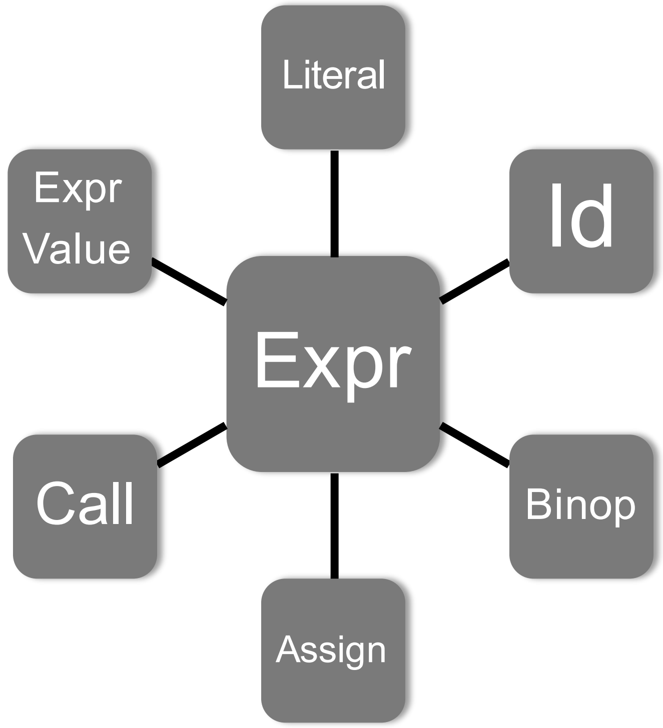
- **basis(vecspace)**
  - return a basis of the vecspace

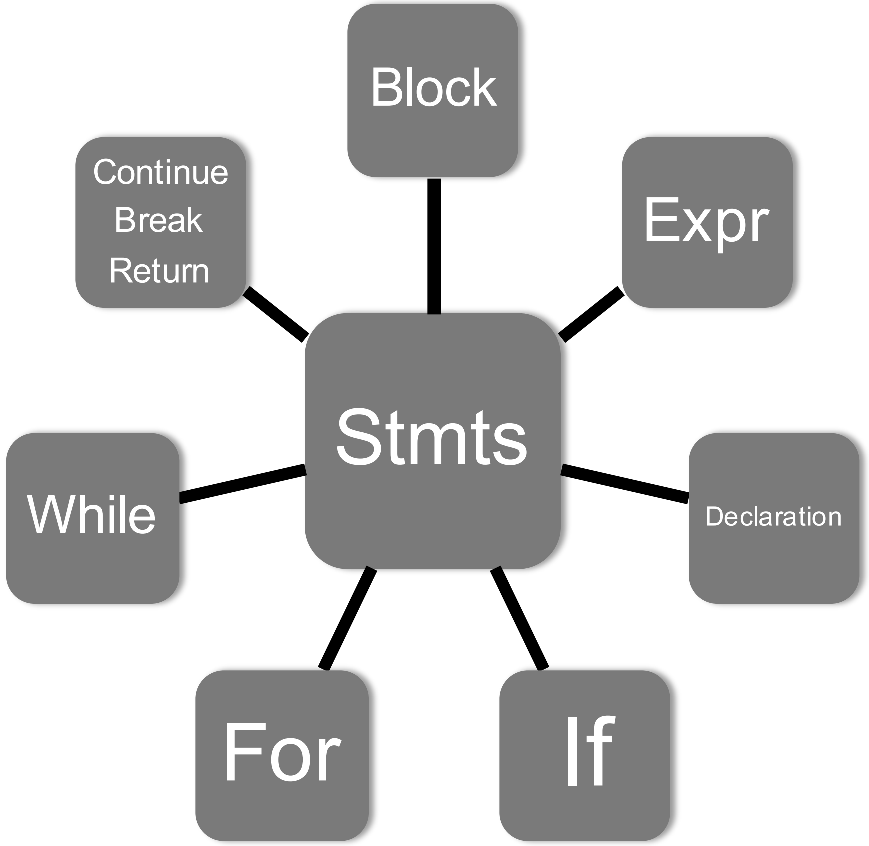
- **solve(matrix, vector)**
  - return an affspace
  - the solution set of linear equation m * x = v

ARCHITECTURE



AST


Literal
Expr Value
Id
Expr
Call
Binop
Assign


Block
Continue Break Return
Expr
Stmts
While
Declaration
For
If

TRANSLATION ENVIRONMENT

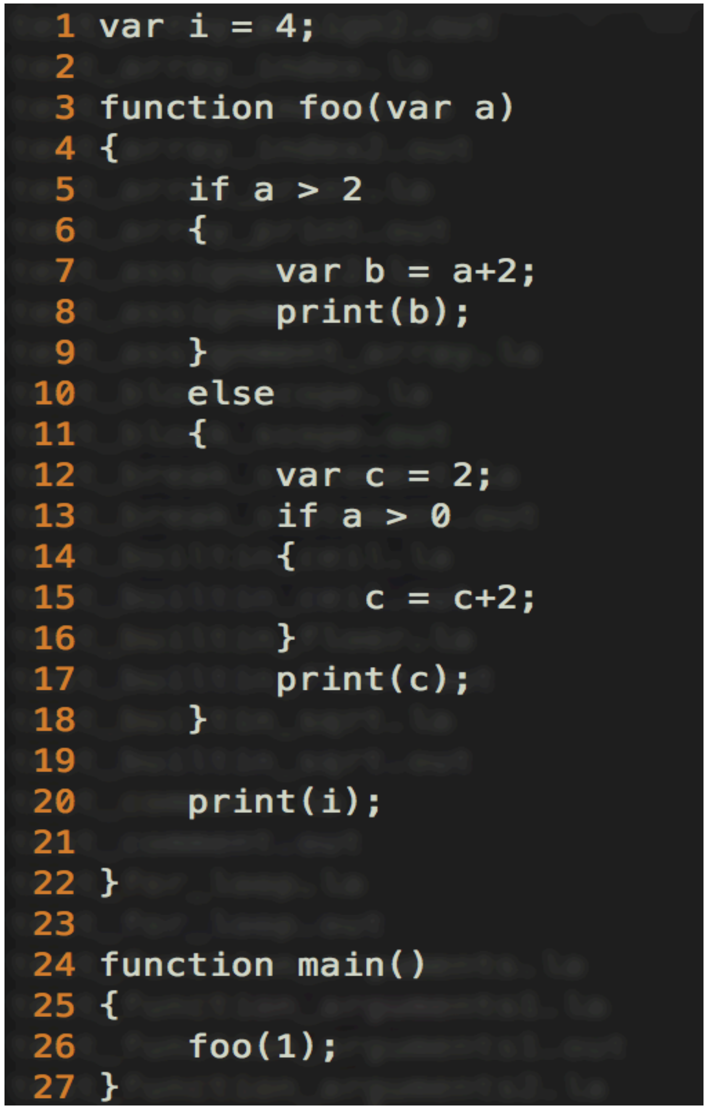
- **Scope: symbol_table**
- **Functions: func StringMap.t**
- **Global_Variables: var StringMap.t**
- **In_While: bool**
- **In_For: bool**

symbol_table:

- **Parent : symbol_table**
- **Vars : var StringMap.t**

SCOPE RULE

- **Global Variable**
  - from defined to end
- **Function Parameters**
  - inside function
- **Local variables**
  - valid inside the nearest block



```
var i = 4;

function foo(var a)
{
    if a > 2
    {
        var b = a+2;
        print(b);
    }
    else
    {
        var c = 2;
        if a > 0
        {
            c = c+2;
        }
        print(c);
    }

    print(i);

}

function main()
{
    foo(1);
}
```

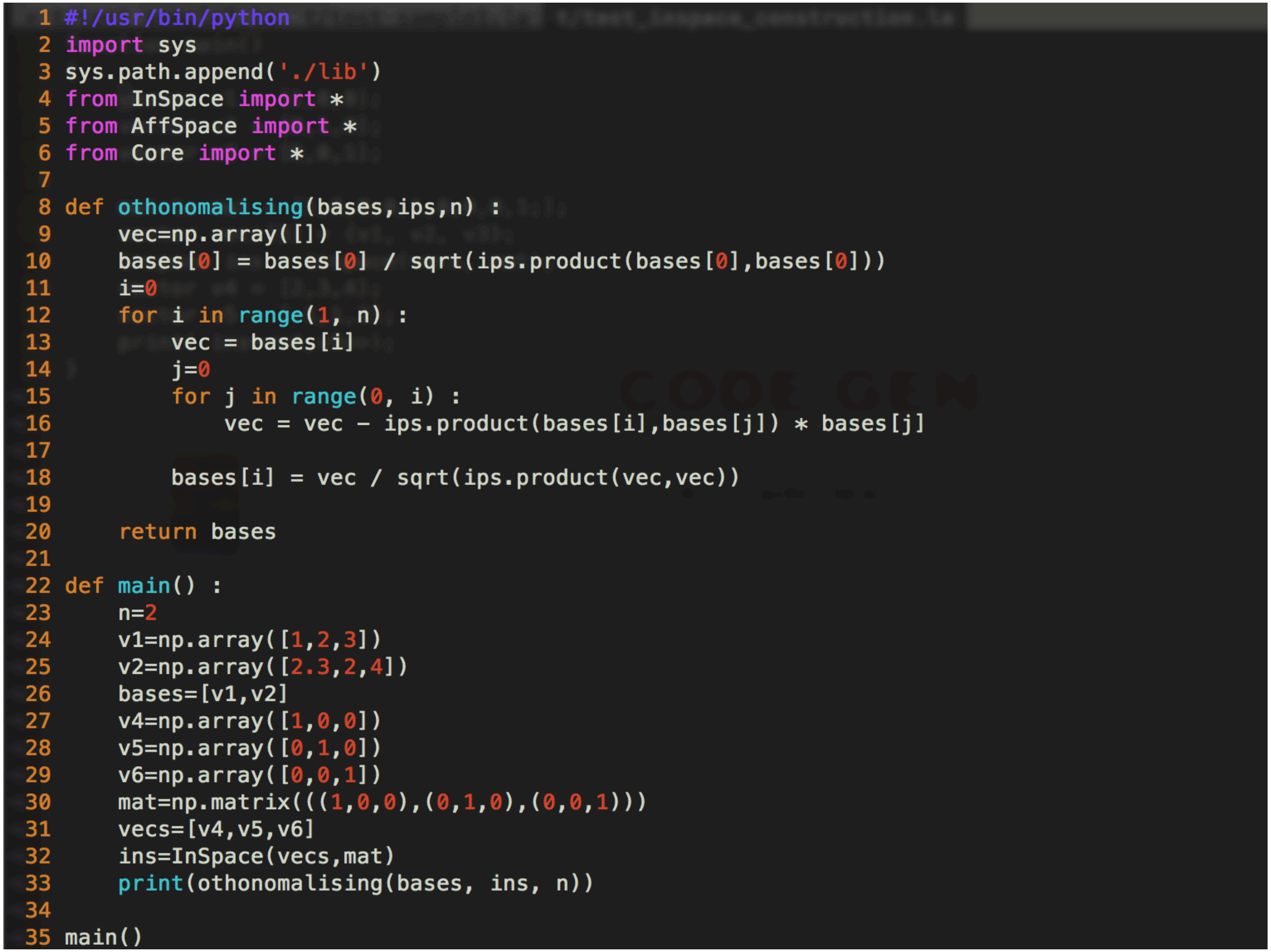
TYPE CHECK

- **Data Type**
  - Variable declaration and assign
  - Array declaration and element assign
  - Function parameters pass

- **Function return type**
  - Our function definition doesn’t declare function return type explicitly, so compiler check return type.

- **Control flow**
  - Conditional expression type check

CODE GEN



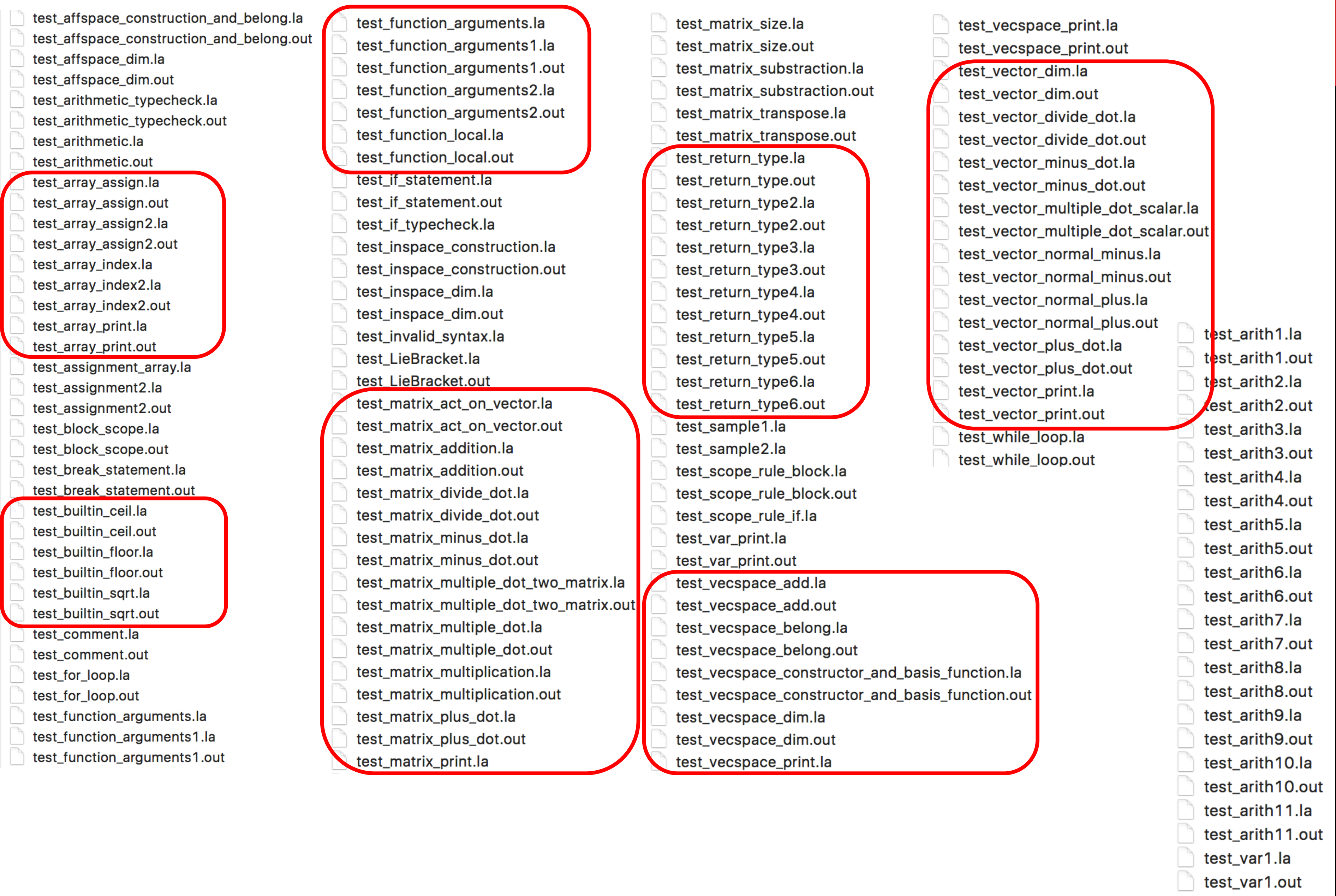
```python
#!/usr/bin/python
import sys
sys.path.append('./lib')
from InSpace import *
from AffSpace import *
from Core import *

def othonomalising(bases,ips,n) :
    vec=np.array([])
    bases[0] = bases[0] / sqrt(ips.product(bases[0],bases[0]))
    i=0
    for i in range(1, n) :
        vec = bases[i]
        j=0
        for j in range(0, i) :
            vec = vec - ips.product(bases[i],bases[j]) * bases[j]

        bases[i] = vec / sqrt(ips.product(vec,vec))

    return bases

def main() :
    n=2
    v1=np.array([1,2,3])
    v2=np.array([2.3,2,4])
    bases=[v1,v2]
    v4=np.array([1,0,0])
    v5=np.array([0,1,0])
    v6=np.array([0,0,1])
    mat=np.matrix(((1,0,0),(0,1,0),(0,0,1)))
    vecs=[v4,v5,v6]
    ins=InSpace(vecs,mat)
    print(othonomalising(bases, ins, n))

main()
```

TESTING

- Test Suites
- Testing Cases
- Automation Testing
- Test Roles

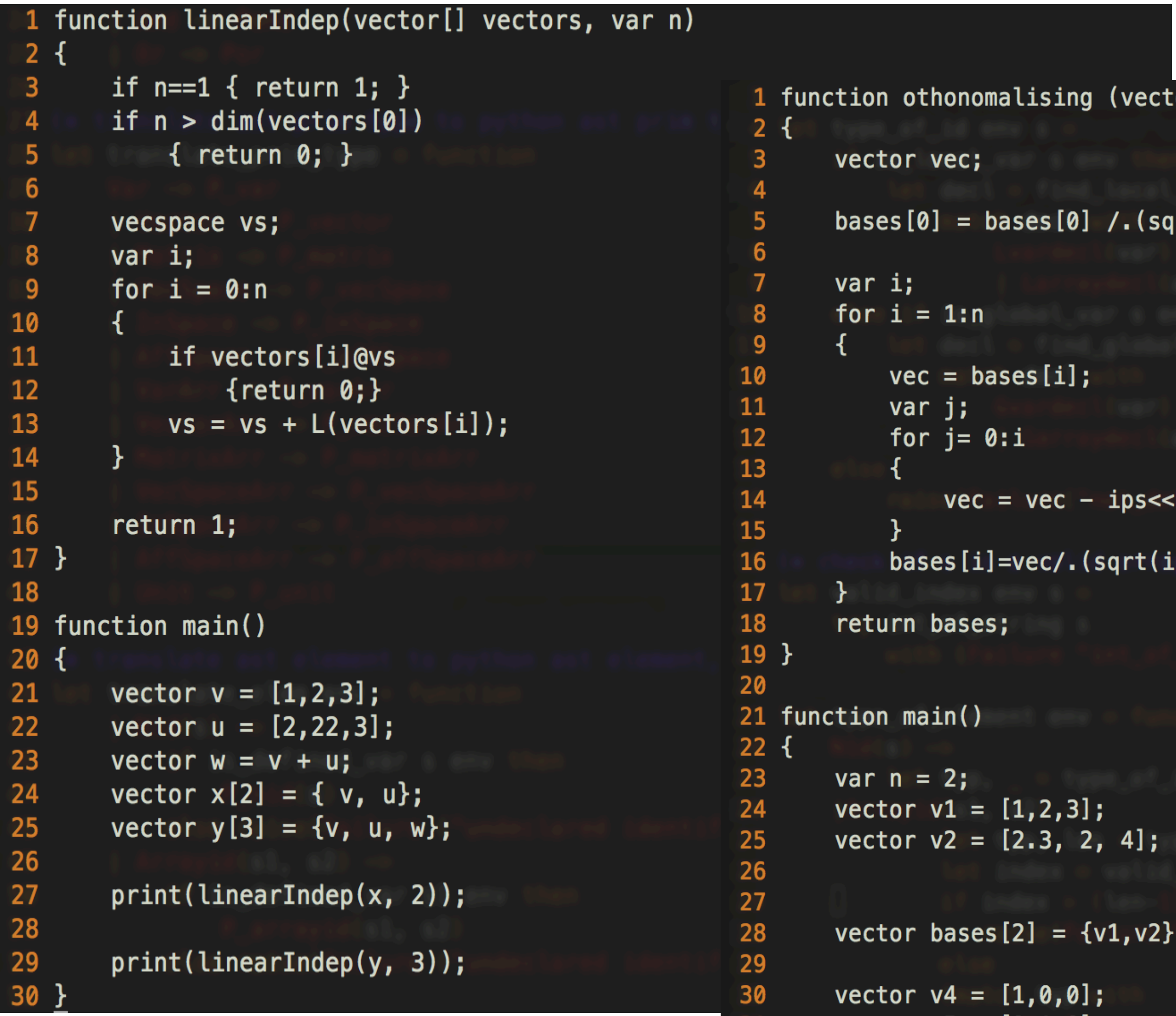
AUTOMATION TESTING

- Fast feedback to the team
- Free up time
- A sense of confidence
- An automated script

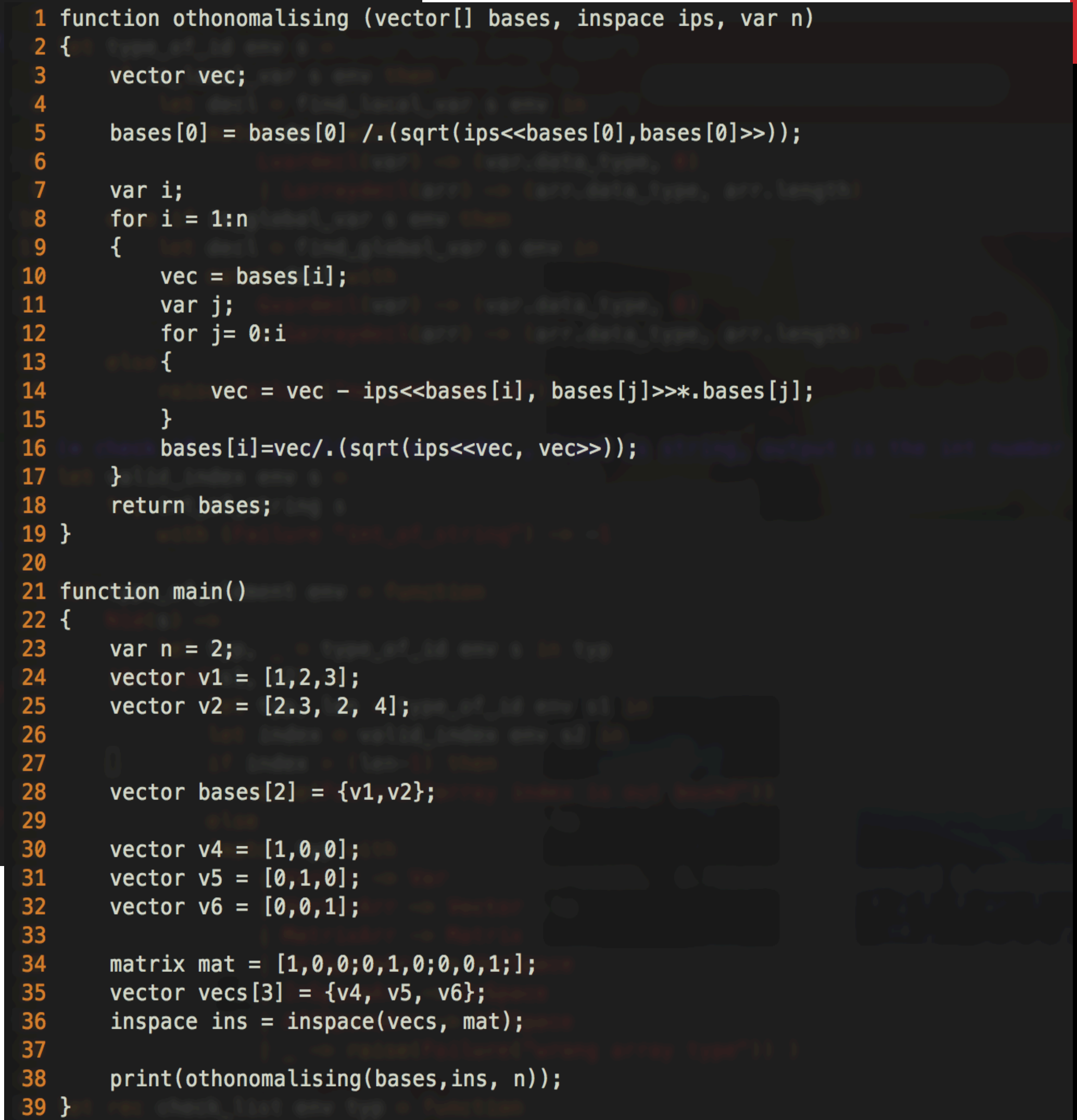


```
test_affspace_construction_and_belong.la
test_affspace_construction_and_belong.out
test_affspace_dim.la
test_affspace_dim.out
test_arithmetic_typecheck.la
test_arithmetic_typecheck.out
test_arithmetic.la
test_arithmetic.out
test_array_assign.la
test_array_assign.out
test_array_assign2.la
test_array_assign2.out
test_array_index.la
test_array_index2.la
test_array_index2.out
test_array_print.la
test_array_print.out
test_assignment_array.la
test_assignment2.la
test_assignment2.out
test_block_scope.la
test_block_scope.out
test_break_statement.la
test_break_statement.out
test_builtin_ceil.la
test_builtin_ceil.out
test_builtin_floor.la
test_builtin_floor.out
test_builtin_sqrt.la
test_builtin_sqrt.out
test_comment.la
test_comment.out
test_for_loop.la
test_for_loop.out
test_function_arguments.la
test_function_arguments1.la
test_function_arguments1.out

test_function_arguments.la
test_function_arguments1.la
test_function_arguments1.out
test_function_arguments2.la
test_function_arguments2.out
test_function_local.la
test_function_local.out
test_if_statement.la
test_if_statement.out
test_if_typecheck.la
test_inspace_construction.la
test_inspace_construction.out
test_inspace_dim.la
test_inspace_dim.out
test_invalid_syntax.la
test_LieBracket.la
test_LieBracket.out
test_matrix_act_on_vector.la
test_matrix_act_on_vector.out
test_matrix_addition.la
test_matrix_addition.out
test_matrix_divide_dot.la
test_matrix_divide_dot.out
test_matrix_minus_dot.la
test_matrix_minus_dot.out
test_matrix_multiple_dot_two_matrix.la
test_matrix_multiple_dot_two_matrix.out
test_matrix_multiple_dot.la
test_matrix_multiple_dot.out
test_matrix_multiplication.la
test_matrix_multiplication.out
test_matrix_plus_dot.la
test_matrix_plus_dot.out
test_matrix_print.la

test_matrix_size.la
test_matrix_size.out
test_matrix_substraction.la
test_matrix_substraction.out
test_matrix_transpose.la
test_matrix_transpose.out
test_return_type.la
test_return_type.out
test_return_type2.la
test_return_type2.out
test_return_type3.la
test_return_type3.out
test_return_type4.la
test_return_type4.out
test_return_type5.la
test_return_type5.out
test_return_type6.la
test_return_type6.out
test_sample1.la
test_sample2.la
test_scope_rule_block.la
test_scope_rule_block.out
test_scope_rule_if.la
test_var_print.la
test_var_print.out
test_vecspace_add.la
test_vecspace_add.out
test_vecspace_belong.la
test_vecspace_belong.out
test_vecspace_constructor_and_basis_function.la
test_vecspace_constructor_and_basis_function.out
test_vecspace_dim.la
test_vecspace_dim.out
test_vecspace_print.la

test_vecspace_print.la
test_vecspace_print.out
test_vector_dim.la
test_vector_dim.out
test_vector_divide_dot.la
test_vector_divide_dot.out
test_vector_minus_dot.la
test_vector_minus_dot.out
test_vector_multiple_dot_scalar.la
test_vector_multiple_dot_scalar.out
test_vector_normal_minus.la
test_vector_normal_minus.out
test_vector_normal_plus.la
test_vector_normal_plus.out
test_vector_plus_dot.la
test_vector_plus_dot.out
test_vector_print.la
test_vector_print.out
test_while_loop.la
test_while_loop.out

test_arith1.la
test_arith1.out
test_arith2.la
test_arith2.out
test_arith3.la
test_arith3.out
test_arith4.la
test_arith4.out
test_arith5.la
test_arith5.out
test_arith6.la
test_arith6.out
test_arith7.la
test_arith7.out
test_arith8.la
test_arith8.out
test_arith9.la
test_arith9.out
test_arith10.la
test_arith10.out
test_arith11.la
test_arith11.out
test_var1.la
test_var1.out
```

EXAMPLES

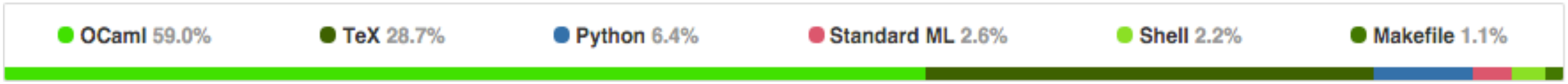


```
function linearIndep(vector[] vectors, var n)
{
    if n==1 { return 1; }
    if n > dim(vectors[0])
        { return 0; }

    vecspace vs;
    var i;
    for i = 0:n
    {
        if vectors[i]@vs
            {return 0;}
        vs = vs + L(vectors[i]);
    }

    return 1;
}

function main()
{
    vector v = [1,2,3];
    vector u = [2,22,3];
    vector w = v + u;
    vector x[2] = { v, u};
    vector y[3] = {v, u, w};

    print(linearIndep(x, 2));

    print(linearIndep(y, 3));
}
```

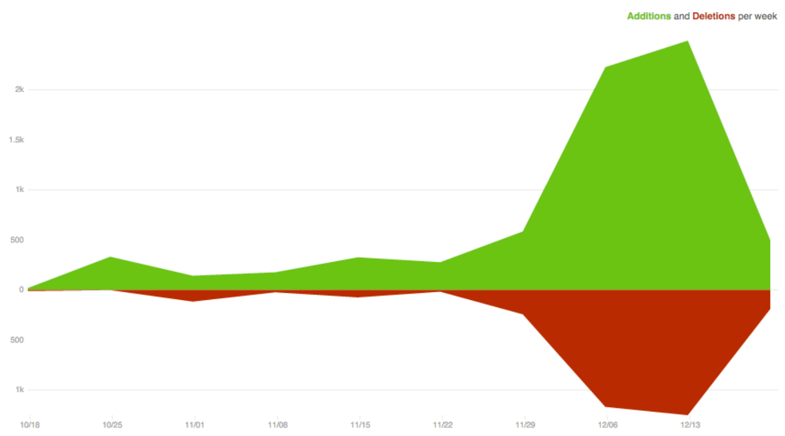


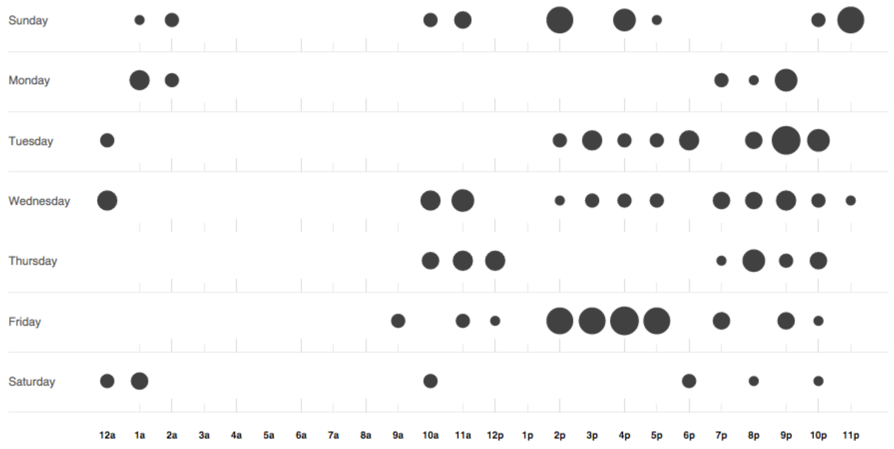
```
function othonomalising (vector[] bases, inspace ips, var n)
{
    vector vec;

    bases[0] = bases[0] /.(sqrt(ips<<bases[0],bases[0]>>));

    var i;
    for i = 1:n
    {
        vec = bases[i];
        var j;
        for j= 0:i
        {
            vec = vec - ips<<bases[i], bases[j]>>*.bases[j];
        }
        bases[i]=vec/.(sqrt(ips<<vec, vec>>));
    }
    return bases;
}

function main()
{
    var n = 2;
    vector v1 = [1,2,3];
    vector v2 = [2.3, 2, 4];


    vector bases[2] = {v1,v2};

    vector v4 = [1,0,0];
    vector v5 = [0,1,0];
    vector v6 = [0,0,1];

    matrix mat = [1,0,0;0,1,0;0,0,1;];
    vector vecs[3] = {v4, v5, v6};
    inspace ins = inspace(vecs, mat);

    print(othonomalising(bases,ins, n));
}
```

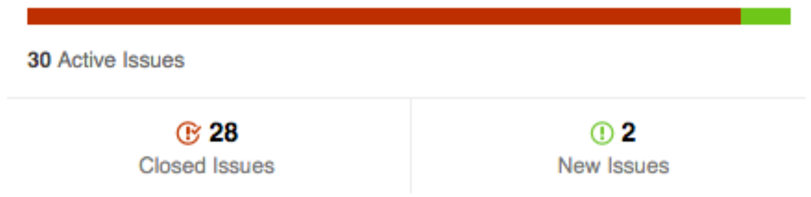
HOW WE WORKED



● OCaml 59.0% ● TeX 28.7% ● Python 6.4% ● Standard ML 2.6% ● Shell 2.2% ● Makefile 1.1%



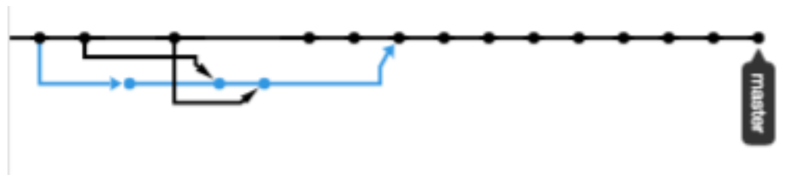




30 Active Issues

28 Closed Issues

2 New Issues



LESSONS LEARNED

- Start EARLY
- Meet regularly
- Plan ahead
- Communication
- Collaboration is key! (Github, Google Drive, WeChat)

DEMO

THANK YOU