# PLOTTER

"Plots on the go"

Ibrahima Niang

Ranjith Kumar S.

Sania Arif

PLOTTER

## In a nutshell

01

### C++

Our backend compiles down to C++, allowing us to re-use code without re-compiling

02

### Built-in function Libraries

Built-In functions written and compiled in our own language, have the ability to be included as libraries

03

### Lists

We have lists for primitive as well as non-primitive data types such as point, string

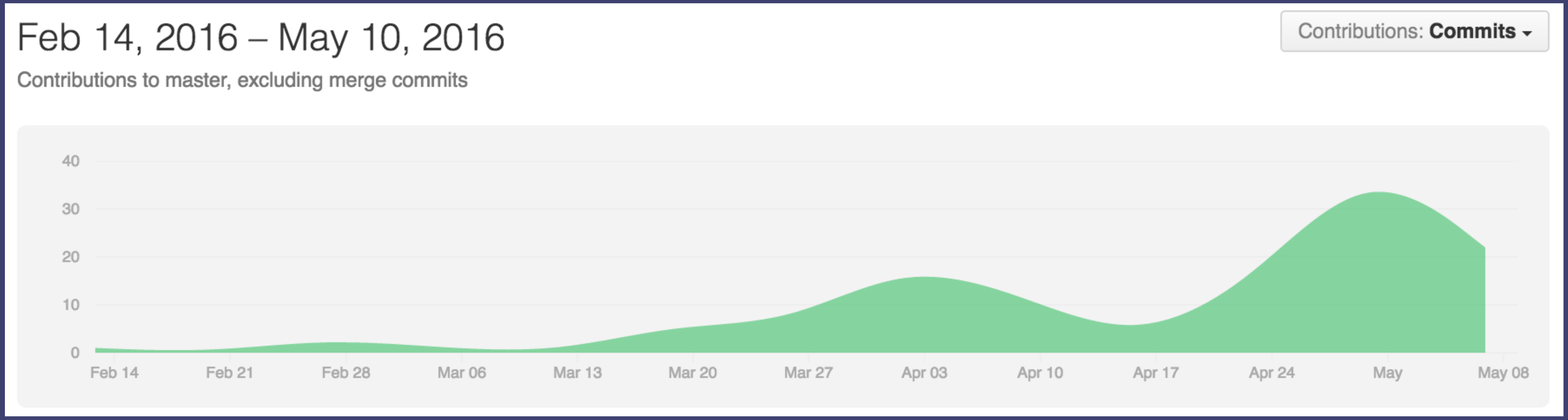
04

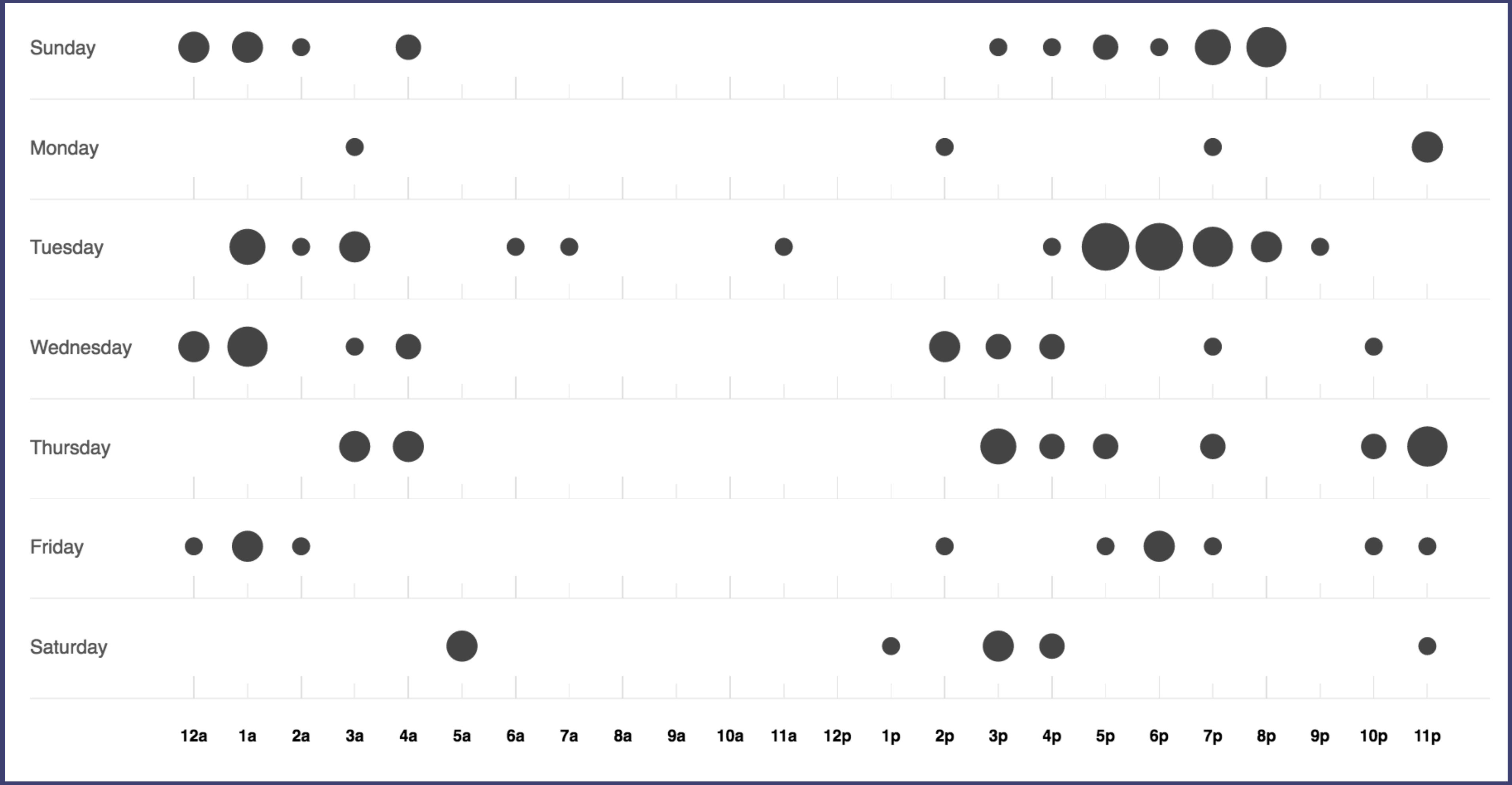
### Useful Error Messages

Every single error that the compiler encounters will print a rich error message for the user

# Project Management

GitHub 175 commits, 3 branches, 25 issues, 2500+ lines of code





# Syntax

Let’s talk PlOtter

## Primitive data types

```
num a
a = 5
a = 3.14

bool b
b = true
b = false

string s
s = "hey"
s = "how are you?"

point p
p = (500, 3.14)
```

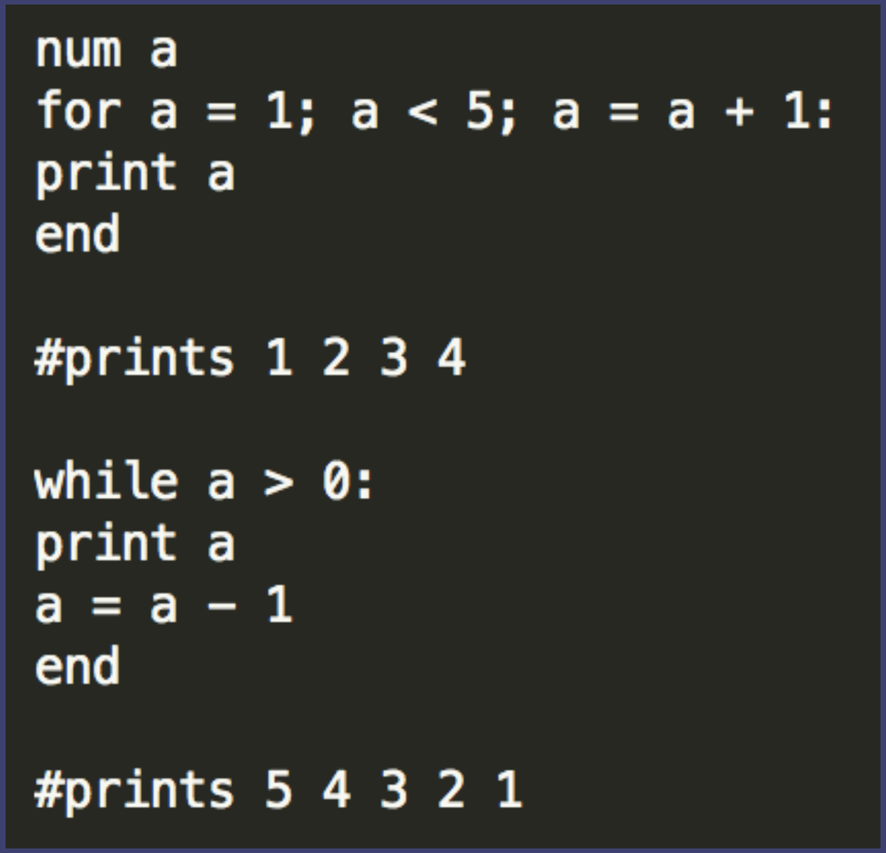
## Lists

```
list num a
a = [6, 9.7, 5]

print a.length() #prints 3

a.append(1) #a = [6, 9.7, 5, 1]

a.remove(1) #a = [6, 5, 1]

a.pop() #a = [6, 5]

print a.length() #prints 2
```

## Comments

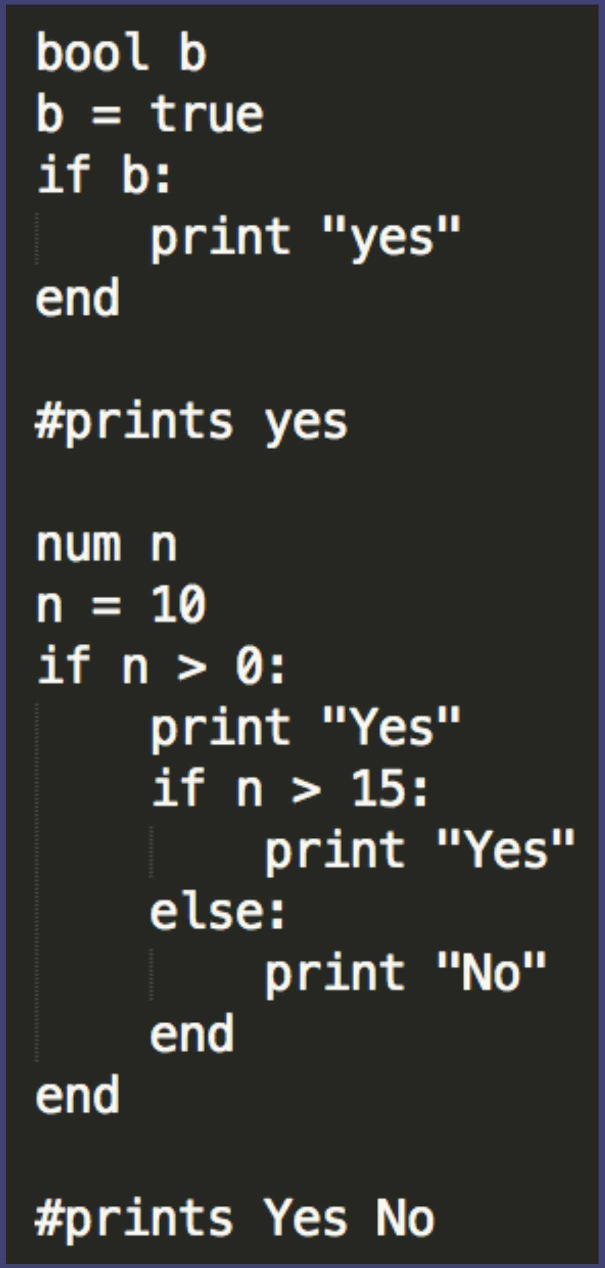
```
# This is a comment

# For multiple lines:

/*
    You can do this,
    instead.
*/
```

## Loops



```
num a
for a = 1; a < 5; a = a + 1:
print a
end

#prints 1 2 3 4

while a > 0:
print a
a = a - 1
end

#prints 5 4 3 2 1
```

## Conditionals



```
bool b
b = true
if b:
    print "yes"
end

#prints yes

num n
n = 10
if n > 0:
    print "Yes"
    if n > 15:
        print "Yes"
    else:
        print "No"
    end
end

#prints Yes No
```

## Built-in functions: Primitive

### Line

```
point p, q
p = (3.14, 3.14)
q = (314, 314)

line(p, q)
#draws a line from point p to q

line((0, 0), (100, 100))
#draws a line from (0,0) to (100,100)
```

Our building block

### Print

```
print "Hi" #prints Hi
print 5    #prints 5
print true #prints true
```

### PrintXY

```
point p
p = (10, 10)
printXY ("hello", p)

# prints hello at (10,10)

printXY ("hello", (500, 500))

# prints hello at (500,500)
```

# Libraries

To use:

```
include plots
```

## Rectangle

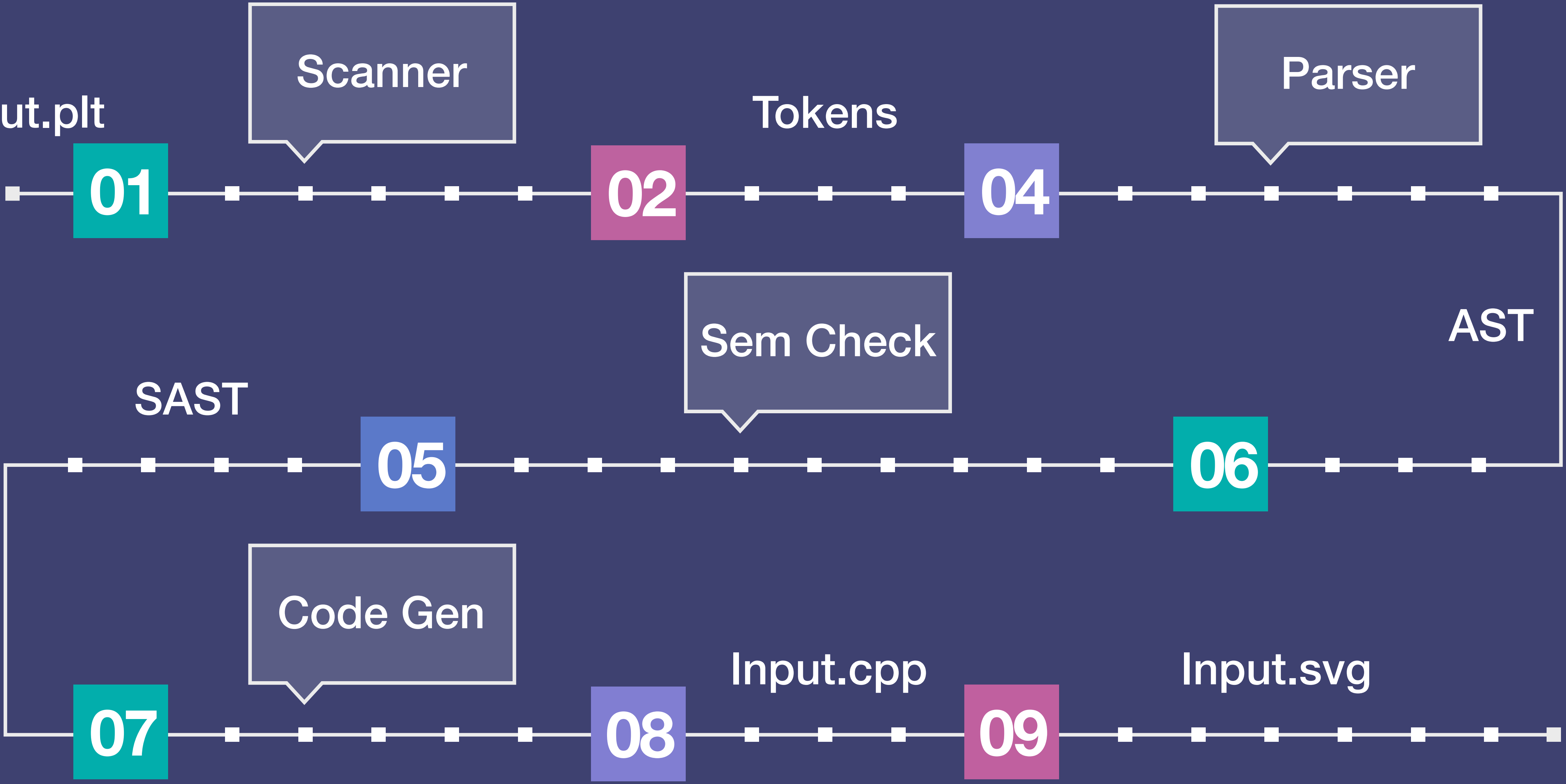
```
fn rect(point a, num h, num w):
    num i
    num x
    num y
    point b
    /* Make a rectanle by
    drawing multiple lines */
    for i=0;i<w;i=i+1:
        x = a[0]
        y = a[1]
        line( (x+i , y), (x+i, y+h))
    end
end
```

## Bar Graph

```
fn barGraph(list num a):

    #Setting the dimensions of the graph
    maxLength = 640
    maxHeight = 480

    #Max ht
    maxDataHt = a[0]
    for i=0;i<a.length();i=i+1:
        if a[i] > maxDataHt:
            maxDataHt = a[i]
        end
    end

    maxDataLn = a.length() #max length

    #padding
    padHz = 10
    padVt = 10

    #bar graph settings 10% of the graph
    gap         = 0.1 * (maxLength - padHz) / maxDataLn
    barWidth    = 0.9 * (maxLength - padHz) / maxDataLn
    scaleFactor = (maxHeight - padVt) / maxDataHt

    #Draw the bars, scaled and with the gap
    x = padHz
    for i=0;i<a.length();i=i+1:
        #Drawing the bar
        rect( (x,maxHeight- a[i]*scaleFactor), a[i]*scaleFactor, barWidth)
        x = x + barWidth + gap
    end
end
```

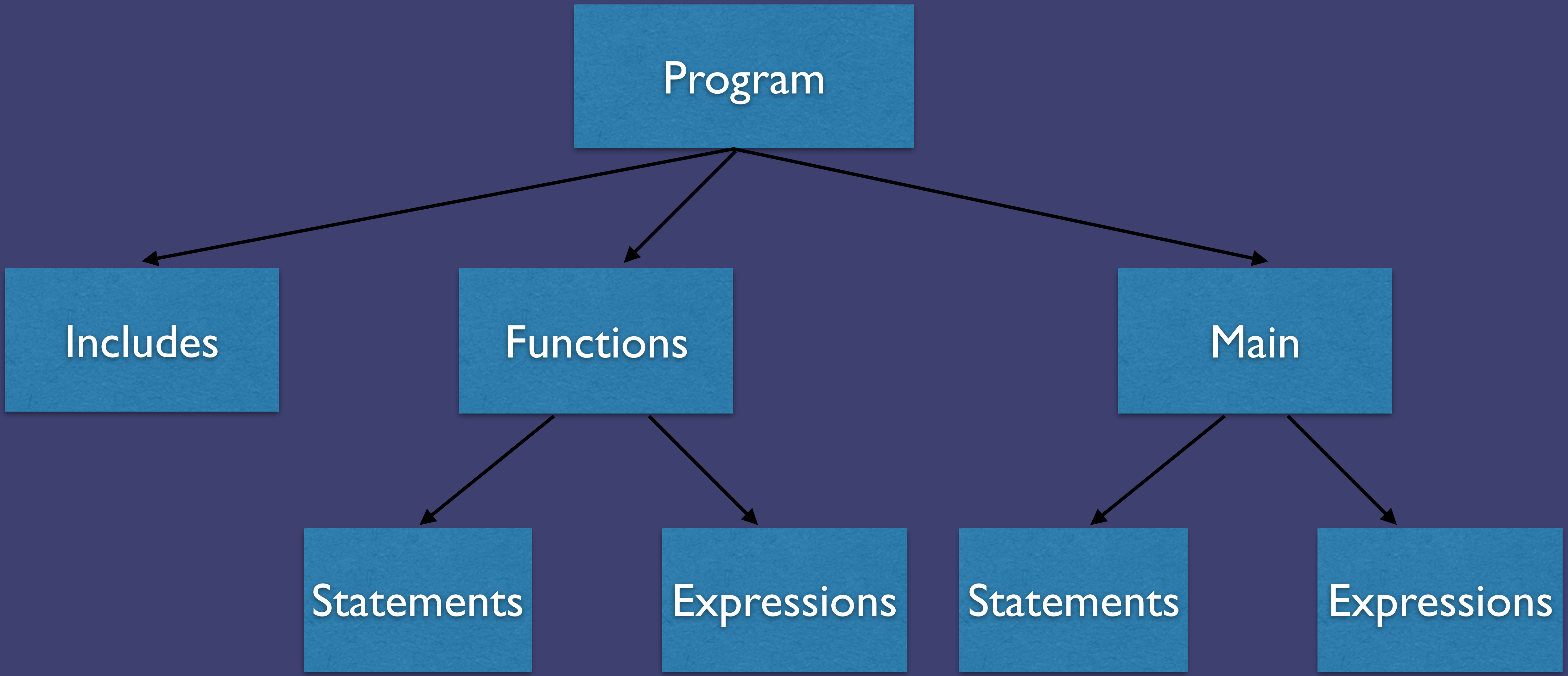
# Pipeline

PlOtter in the making



# Implementation

## AST



# Semantic Checking

The meat of the compiler

**Maintaining a Symbol table**

Undeclared, redeclared functions and identifiers

**Type Checking**

Validation according to language specs

**Scoping and visibility**

Static, block

# Error Reporting

**Scanner:** On Error, report error, stop scanning.

Program:

```
~ point p
p = (10, 10)
```

Output:

```
Fatal error: exception Failure("Illegal character : ~")
```

**Parser:** On Error, report error, continue parsing.

Program:

```
point p
p = (10, 20
```

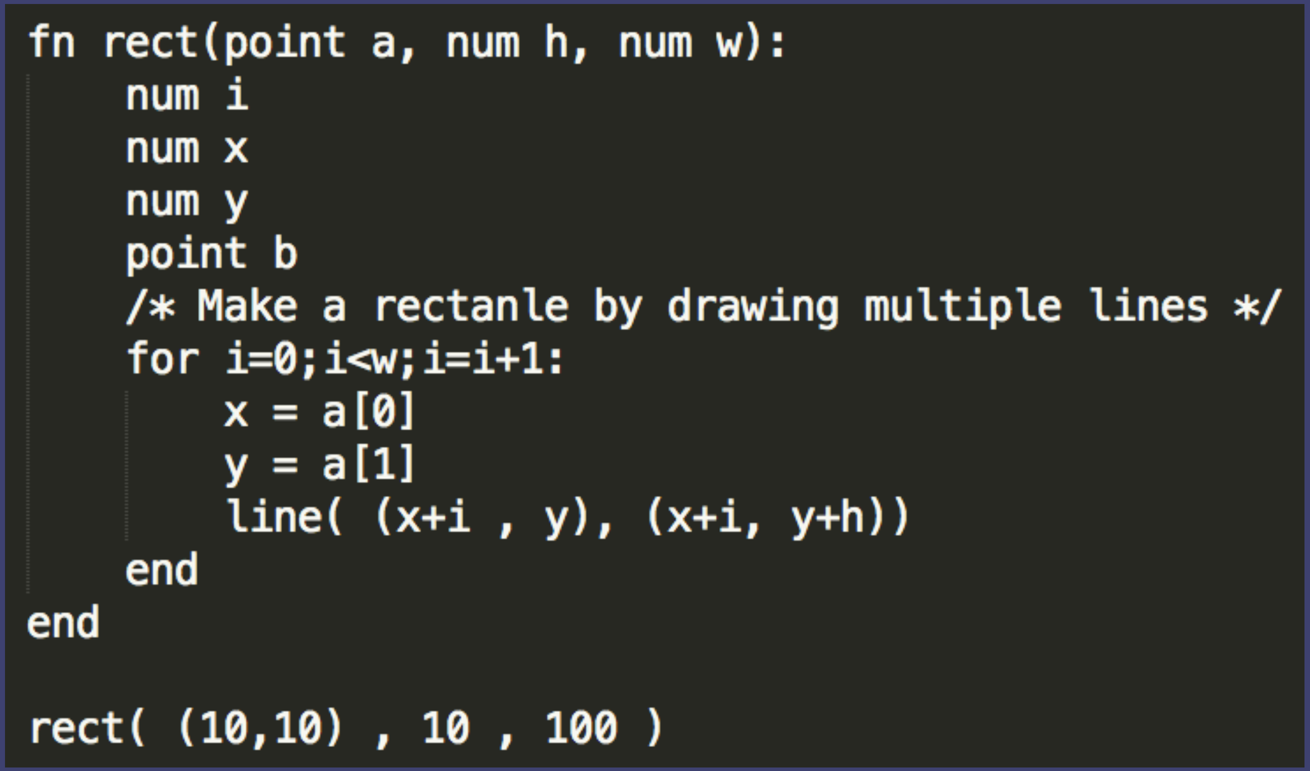
Output:

```
Characters: 12..19: Syntax error: Left ( is unmatched with right ).
```

# Implementation

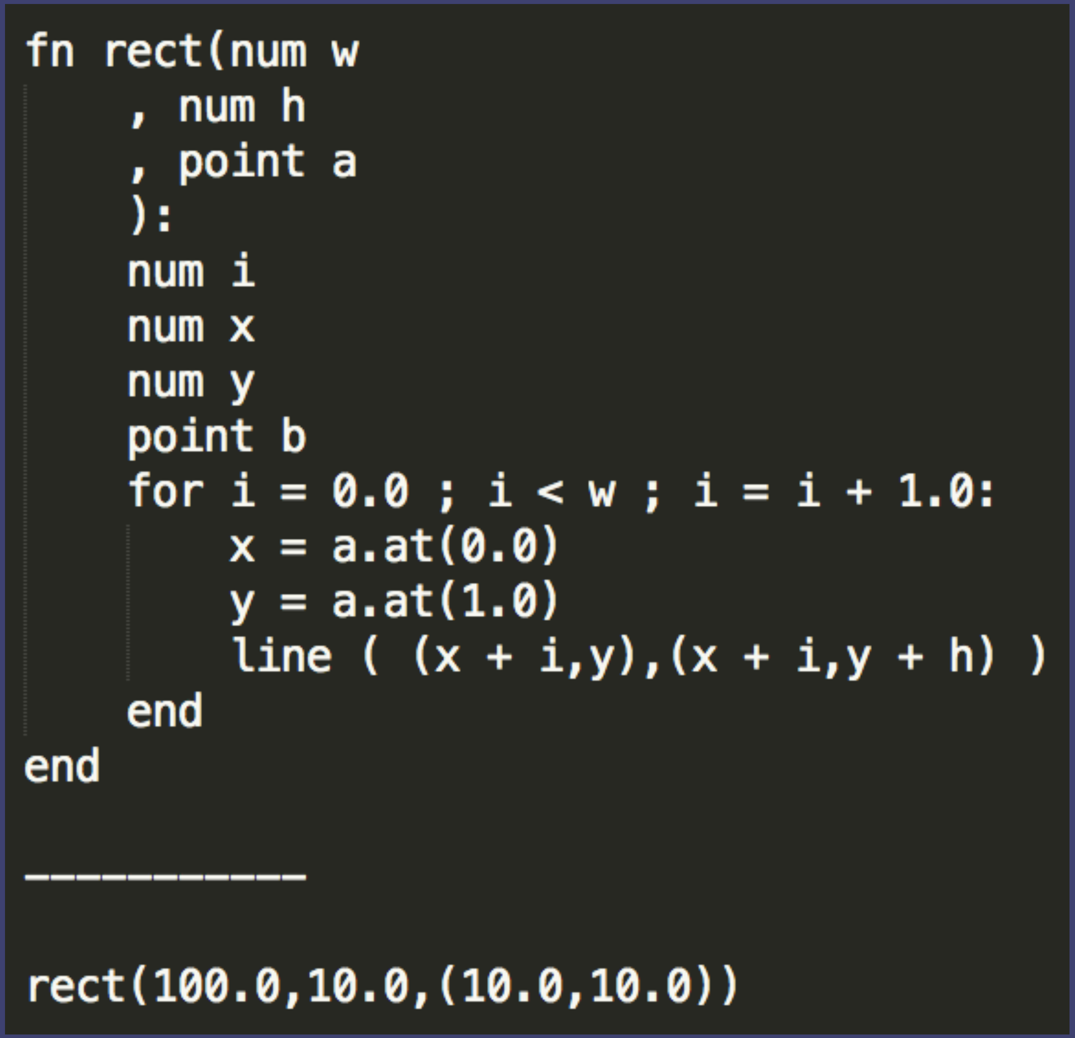
## Compiler Flags

### Program



```
fn rect(point a, num h, num w):
    num i
    num x
    num y
    point b
    /* Make a rectanle by drawing multiple lines */
    for i=0;i<w;i=i+1:
        x = a[0]
        y = a[1]
        line( (x+i , y), (x+i, y+h))
    end
end

rect( (10,10) , 10 , 100 )
```

### Pretty Printing from AST



```
fn rect(num w
    , num h
    , point a
    ):
    num i
    num x
    num y
    point b
    for i = 0.0 ; i < w ; i = i + 1.0:
        x = a.at(0.0)
        y = a.at(1.0)
        line ( (x + i,y),(x + i,y + h) )
    end
end

-----------

rect(100.0,10.0,(10.0,10.0))
```

# Test Driven Development

New test for every feature

Goal part 1: Make test pass

Goal part 2: Fail no other tests

## Pass Tests

Proceeded by the word "Pass"

Tests that we know should pass

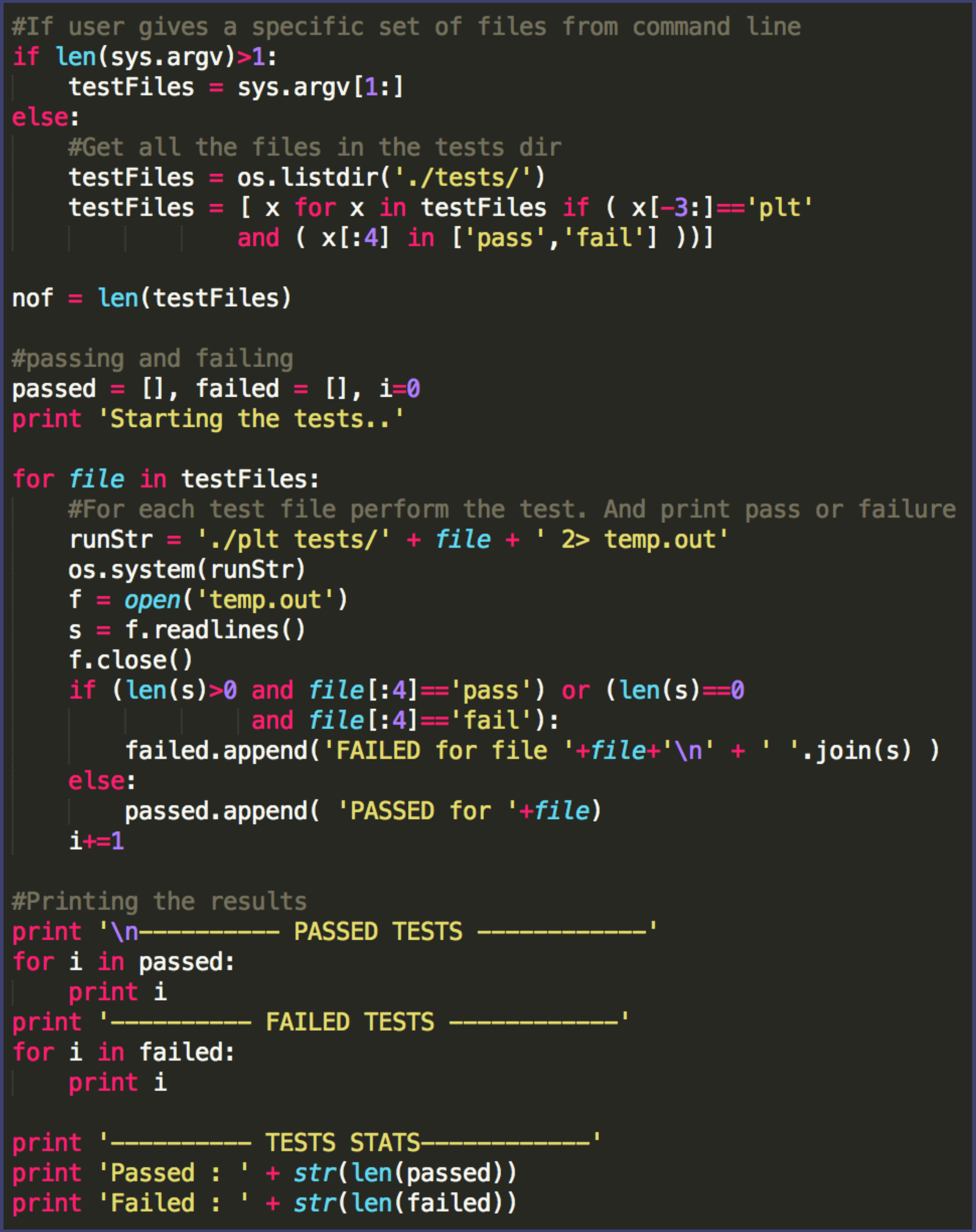
## Fail Tests

Proceeded by the word "Fail"

Tests that we know should fail

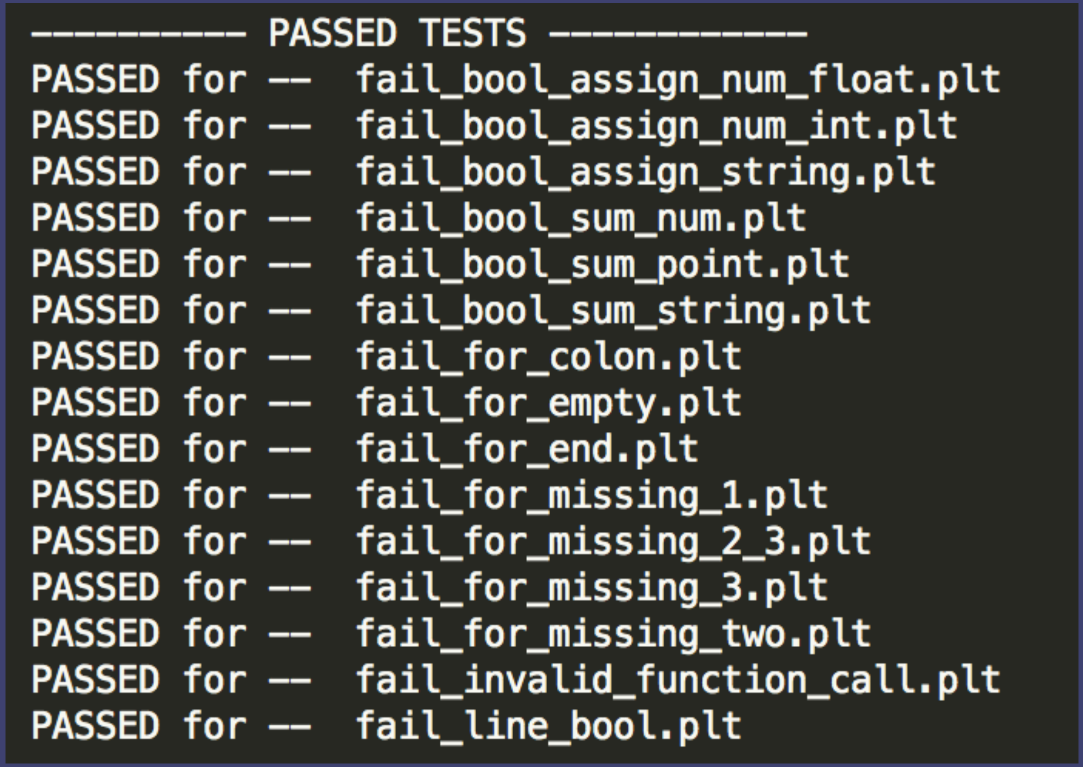
# Test Suite

## Script



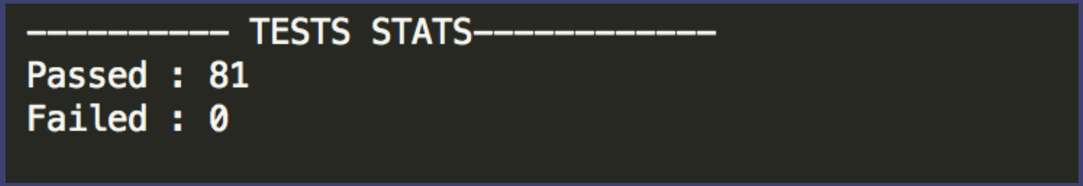
```python
#If user gives a specific set of files from command line
if len(sys.argv)>1:
    testFiles = sys.argv[1:]
else:
    #Get all the files in the tests dir
    testFiles = os.listdir('./tests/')
    testFiles = [ x for x in testFiles if ( x[-3:]=='plt'
                  and ( x[:4] in ['pass','fail'] ))]

nof = len(testFiles)

#passing and failing
passed = [], failed = [], i=0
print 'Starting the tests..'

for file in testFiles:
    #For each test file perform the test. And print pass or failure
    runStr = './plt tests/' + file + ' 2> temp.out'
    os.system(runStr)
    f = open('temp.out')
    s = f.readlines()
    f.close()
    if (len(s)>0 and file[:4]=='pass') or (len(s)==0
                and file[:4]=='fail'):
        failed.append('FAILED for file '+file+'\n' + ' '.join(s) )
    else:
        passed.append( 'PASSED for '+file)
    i+=1

#Printing the results
print '\n---------- PASSED TESTS ------------'
for i in passed:
    print i
print '---------- FAILED TESTS ------------'
for i in failed:
    print i

print '---------- TESTS STATS------------'
print 'Passed : ' + str(len(passed))
print 'Failed : ' + str(len(failed))
```

## Sample Output



```
---------- PASSED TESTS ------------
PASSED for --  fail_bool_assign_num_float.plt
PASSED for --  fail_bool_assign_num_int.plt
PASSED for --  fail_bool_assign_string.plt
PASSED for --  fail_bool_sum_num.plt
PASSED for --  fail_bool_sum_point.plt
PASSED for --  fail_bool_sum_string.plt
PASSED for --  fail_for_colon.plt
PASSED for --  fail_for_empty.plt
PASSED for --  fail_for_end.plt
PASSED for --  fail_for_missing_1.plt
PASSED for --  fail_for_missing_2_3.plt
PASSED for --  fail_for_missing_3.plt
PASSED for --  fail_for_missing_two.plt
PASSED for --  fail_invalid_function_call.plt
PASSED for --  fail_line_bool.plt
```

⋮



```
---------- TESTS STATS------------
Passed : 81
Failed : 0
```

PLOTTER

# Demo

## The future of PlOtter

Customizability

Support for math functions

Import data from external sources

REPL window for on-the-go compiling

More libraries

## Takeaways

Pair Programming saves lives.

OCaml is awesome, give it time.

Use Prof. Edwards' slides.

Choose teammates wisely, you'll be stuck with them for the term.

Courtesy: Prof. Edwards