Pumpkin

Joshua Boggs
Christopher Evans
Gabriela Melchior
Quentin Robbins
Language Overview

- Pumpkin is patchwork functional programming language.
- The Pumpkin programming language is a light-functional scripting language, which allows for coding flexibility and concise syntax.
- Pumpkin supports many syntactic optimizations for function nesting and chaining, such as pipes, partially applied functions, and function composition.
- This language focuses on easily modeling the flow of data through function.
Motivation

Easy functional language, with beautiful syntax

Simple to type, no need for type declaration

Flexible: partial and anonymous functions
Tutorial Introduction

Declare variables with val:

```scala
val y : Bool = True
```

Declare functions with def:

```scala
def add(a: Int,  b: Int): Int => a + b
```

Pipe functions with |>:

```scala
val x = [1,2,3] |> (a: List[Int] => len(a)%2)
if x is 0:
    print("Even")
else:
    print("Odd")
```
Create function compositions with >> or <<:

```scala
val plusTwoTimesThree = (x: Int => x * 3) <<< (x: Int => x + 2)
plusTwoTimesThree(4) # => 18
```

Type inference: for val and def, the types are not necessary.
Function control with if...else loops.
def gcd(a : Int, b : Int) : Int =>
    if(b is 0):
        a
    else:
        gcd(b, a % b)

def relativePrimes(a: Int) =>
    if (a is 1):
        True
    else:
        False

val p = relativePrimes << gcd

if(p(25, 15)):
    print("You have relative primes")
else:
    print("Not relative primes")
Example #2

def reduce(func: (Int, Int => Int), acc: Int, l: List[Int]): Int =>
    if(is_empty(l)):
        acc
    else:
        reduce(func, func(hd(l), acc), tl(l))

def map(f: (Int => Int), l: List[Int]): List[Int] =>
    if(is_empty(l)):
        l
    else:
        f(hd(l))::(map(f, tl(l)))

def even(n: Int): Bool =>
    if(n % 2 is 0):
        True
    else:
        False

val x = [1,2,3,4] |> map((x:Int => x + 5 :Int)) |> reduce((x: Int, y: Int => x + y : Int), 0) |> even
print(x)
Implementation

Main Flow
Scanner -> Parser -> Ast -> Analyzer -> Sast -> Codegen

Helpers

Utils: strings for testing

Pmkn+Processor: executable, can run code through files incrementally in order to test specific modules
Summary

Pumpkin is functional
Pumpkin is flow oriented
Pumpkin has type inference
Powerful and easy to use!
Lessons Learned

It is not easy to know what will be hard to implement until you get simple things out of the way.

Look towards successful precedents for inspiration and guidance.
Thank you for a wonderful semester!