Hogwarts

Language Summary and Example Programs

Version 9.¾

Roxanne Lai

This is the documentation for a new magical programming language called Hogwarts, for a language project in Theory of Programming Languages at Marist College.

1.Introduction

Hogwarts is a modern, object-oriented, and strongly-typed (but with type-inference) programming language, best used in a procedural environment. It is based on Python, Scala, and the wizarding world created by J.K. Rowling, but differing in the following ways:

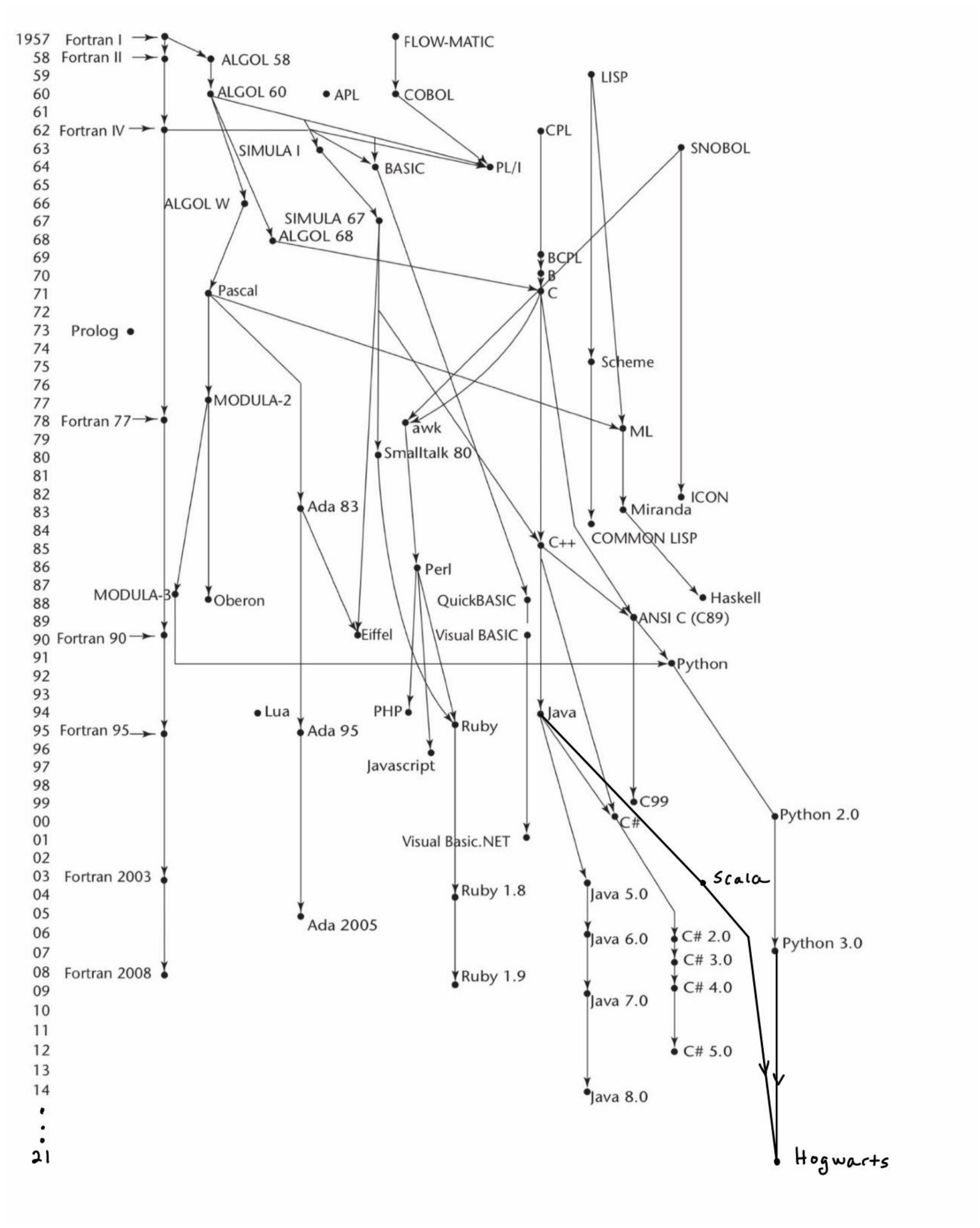
1. Hogwarts is strongly typed, but it also provides type inference to make dealing with types easier, and reduce redundant code.
2. The curly-brackets from Scala have been removed and instead Hogwarts uses the block structure of Python, meaning indentation determines when each function,class, or code block begins and ends.
3. In Hogwarts, print is `incant`, and provides Python's extremely useful f-string formatting.
4. In Hogwarts there are no Null values, instead there is `Detention`.
5. In Hogwarts return values are returned via the "owl" keyword, since in Hogwarts messages are sent via owls. For example if a function wants to return a null value, you would `owl Detention`.
6. While Hogwarts is type-inferred, when we need to declare type for a parameter, we would say `x: Int`. In general, variable types don't need to be declared, because they can be inferred from the right hand side of the assignment.
7. When it comes to constructors, instead of Python's `__init__` (which is not very nice to look at), Hogwarts's constructor is modeled more after Scala or Java's constructor format.
8. Hogwarts supports unary the operators `++`, and `--`.

```
          /\__/\
_____(  ‘ . ‘  )_____
Vvvv   ^^   vvvV
```

(Bat about town)

1.1 Genealogy

The origins of Hogwarts are Scala and Python, which is why Hogwarts is situated below them. The way I would describe Hogwarts is that it is a more elegant blend of Python with Scala's type-inferencing. While Muggles had languages like Scala and Python, the wizarding world wanted a language of their own, which is why Hogwarts was created.



1.2 Hello world

Morally obligated version of a “Hello World” program in Hogwarts.

```
magic helloWorld(mana):
   spell helloWorld(): --<{ this is the constructor
     incant(“Hello World!”) --<{ this is where we do stuff
```

1.3 Program structure

The key organizational concepts in Hogwarts are as follows:

1. Every magic (class) is public, so we don’t need to declare private vs public
2. Every major (non-nested) class needs to be defined in a file named <class>.hgw
3. Hogwarts uses indentation via spaces or tabs like Python, instead of {} or begin/end syntax.
4. While Hogwarts is strongly typed, it also has type-inference
5. Hogwarts supports single inheritance like Python, but in Hogwarts classes are called “magic”, and the root of the object hierarchy is a class called “mana”.
6. In Hogwarts, like Scala, a class can provide a main function which can be executed by the interpreter. That is declared like this:
   ```
   spell main(String[] args) -> Int:
           incant(“This is the main function!”)
   ```
7. The concept of a “package” / “module” is represented as a “tome”, since spells are written in tomes.

```
tome WizardWorld

magic Student(mana):
    --<{ this is the constructor, with a default value
    spell Student(fname: String, lname: String, gpa = 3.0:
    Float):

       thine.fname = fname
       thine.lname = lname
       thine.gpa = gpa

    spell displayGpa():
       incant(f’gpa is: {thine.gpa}’)

    spell displayStudent():
       incant(f’First name: {thine.fname} Last name:
              {thine.lname}’)

    spell Boolean isGoodStudent():
```

```
            if thy thine.gpa > 3.0:
                owl True
            else:
                owl False

*<|
This would create first mana(object) of the Student
magic(class)
|>*
newStudent = Student("Hermione", "Granger", 3.6)
newStudent.displayStudent()
*<|
This will print First name: Hermione Last name: Granger
|>*
```

This example declares a `magic` (class) named `Student` in a tome(package/module) called WizardWorld. The fully qualified name of this class is `WizardWorld.Student`. The `Student magic`(class) has three attributes: `fname`, `lname`, and `gpa`. It has a constructor which takes three arguments whose types are specified, which in this case are `String` and `Int`, as well as a float value (in the case of `gpa`). Assigning to the "thine" (Python's "self") makes those values attributes of the object, with the types inferred from the values declared in the constructor. The `Student magic` (class) also has several methods: `displayGpa`, `displayStudent`, and `isGoodStudent`. The `displayGpa` method will `incant` (print) the `gpa` of a student, and the `displayStudent` method is used to `incant`(print) the first and last name of a student using the `incant` (aka print) command. The `isGoodStudent` method will `owl` (return) a boolean value based on the current gpa of a given student.

1.4 Types and variables

Hogwarts supports the same types as Python, using the same names. As in Python, there is only one kind of variable in Hogwarts: reference variables. In Hogwarts variables are really just names for objects, so when a function is called instead of passing the actual object on the stack, Hogwarts just creates new sets of labels/names for the same objects. See Section 3 for details.

1.5 Statements Differing from Python and Scala

Statement	Example

Expression statement	spell countPotions(numPotions: Int) -> Azkaban:
	    numPotions = 12
	    incant("original amount of potions")
	    numPotions ++
	    incant("The final amount = ", numPotions)

If / else statement	spell studentPrize(numPoints: Int, status: String):
	     if thy numPoints > 10:
	        status = "good"
	     else if thy numPoints == 10:
	         status = "okay"
	     else:
	        status = "bad"
	     owl status

While statement	spell iterationEx():
	    i: int
	    i = 0
	    while thy i < 10:
	        incant(i)
	        i ++

Constructor	spell myMagic(x: Int, y: Int):
	    thine.x = x
	    thine.y = y

For statement	spell loopingEx(foods: Ingredients):
	        for thy x within foods:
	            incant(x)
	        if thy x == "newt"
	            depart

2. Lexical structure

2.1 Programs

A Hogwarts ***program*** consists of one or more ***source files***. A source file is an ordered sequence of Unicode characters. Like Python, Hogwarts programs are parsed into bytecode (enforcing various language rules and constraints, including strong typing) which are executed by the Hogwarts virtual machine.

So, execution of a Hogwarts program works like this:

1. The pre-processor searches for all imports in the source file(s) and expands them.
2. The parser converts the expanded code and performs lexical and syntactic analysis, enforcing language constraints and rules like type inference.
3. The interpreter executes the bytecode in it's virtual machine.

2.2 Grammars

This specification presents the syntax of the Hogwarts programming language where it differs from Python and Scala.

2.2.1 Lexical grammar where differing from Python and Scala

The lexical grammar of Hogwarts is similar to the lexical grammar of Python. However, unlike Python, in Hogwarts identifiers cannot begin with an underscore, and digits are allowed if the identifier starts with a character. Also, unlike Python, Hogwarts uses ++ for incrementing instead of `+= 1`. Here are the BNF grammar productions...

```
< unary operator >       →      ++ | -- | + | - | !
< identifier >           →      <character> <character list>
                         →      <character> < digit>
                         →      <character>
<character list>         →      <character> <character list>
                         →      <character>
< keyword>               →      < language defined>
                         →      < variable defined>
< exponent operator>     →      ^
<single line comment>    →      --<{
< delimited comment>     →      *<| . . .  |>*
```

2.2.2 Syntactic ("parse") grammar where differing from Python and Scala

The syntactic grammar of Hogwarts is similar to a blend of Python and Scala, with some differences, which are outlined below. For example, unlike Scala, Hogwarts declares parameters with <identifier>: <object type> instead of <object type> <identifier>. Here are the BNF grammar productions...

< tome declaration>	→	tome < identifier>
< magic declaration>	→	magic <identifier>
<function declaration>	→	spell <identifier> <parameter list>
	→	spell <identifier>
<parameter list>	→	<parameter> <parameter list>
	→	<parameter>
< parameter>	→	<identifier> **:** <object type>

2.3 Lexical analysis

This specification presents the syntax for comments in the Hogwarts programming language.

2.3.1 Comments

In Hogwarts there are two forms of comments that are supported: single-line comments and delimited/multi-line comments. ***Single-line comments*** start with the characters `--<{` and extend to the end of the source line. Depending on the font, the comments may or may not look like a broomstick, but it was intended to represent a broomstick that a wizard might use for playing quidditch! ***Delimited/multi-line comments*** start with the characters `*<|` and end with the characters `|>*`. Again, like with the single-line comments, the delimited/multi-line comments may or may not look like a wizard's hat, but it was meant to look like the sorting hat used to determine the house of each wizard. In Hogwarts, delimited comments may span multiple lines. Also, in Hogwarts comments do not nest (unlike Buckbeak the hippogriff).

```
     _/\_
     //  }
      / \
  >---/  /"---
      LL
```

(The witching hour)

2.4 Tokens

There are several kinds of tokens: identifiers, keywords, literals, operators, and punctuators. Also, white spaces are significant because they are used to infer block structure, and are therefore considered tokens. However, comments are not tokens, though they act as separators for tokens where needed.

Tokens:

- identifier
- keyword
- integer-literal
- real-literal
- character -literal
- string-literal
- operator-or-punctuator
- white space

2.4.1 Keywords differing from Python or Scala

A **keyword** is an identifier-like sequence of characters that is reserved, and cannot be used as an identifier.

New Keywords	Removed/Replaced Keywords
❖ magic (class)	❖ class
❖ incant (print)	❖ print
❖ owl (return)	❖ return
❖ Detention (null)	❖ null
❖ spell (def)	❖ def
❖ if thy (if)	❖ if
❖ while thy (while)	❖ while
❖ Azkaban (void/NoReturn)	❖ void/NoReturn
❖ mana (object)	❖ object
❖ for thy (for)	❖ for
❖ depart(break)	❖ break
❖ within (in)	❖ in
❖ Ingredient(list /array)	❖ this.
❖ codex (dict/dictionary)	❖ list /Array
❖ thine.(this.)	❖ dict/dictionary

3. Type System

Hogwarts uses a **strong static** type system. Strong typing means that type errors are caught and expressed to the programmer during compilation. Static typing means early binding compile-time type checking. However, Hogwarts makes things easier for the programmer by supporting a Scala-like type inference system. Hogwarts can infer the types of function-scoped variables based on the explicit types of the function arguments, and can thus infer the return type. The interpreter will complain when a variable's type can not be inferred, and it's type has not been explicitly declared. This is different from Python's dynamic typing, which will simply force a variable's type to the type on the right-hand-side (RHS) of an expression at runtime.

3.1 Type Rules

The type rules for Hogwarts are as follows:

$$\frac{\begin{array}{l} S \vdash e_1 : T \\ S \vdash e_2 : T \\ T \text{ is a primitive type} \end{array}}{S \vdash e_1 = e_2 : T}
\qquad
\frac{\begin{array}{l} S \vdash e_1 : \text{String} \\ S \vdash e_2 : \text{string} \end{array}}{S \vdash e_1 \,.\, e_2 : \text{string}}
\qquad
\frac{\begin{array}{l} S \vdash e_1 : \text{integer} \\ S \vdash e_2 : \text{integer} \end{array}}{S \vdash e_1 + e_2 : \text{integer}}$$

$$\frac{\begin{array}{l} S \vdash e_1 : T \\ S \vdash e_2 : T \\ T \text{ is a primitive type} \end{array}}{S \vdash e_1 = e_2 : T}
\qquad
\frac{\begin{array}{l} S \vdash e_1 : T \\ S \vdash e_2 : T \\ T \text{ is a primitive type} \end{array}}{S \vdash e_1 \,!\, e_2 : \text{boolean}}
\qquad
\frac{\begin{array}{l} S \vdash e_1 : T \\ S \vdash e_2 : T \\ T \text{ is a primitive type} \end{array}}{S \vdash e_1 < e_2 : \text{boolean}}$$

$$\frac{\begin{array}{l} S \vdash e_1 : T \\ S \vdash e_2 : T \\ T \text{ is a primitive type} \end{array}}{S \vdash e_1 > e_2 : \text{boolean}}
\qquad
\frac{\begin{array}{l} S \vdash e_1 : T \\ S \vdash e_2 : T \\ T \text{ is a primitive type} \end{array}}{S \vdash e_1 == e_2 : \text{boolean}}
\qquad
\frac{}{\vdash \text{false} : \text{boolean}}
\qquad
\frac{}{\vdash \text{true} : \text{boolean}}$$

$$\frac{i \text{ is an integer literal or Constant}}{\vdash i : \text{integer}}
\qquad
\frac{s \text{ is a String literal or Constant}}{\vdash s : \text{string}}$$

3.2 Reference types (differing from Python and Scala)

As in Python, all values in Hogwarts are reference values. For example, when this code is run, it will print out `“magic”` and `“zip zap”` because the values are references to the originally constructed (mutable) list:

```
items  = [“feather”, “magic”, “zip zap”]
steal = items
steal.remove(“feather”)
incant(items)
```

Ingredient - a list of things

<u>Ex.</u>

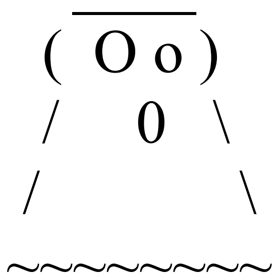
```
items = [“cat-eye”, “newt”, “witch-hazel”]
spell potionItems(items: Ingredient):
                    for thy x within items:
                           incant(x)
                    if thy x == “cat-eye”
                           depart
```

Detention - a null type representing no value

<u>Ex.</u> `owl Detention`

codex - a collection of unordered values accessed by key instead of by index (aka a dictionary)

<u>Ex:</u> `myCodex = { 1: “crucio”, 2: “leviosa”, 3: “avada kedavra”}`



(A haunting sight)

4. Example Programs

This section includes six example programs to illustrate my new language Hogwarts and demonstrates its use; especially what's new and improved over current languages as well as Python and Scala, on which I based my language.

4.1 Caesar Cipher encrypt

```
--<{ A Hogwarts program to illustrate Caesar Cipher Encrypt
spell encrypt(originalText: String, shiftVal: Int):
    result = ""

    --<{ Iterate through originalText
    for thy i within range(len(originalText)):
        char = originalText[i]

        --<{ Handle uppercase characters
        if thy (char.isupper()):
            result += chr((ord(char) + shiftVal - 65) % 26 + 65)

        --<{ Handle lowercase characters
        else:
            result += chr((ord(char) + shiftVal - 97) % 26 + 97)

    owl result

--<{ Do the printing to check encrypt
originalText = "this is a test string from Alan"
shiftVal = 8
incant("originalText  : " , originalText)
incant("Shift Value : " , str(shiftVal))
incant("Encrypted Text: " , encrypt(originalText,shiftVal))
```

4.2 Caesar Cipher decrypt

```
--<{ A Hogwarts program to illustrate Caesar Cipher Decrypt
spell decrypt(originalText: String, shiftVal: Int):
    result = ""
    --<{ Iterate through originalText
    for thy i within range(len(originalText)):
        char = originalText[i]

        --<{ Handle uppercase characters
        if thy (char.isupper()):
            result += chr((ord(char) + shiftVal - 65) % 26 + 65)

        --<{ Handle lowercase characters
        else:
            result += chr((ord(char) + shiftVal - 97) % 26 + 97)

    owl result

--<{ Do the printing to check decrypt
originalText = "this is a test string from Alan"
shiftVal = 8
incant("Encrypted Text: ", decrypt(originalText, shiftVal))
incant("Shift Value: ", str(shiftVal))
incant("Decrypted/ Original Text: ", decrypt(originalText, 26 -
shiftVal))
```

4.3 Factorial

```
--<{ A Hogwarts program to find the factorial of a given number
spell factorialExample(n: Int):
    if thy n < 0:
        owl 0
    else if thy n == 0 or n == 1:
        owl 1
    else:
        factorial = 1
        while thy(n > 1):
            factorial *= n
            n -= 1
        owl factorial

--<{ Driver Code to test factorialExample
num = 9;
incant("Factorial of", num, "is", factorialExample(num))
```

4.4 Quick Sort

```
--<{ A Hogwarts program for implementation of Quicksort Sort

spell partition(array: Ingredient, low: Int, high: Int):
    i = (low - 1) --<{ This is the index of the smaller element
    pivot = array[high]

    for thy j within range(low, high):
        *<|
          If current element is smaller-than or equal-to
          the pivot…
        |>*
        if thy array[j] <= pivot:
        --<{ Increment the index of the smaller element
            i = i + 1
            array[i], array[j] = array[j], array[i]

    array[i + 1], array[high] = array[high], array[i + 1]
    owl (i + 1)

*<|
The function that implements Quick sort, where array is the
Ingredient(aka Array) we wish to sort, low is the starting
index, and high is the ending index
|>*
spell quickSort(array: Ingredient, low: Int, high: Int):
    if thy len(array) == 1:
        owl array
    if thy low < high:

        *<|
          partIndex is the partitioning index, so array[p] is
          now at the right place
        |>*
        partIndex = partition(array, low, high)

        *<|
          Separately sort elements before partition and
          after partition
        |>*
        quickSort(array, low, partIndex - 1)
        quickSort(array, partIndex + 1, high)

--<{ Driver Code to test quickSort
array = [10, 7, 8, 9, 1, 5]
```

```
n = len(array)
quickSort(array, 0, n - 1)
incant("Sorted array is:")
for thy i within range(n):
    incant("%d" % array[i]),
```

4.5 Stack

```
*<|
 A Hogwarts program to illustrate stack implementation using
 a list
|>*

stack = []

--<{ append() is the function to push an element in the stack
stack.append("Harry Potter")
stack.append("Draco Malfoy")
stack.append("Hermione Granger")

incant("Initial stack")
incant(stack)

--<{ pop() is the function to remove an element from the stack
in LIFO (last in first out) order

incant("\nElements removed from stack:")
incant(stack.pop())
incant(stack.pop())
incant(stack.pop())

incant('\nThe stack after elements were removed:')
incant(stack)
```

4.6 Text adventure game

```
*<|
A Hogwarts Text adventure game, to learn about the magic world!
|>*

true = ["T", "t", "True"]
false = ["F", "f", "False"]
*<|
This is where we store the user’s score (number of correct
answers)
|>*
score = 0

--<{ This is where we store the user’s name
userName = input ("What's your name?")

incant("\nHello, ", userName, ", we are going to see how much you
know about the wizarding world! The answers to the questions are
only True or False.”)

incant("\nHarry Potter is the boy who lived")
answer = input(": ")
*<|
If the user guesses correctly they get one point, but if they
guess incorrectly they do not get a point
|>*
if thy answer within true:
  score += 1

incant("\nThe golden snitch is a person who tells on someone")
answer = input(": ")
if thy answer within false:
  score += 1

incant("\nMagic is something that can only be used for good")
answer = input(": ")
if thy answer within false:
  score += 1

incant("\nMagic does not only exist in the UK")
answer = input(": ")
if thy answer within true:
  score += 1

incant("\nMagic does not take a long time to master")
```

```
answer = input(": ")
if thy answer within false:
  score += 1

--<{ After the user has answered all the questions we print...
incant("\nYou have completed the game, ", userName, "! You
scored", score, "out of 5!")
```