Winnie Delinois

Professor Robila

CSIT 313_02

May 9th 2023

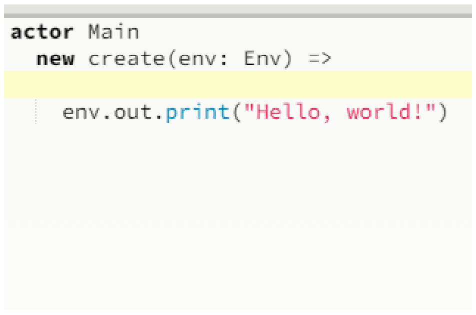
Pony Project Report

Pony is an "open source, object-oriented, actor model, capabilities-secure, high performance" programming language. Pony was created by Sylvan Clebsch, a computer scientist, who started the foundation of the language by creating a C-based actor model in 2011. Clebsch was motivated to create the language due to his frustration with existing programming languages. In existing languages such as C and C++, programmers would often run into memory errors, as well as persistent issues with data races. The programmers would "send a pointer from one actor to another, convinced that it was safe, and it would turn out it wasn't" resulting in deadlock conditions. On the other hand, in situations where bugs were not encountered, the performance aspect of the program would be subpar. Generally speaking, there was a lack of consistency with current programs. Because of this, Clebsch developed Pony with the help of contractors while residing in London. Currently, Pony is maintained by volunteers and contributors on their official GitHub and is led by Clebsch himself. This modern language was designed to address the challenges faced when it comes to concurrent and distributed programming. A brief description of Pony is that it's a type-safe, memory-safe, and data-race-free, statically typed language that makes an emphasis on the program's performance, as well as safety and simplicity. The language is also deadlock-free, an AOT (ahead of time) language, and does not require an interpreter or a virtual machine. Along with this, Pony's native code is compatible with the programming language C since it can call its languages using the foreign function interface. The language was designed to be easy to use, as it shares similarities to the languages Python and Ruby. With that being said, in this report, I will be discussing an overview of Pony. This will include how it can be installed, key aspects of the language to know when first starting, as well as my overall opinion on it.

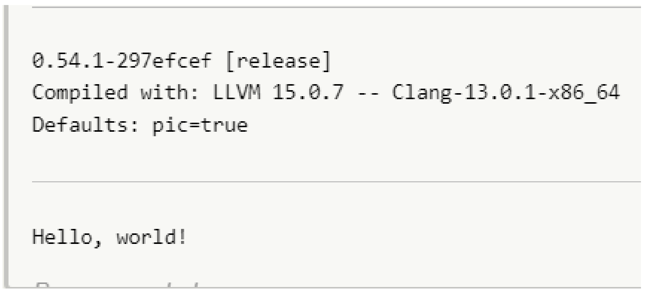
To begin, I wanted to first describe the process of downloading Pony. The official Pony website includes a link to a GitHub blog that sets out the various steps for installing Pony to your system. Since my preferred device is a Windows computer, I will be recounting those specific steps. To use Pony on your computer, you need to first have Visual Studios from 2022 or 2019, or you would need to have Microsoft C++ Build tools. Both links for either program are included in the blog for easy installation. After the program that was chosen was installed onto your computer, you would need to install "Desktop Development with C++ workload, along with the latest Windows SDK (10.x.x.x) for Desktop individual component". After the prerequisites have been added to your device, you would need to get a prebuilt release or nightly build from **ponyup**. The link titled **ponyup** on the installation GitHub page leads to another set of instructions that aids you in installing Pony dependencies from either the compiler or Windows Powershell. The GitHub blog also informs that the latest version of **ponyc** can be installed on Windows by typing "ponyup update ponyc release". The blog does not state where the phrase "ponyup update ponyc release" should be written. Although I made the instructions for installation as concise as I could, I had trouble following the instructions from the blog myself. The blog does not have the option to follow along from a video, so I found implementing the steps for the language to be installed correctly difficult. Because of this, I ended up using the online Pony compiler that can be accessed through your browser. For the purpose of this report, I did not feel as though this decision had a significant difference in the quality of my report in both learning the language and teaching it. However, if you are interested in Pony and planning on using it for more difficult programs, I would suggest using the language from a C++ toolset like Visual Studios or Microsoft C++ Build tools.

Moving forward, the next section that I will be discussing is the language itself. In this portion of the report, I wanted to provide a summary of the different aspects of the language. I found this necessary so that audiences could feel a sense of familiarity with Pony when starting to learn it for the first time. The main attributes that I will be discussing are the “Hello World” expression, variables, and type expressions. These explanations will be accompanied by photos of sample code either written by myself or taken from the provided GitHub links where this information is available. To begin, I wanted to first go over the “Hello World” expression that’s provided when the Pony online compiler is opened. I thought this expression would be important to go over first since it’s traditionally used to introduce new programmers to a language. It is also necessary since it helps determine whether the language is installed correctly, and ready to use.

Sample Code (**GitHub**):



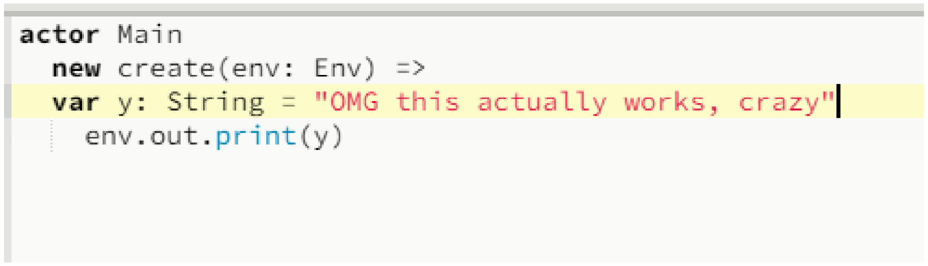
```
actor Main
  new create(env: Env) =>

    env.out.print("Hello, world!")
```



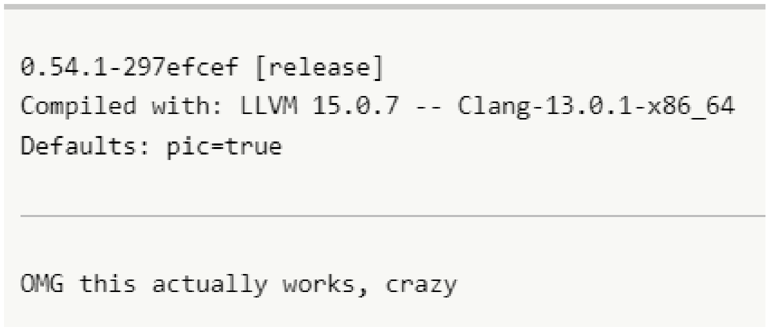
```
0.54.1-297efcef [release]
Compiled with: LLVM 15.0.7 -- Clang-13.0.1-x86_64
Defaults: pic=true


Hello, world!
```

This code snippet is the default code shown on the online Pony compiler. On line one, it has the phrase “**actor Main**”. The **actor Main** in Pony is defined as a type declaration. As also stated in the Pony GitHub, the keyword actor is used to define an actor, much like how you'd describe a class in Python or in Java. The main difference between an actor and a class is that “an actor can have asynchronous methods, called behaviors". In order for a Pony program to run, it needs to have a Main actor, similar to a main method in Java. Moving onto the second line of code, it states "**new create(env: Env)⇒**". This line of code describes a constructor. The keyword new means "it's a function that creates a new instance of the type". In this language, constructors have names, which is why this constructor is labeled create. Afterward, the parameters of the function come next. In this specific example, the constructor has only one parameter called "env that is of the type Env". This is in reference to the environment that the program is in. In Pony, "the type of something always comes after its name and is separated by a colon". In Java or C++, you may write it as Evn env but in Pony it's written the other way around. Finally, the last line of code prints out the string within the print statement from the environment. In Pony, string literals can be in double quotes, or in triple quotes similar to Python. With that, you have an understanding of your first Pony program. Next, I will be discussing what types of variables are in Pony, how they are used, and other essential information.

Sample Code (**My Own**):



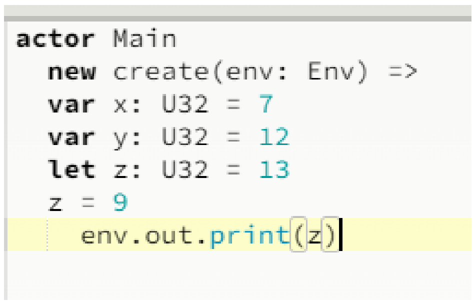
```
actor Main
  new create(env: Env) =>
  var y: String = "OMG this actually works, crazy"
    env.out.print(y)
```



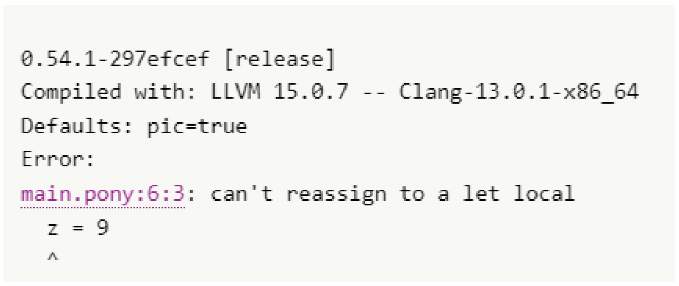
```
0.54.1-297efcef [release]
Compiled with: LLVM 15.0.7 -- Clang-13.0.1-x86_64
Defaults: pic=true

OMG this actually works, crazy
```

A lot like other languages, Pony allows you to store data in variables. Within Pony, there are different types of variables that have different lifetimes and purposes. In order to define a local variable the var keyword is used and afterwards comes the variable's name. Optionally, you can place a colon after the name. In my example, I am assigning my string literal “OMG this actually works, crazy”, to the variable y. Along with this, you don’t need to give a value to a variable after defining it, you could always define it later if you prefer. However, if you attempt to read the value of a variable before defining it, the compiler with give you an error. Local variables can be declared with either **let** or **var**. **Var** can be assigned and reassigned as often as desired, while **let** "means the variable can only be assigned once". Using let instead of var also means that the variable needs to be assigned immediately. If not, it will produce an error in the compiler.

Sample Code (**My Own**):



```
actor Main
  new create(env: Env) =>
  var x: U32 = 7
  var y: U32 = 12
  let z: U32 = 13
  z = 9
    env.out.print(z)
```



```
0.54.1-297efcef [release]
Compiled with: LLVM 15.0.7 -- Clang-13.0.1-x86_64
Defaults: pic=true
Error:
main.pony:6:3: can't reassign to a let local
  z = 9
  ^
```

Sample Code (**GitHub**):

```
let x: U32 = 3 // Ok
let y: U32 // Error, can't declare a let local without assigning to it
y = 6 // Error, can't reassign to a let local
```

An important aspect to note is “that a variable having been declared with let only restricts reassignment, and does not influence the mutability of the object it references”. This is the role of reference capabilities. The use of **let** to describe variables is never mandatory, but if the programmer knows that the reference to a specific variable will remain unchanged, using **let** is a helpful tool to catch errors. It is also useful in terms of comments to show that what is referenced by the variable should not be changed. **Fields** in Pony are variables that live within objects and have the same lifetime as the object they are in instead of being scoped. According to Pony’s GitHub, fields are "set up by the object constructor and disposed of along with the object". If the name of a field starts with “_”(underscore), then it is private. This means that the type the field is in can have code that either reads or writes that specific field. Otherwise, the field is public and can be read from anywhere in the program. Field assignments can be either **var** or **let** and the rules for field assignments differ from variable assignments.

Sample Code (**GitHub**):

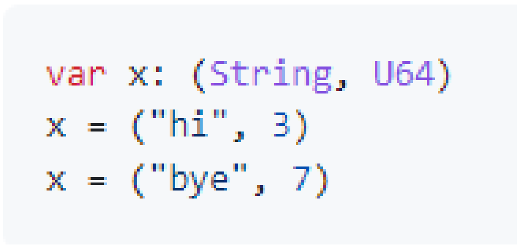
```
class Wombat
  let name: String = "Fantastibat"
  var _hunger_level: U32 = 0
```

```
class Wombat
  let name: String
  var _hunger_level: U32

  new create(hunger: U32) =>
    name = "Fantastibat"
    _hunger_level = hunger
```

Graph 1 **Graph 2**

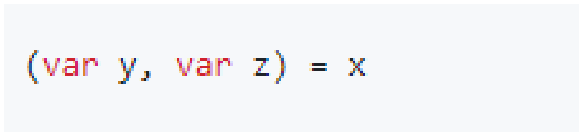
No matter what type of field is used, either "an initial value has to be assigned in their definition or an initial value has to be assigned in the constructor method". In the code example provided by Pony's Github, the starting value of the two fields of the class **Wombat** is assigned at the definition level. The fields could also be assigned in the constructor method as shown in Graph 2. If the assignment is not done at either the definition level or the constructor method, then an error is made in the compiler. This detail is also true for both the var and let fields. The last topic that I wanted to introduce in becoming comfortable with programming in Pony is **type expressions**. Type expressions are more commonly seen in functional programming, and are also called an **algebraic data type**. Pony has three kinds of type expressions. These are **tuples**, **unions**, and **intersections**. A tuple type is a sequence of types. For example, if you wanted to write a String followed by U64 it would look like this.

Sample Code (**GitHub**):

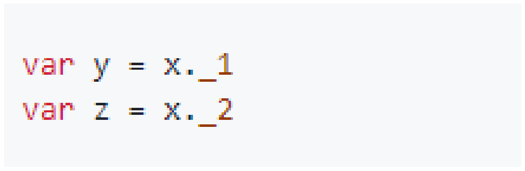


```
var x: (String, U64)
x = ("hi", 3)
x = ("bye", 7)
```

All type expressions are written in parentheses and the elements of the tuple are separated by commas. There is also an option of "destructuring a tuple using assignment or accessing the elements of the tuple directly". It may be helpful to use a tuple instead of a class since tuples express "a collection of values that don't have any associated code or expected behavior". If you only need to return more than one value from a function, tuples would make that process easier in comparison to using a class.

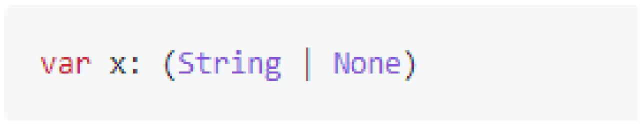


```
(var y, var z) = x
```



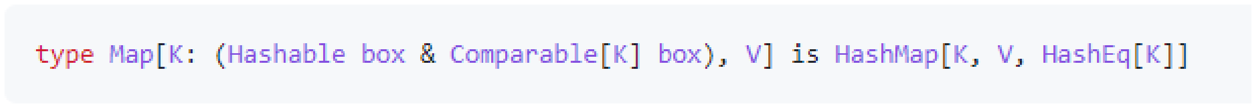
```
var y = x._1
var z = x._2
```

A **union** type is written like a tuple but uses **|** ("or") instead of a comma between its elements. While tuples represent a collection of values, unions "represent a single value that can be any of the specified types".



```
var x: (String | None)
```

An **intersection** uses & ("and") between its elements. Intersection represents the opposite of a union, it is a "single value that is all of the specified types at the same time". This is useful when combining traits and interfaces as shown in the sample code below. The code is basically showing that K is Hashable & Comparable[K] at the same time.



```
type Map[K: (Hashable box & Comparable[K] box), V] is HashMap[K, V, HashEq[K]]
```

Type expressions can be combined into more complex types. Below is another example from the GitHub blog. In this example, there is an array where each element is either a tuple of (**K, V**) or **a _MapEmpty**, or **_MapDeleted**.

```
var _array: Array[((K, V) | _MapEmpty | _MapDeleted)]
```

Since every type expression has parentheses surrounding it, they become easier to read throughout the coding process. However, if complex type expressions are used often, it would be useful to provide a type alias for it

```
type Number is (Signed | Unsigned | Float)

type Signed is (I8 | I16 | I32 | I64 | I128)

type Unsigned is (U8 | U16 | U32 | U64 | U128)

type Float is (F32 | F64)
```

These are all type aliases that are used by the standard library. By having the aliases associated with the types, the code is now easier to read and follow. The use of aliases also makes it look cleaner as well.

In this final section of my report, I will be discussing my opinions on Pony. This will include whether or not I find it efficient, the language's pros and cons, as well as if I would recommend it. To begin, I think in terms of efficiency Pony was fairly easy to follow. The GitHub blogs provided by Pony volunteers were very useful in explaining the basics of the language. The blogs also provided clear instructions to follow while coding as well as providing code snippets that were readable. If I had a line of code that had an error, I did not spend a significant amount of my time trying to figure out the issue. Because of this, I would say that Pony is efficient. I will now discuss some pros and cons that I found while learning the language myself. Starting with the pros, Pony is easy to read, it has simple grammar, and it has similar attributes to object-oriented languages which creates a sense of comfortability. For example, you can comment out lines of code with double slashes in Pony (//) the same way as you would in

Java. The cons involving the language is that there are no videos that you would be able to follow along to. I know this may not be a con for some people, but I consider myself a visual learner. Because of this, not having a video to learn from made the process of learning Pony a bit challenging. Secondly, the GitHub files are not organized that well, which would result in a thorough search to find sub-topics to start learning from. They also are not in chronological order and the files do not suggest the best method to learn the language step by step. Lastly as mentioned before, I had some trouble downloading ponyc onto my device. Although the instructions were quite thorough, I still believe I would have benefited tremendously from following a video tutorial. All in all, I think this language would be easier to understand than Java (Java is not my preferred language), but it depends on the programmer. A slight advantage that Java has over Pony is that it has been around long enough to have multiple video tutorials available whereas Pony does not. If you have the time to learn the language and are fine with not having video references then I would recommend it. If not, you could use Python or Java to save yourself the trouble of having to find or go through multiple resources.

References

Allen, SeanT. "Pony-Tutorial/Type-Expressions.md at Main · Ponylang/Pony-Tutorial." GitHub, 10 Mar. 2021,

https://github.com/ponylang/pony-tutorial/blob/main/docs/types/type-expressions.md.

"Pony-tutorial/variables.md at Main · Ponylang/Pony-tutorial." GitHub, 23 May 2021, github.com/ponylang/pony-tutorial/blob/main/docs/expressions/variables.md.

Ponylang. "Pony-tutorial/how-it-works.md at Main · Ponylang/Pony-tutorial." GitHub, github.com/ponylang/pony-tutorial/blob/main/docs/getting-started/how-it-works.md.

Pony, https://www.ponylang.io/.

Zulip, ponylang.zulipchat.com/. Accessed 9 May 2023.