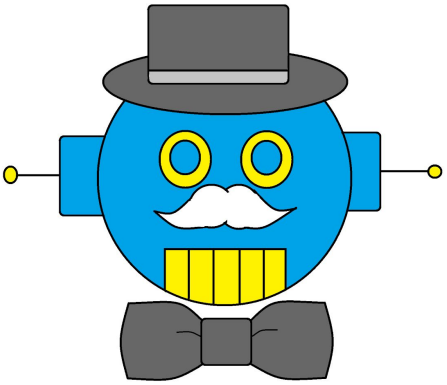
University of Puerto Rico - Mayagüez Campus

Programming Languages - ICOM 4036

Prof. Wilson Rivera - Spring Semester 2017-2018



JARVIS

Final Report

Herbert Pérez Avilés

Mario Flores Rivera

Fernando Guzmán Casas

Kelvin Roche Rivera

Introduction

What's our idea?

Our idea is to create a programming language that simplifies the process of building a local computer bot, allowing someone with almost no programming knowledge to create a bot with a variety of basic functions. This allows an early basic introduction to AI to inexperienced programmers.

What's a bot?

A bot is a piece of software that can execute command, reply to messages or perform routine task, either automatically or with minimal human intervention. For this project, the bot will be an executable program that runs in the terminal/command prompt.

Why bots?

A bot can be useful for simple tasks, or maybe just used for fun. However, the process behind making one can go from extremely simple to very complex in a short time. In order to create a bot, a large amount of possible inputs have to be processed, and in more advanced tasks the bot has to be able to have some sort of memory so that it can better adapt and respond to user requests. For this to be possible the bot would also have to interact with the computer's file system, which for someone who is not familiar with programming can quickly become a nightmare. Even if you have programming knowledge the process can become very tedious due to the many possibilities that would need programming, making *Jarvis* useful to anyone, no matter their programming experience level. This is why our programming language is intended to have a simple syntax anyone can follow, based on "User Input" : "Bot Response".

How will it work?

The user will be able to program their own basic commands to the bot following the syntax that will be explained later on in this document, which lets them not only create their own, but also use a library of more advanced pre-defined commands. The actual code our programming language will be simplifying will be written in Python**,** allowing users that do have advanced programming knowledge to add to the library of predefined commands, expanding the possibilities of what *Jarvis* can let you do.

Language Tutorial

Download Jarvis:

1. Make sure you download and install Python 3 in your computer.
2. Access the project by either:
    a. Selecting “View Github Project” in the website: *k3rmoon.github.io/Jarvis*
    b. Open the project directly at *github.com/K3RMoon/Jarvis*
3. Download or Clone the Project by selecting the “Clone or Download” option.
4. If downloaded, unzip the file, and make sure you know the location of the project folder in your computer.

Make a File to Run:

1. Inside the project directory create a text file with extension (.jvs).
2. In this file, write your code, which is gonna be the bots names with their set of rules.
3. Save your file.

Run in Terminal:

1. Open the terminal.
2. Change the directory to where the project folder of Jarvis is located.
3. Run the command *python Jarvis.py YourFileName*, where *YourFileName* is the name of the file with the code that you created without the extension.
4. The program will run, now enjoy!

Using the Bots:

1. When the program starts, it will give you the names of the available bots you can use.

2. If you type a bot's name and nothing more, the program will show you the possible commands you can use with that bot.
3. To run a command type the bot's name followed by a comma (,) and then type the command you want it to execute. The bots can only use commands that it knows.
4. Example:
    - >>*Jarvis, Hello*
    - Hello, World!
5. To exit the program type "quit".

Language Reference Manual

API:

- Response - Create the rule response for a Bot. This rule gives a message in response to a particular message.
- Learn - The Learn rule allows your bot to learn a number, name, definition, etc.
- Forget - The Forget rule allows your bot to forget something that your bot learns with the Learn rule.
- Action - Create a rule that complete a task from the Actions list.

Actions:

- Sum(num1, num2) – Yields the sum of num1 plus num2.
- Subtract(num1, num2) – Yields the difference between num1 and num2.
- Multiply(num1, num2) – Yields the product of num1 times num2.
- Divide(num1, num2) – Yields the division of num1 and num2.
- RollDice( ) – Give a random number between 1 and 6.
- Modulo(num1, num2) – Yields the modulo of num1 and num2.
- Power(num1, num2) – Calculate the power of num1 to a power num2.
- Root(num) – Yields the square root of num.
- Joke( ) – Give a random joke.
- Random( ) – Give a random number between 1 and 10000.
- *More Actions in development.

Format

To create a bot just type the bot's name followed by braces. The bot's rules will be clustered by these braces. Example:

```
Jarvis {

...

}

Friday {

...

}
```

Rules:

To set up a rule you need to set the actual *Rule* (what you're going to say to the bot that it's going to respond to), and its *Response* (what the bot is gonna respond to the Rule).

For it you need to set the *Rule* as a string, and ,separated by a colon (:), the format of the *Response* you desire. Always finish the declaration of a *Rule* with a semicolon (;).

"String Rule": ResponseName(parameters, ...);

There are 4 possibilities for *Responses* mentioned in the API. The formats of use are as following:

Response

- Type the keyword Response and in parenthesis as parameter, you can put any combination of Strings and/or variables as you desire, separated by (+).
- *Response("String" + variable ...);*

Learn

- Type the keyword Learn and in parenthesis as parameter type the name of the variable you want the bot to save (more on this later).
- *Learn(variable);*

Forget

- Type the keyword Forget and in parenthesis as parameter type the name of the variable you want the bot to forget.
- *Forget(variable);*

Action

- Type the keyword Action followed by a period. After this period you specify one of the predetermined Actions mentioned in the API, with their respective parameters if necessary.
- *Action.ActionName(parameters, ...);*

Hello, World!

```
Jarvis {

"Hello": Response("Hello, World!");

}
```

Parameters:

Each response can accept parameters from the *Rule*. To write a parameter you write a name for the parameter preceded by (~).

For each type of response, the parameter use is different:

Learn

- For the *Learn* response you need to put 2 parameters in the rule. The first one belongs to the variable name that the user is gonna input, and the second one is the value of this variable, which is what the user wants the user to learn.
- *"Learn that my ~param is ~param.": Learn(param);*
- In the above case, the first ~param is the name of the variable that the bot will learn while the second one is the actual value the user wants the bot to learn.

Response

- The *Response* rule may take an argument from the *Rule*. If the parameter is the name of a variable the bot has previously learned, it will look for its value in it's *knowledge* and return a string with a concatenation of the response and the value
- *"What is my ~param":Response(param + "right?");*
- *It can also assume the knowledge of a variable and set the response to its explicit value*
    - *"What is my name":Response(username + "right?");*
- *In this case, username is assumed to be in knowledge and the it will return the same result as the previous case, if its parameter was set to username.*

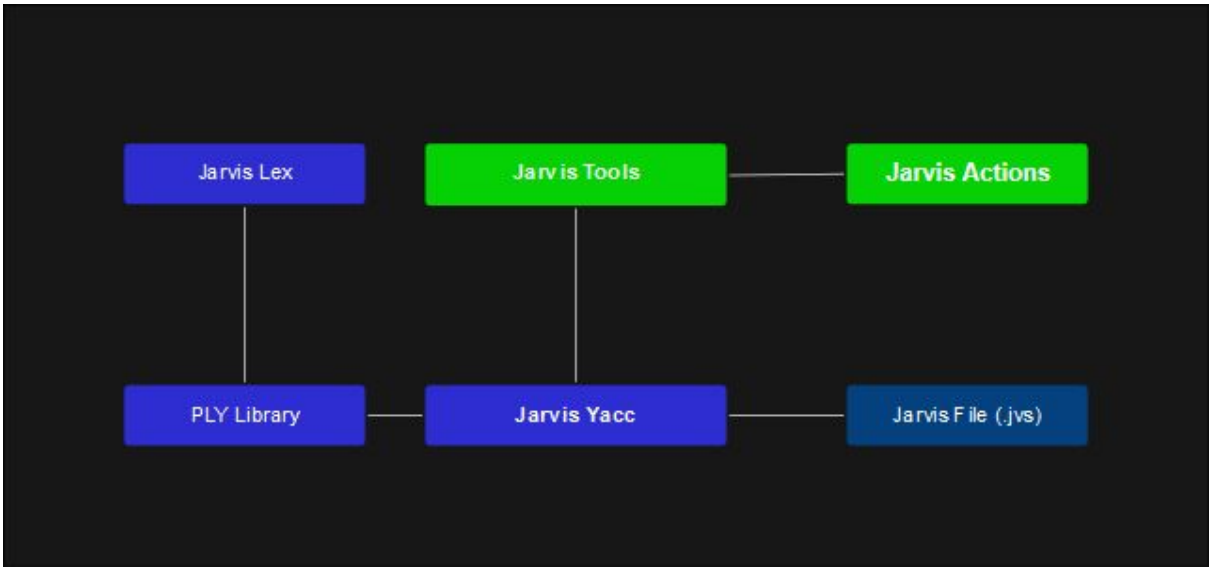
Forget

- As the opposite of the *Learn* rule, *Forget* will take one argument which represents the name of a variable in order to delete it's value
    - "*Forget my ~param":Forget(param);*

Action

- Actions are predefined by the intermediate and can be expanded to add more. For instance the Sum Action takes two parameters which are two numbers. In this case the Action will return the added value of the numbers given
    - *"Sum ~param and ~param":Action.Sum(param, param);*

Language Development



Modules Interaction

Jarvis Rules File

- The set of rules that the Bot will follow.
- It's The actual Jarvis Language, which is translated to python using the Yacc and the Lexer from the PLY Libray
- It's execution results in a system of bots that contains every bot that is correctly created

The Yacc / Lexer

- Takes as input the Jarvis Rules File and translates it to the python code in Jarvis Tools using the PLY library

The Jarvis Tools

- The backend of all the bots that are developed. It has all the essentials functions that a bot can perform implemented in it
- Highly modular allowing modification of the base code to add or remove functions.

- Since the yacc runs some of its commands to create the bot file, the Bot uses the file to know which commands it will have activated and to what it will respond specifically.
- it has a memory file that keeps track of what the bot has learned
- more than one bot can be created at once, and the system responds to whichever is called

Development Environment

- PyCharm & Visual Studio Code
    - IDEs used for the main development of the code.
- GitHub
    - repository of choice for this project...simple, easy to use and reliable.
- PLY
    - parsing tool used in this project. It's all purely Python, and also works great with it.

Test Methodology

Modular Integration

The Language was divided in three main components: the Yacc, Lexer and Jarvis Tools which was previously a standalone project called sample bot. The modules were first tested as separated modules. After ensuring their functionality, their integration was continuously tested and debugged.

Tested On

- The Windows Command Prompt / Mac Terminal / Ubuntu Bash
- PyCharm
- Visual Studio Code

Conclusions and Future Work

Jarvis is a simple, easy to learn programming language compiled using Python and PLY. It's basic syntax makes it possible for anyone to learn it without any type of programming background needed. It is also a practical language, that illustrates the functionality of a bot and how it can be easily implemented.

Thanks to the Jarvis Programming Language modularity, it is easy to expand for more advanced programmers who which to simplify more advanced tasks. For instance, Jarvis can be easily modified to become a file manager assistant, that finds files duplicates according to a given file extension. It can also serve as a package tracker that gives shipment details and saves historic data for time of delivery predictions and so on.

Finally, Jarvis is a practical illustration of all how a high level programming language actually works, and how it can facilitate more complex tasks for new developers without dealing with more complicated concepts such as Object Oriented Programming, Parallel Programming among others.