GSoC 2017 Proposal to Kivy (Python Software Foundation)
Project: KV Compiler: A compiler for the KV language

Sub-organization information

Kivy

Mentors

Matthew Einhorn (matham)

Student Information

Name: Yash Jain

Email: yashjain.lnm@gmail.com

Telephone: +919414419320 / +919660243960

Time Zone: Jaipur, India UTC+5:30

IRC: yaki29@irc.freenode.net

Source Control Username: http://www.github.com/yaki29

Facebook: https://www.facebook.com/profile.php?id=100003284453761

Blogs: https://yaki29.github.io/Blog (Will be available)

University Information

University: The LNM Institute of Information Technology, Jaipur

Major: Computer Science and Engineering

Current Year: 2nd Year

Expected Graduation Completion: By June 2019

Degree: B-Tech

Project Proposal Information

Proposal Title: **KV Compiler: A compiler for the KV language**

Proposal Abstract: This Proposal is based on understanding of kv language in the ideas page provided by Kivy Organization. The goal of the project will be to create a compiler which compiles kv code into python code, with debug/optimization option.

Project Description:

A compiler is a computer program (or a set of programs) that transforms source code written in a programming language (the source language) into another computer language (the target language). It might include debugging and optimization as well. Here in this case the source language would be kv-lang and the target language would be python.

Currently, the bindings are not at all optimized because upon each widget creation all of the rules are re-evaluated and bound. This process can be significantly optimized by pre-compiling the kv code, especially the bindings. Hence we need a better and optimized compiler.

- **How the compiler will be implemented(Technical Overview):**

What our target is to design a consistent, modular and extensible compiler which compiles kv code into python code. For now, what our aim is not to rebind again and again every time a binding is changed or new one is added. We have to take care of the edge cases, like sometimes it is unavoidable, like here, *"x:self.y.z",* when y changes we have to rebind. Currently, optimization comes from batching bindings i.e. we do tons of bindings when we instantiate a widget, we could batch them so that things are triggered less often during the binding and instantiation stage.
We'd have to think about how kivy does things currently in terms of binding and how it could be perhaps improved, specifically for usage in the compiler. What is needed for this project is to have a pretty deep understanding of how kivy lang currently work, how event dispatcher works as well as how kivy properties work, because the compiler would have to make heavy usage of this.
We will have to look at how parsing is done, how parse tree is generated etc. The compiler only does things once we have the kv files parsed into nice structures. We will have to batch bindings to get the results stated above, first hurdle comes the order of bindings, in which order they are executed, in this case it would be in defined manner(until some better solution comes out). Basically, what we do is gather all the widgets in the tree, create bindings and then dispatch any that may have changed. We may apply other local optimization techniques later. Most important thing is we will have to perform hell lot of testing in regular intervals and sort out the bugs as we move ahead in this project.
The key is that the compiler would have to be written such that additional things can be added later. There would be quite a bit of research involved later because we'd have to figure out what is best way to implement various aspect of the project for which we have multiple options.

- **Access to required hardware:**
  I personally own Linux, Windows and Android. If anything else is needed I will buy it within a week.

- **Dividing the work flow:**
  My summer vacations will start from mid-May and will be ending in mid-July, So I have planned my workflow accordingly. I will finish the basic understanding(theory) part as soon as possible, understand the alpha stage compiler and jump to the next part as explained in the timeline. I will be dividing my work in 4 parts.

  - Phase-1: Understand the alpha stage compiler. (Explained in timeline).
  - Phase-2: Add binding features and basic optimization methods. (Explained in timeline)
  - Phase-3: Consistent and Optimize compiler is hoped, with testing the code. (Explained in timeline)
  - Phase-4: Winding up the project(extensive testing and complex optimization). (Explained in timeline)

Timeline:

Up to 23rd May	*For compiler working to understand I'll have to study a lot about kivy-bindings, widget-behaviors, how they work, how they show changes, when they are changed etc.* *I will be reading regularly as suggested by my mentors. I would gain as much knowledge as I can regarding optimization and debugging.* *I will be in touch with my mentors and take suggestions.* *During this time, if my mentor thinks that I am ready to start programming then I will start working on my Phase-1.*
23rd May – 17th June	**<u>Phase-1</u> (Week 1 - 4)** • During this phase I will be working on alpha stage compiler, understand how it works, flow of control and how it is different from the existing one. • Enlist missing parts of the compiler and the features to be added. • Present optimization techniques, keeping consistency as my first motive. • Documentation and examples will be done along with this.
23rd May – 5th June	**(Week 1 & 2)** • Start writing a basic compiler which at least compiles basic examples of kv-language. • Perform some testing and debug the code and keep on adding features as reviewed by the mentors.
6th June – 17th June	**(Week 3 & 4)** • Work on batching the bindings and test how it works for different set of examples. If my mentor suggests, I might start working for Phase-2.
18th June – 7th July	**Phase-2 (Week 5 & 6 & 7)** • Add features to the compiler which analyses the widget tree and rebind the bindings when they are changed or new ones are added. • Documention and examples will be done along with this.
18th June – 26th June	**(Week-5)** • Code the compiler that does batching of bindings and rebind them as they get changed. • Extensive testing will be needed here with resolving the bugs.

Mid-Term Evaluation	**(6th Week)** • Make Preparation for the mid-term evaluation. • Seek feedback and make revisions based on that. • Submit the evaluations before 30th June.
30th June – 7th July	**(Week 7)** • Will take the feedback from the evaluation and make changes (if required). • Continue with the previous task. If done, then move to next step. • If completed before time, then jump to Phase-3
07thJuly – 24th July	**Phase-3 Week (8, 9 & 10)** • During this period, I will be working more on Consistency and Optimization methods and techniques. • Documentation and regular testing will be done along with it.
07th July – 24th July	**Week (8 & 9 & 10)** • Discuss with the mentors about various methods and techniques that can be used to optimize the compiler. • Apply those techniques and perform regular testing and debug the code along with it. • Submit the evaluations before 28th July.
1st August – 23rd August	**Phase-4 Week (11 & 12 & 13)** • During this phase, I will be working on complex features. • Documention and examples will be done along with it.
1st August – 18th August	**Week (11 & 12)** • Complex Optimization techniques will be implemented, as discussed with the mentors. • Extensive testing will be my main focus during this phase.
19th August -23rd August	**(Week 13)** • Continue to Implement features (if left). • Complete any missing documentations. • Complete evaluations and send them before 29th August.

24th August – 29th August	• A Buffer of one week has been kept for any unpredictable delay.
Onwards	• Keep contributing to Kivy and its sister projects and make use of these features.

Link to a patch/code sample, preferably one you have submitted to your sub-org (*):
I have implemented the following:

1. **Facelock using Computer Vision techniques. (Merged)**
2. **BeautifulSoup4 recipe for android. (Waiting for approval)**
3. **Camera example for kivy. (Merged)**
4. **Corrected Basic Documentation mistakes in p4a.**

Other Commitments:

- Have you applied to any other organization? **No.**
- Do you have any other commitments during the main GSoC time period? **No.**
- Do you have exams or classes that overlap with this period? **No.**
  (Even if there is some kind of emergency, in that case I will work extra hard during the weekends.)

Why am I apt. for this project:

Familiarity with Kivy and its coding style. I have an extreme interest and curiosity towards coding and have been doing it for past 2 years with renowned languages Python, Java and C. I have been contributing to Kivy and its sister projects for past 5-6 months(mainly Python-for-android and Kivy-Garden). I am working on a 2-pass Assembler at my college and some python projects in Kivy, some of them could be found on my GitHub profile. I reached semi final round of international coding challenge, **Snackdown 2016, Codechef.** I have also earned T-shirts and accessories from **OSFY(Open Source for You)** for my articles. One of them is going to publish in April's edition.

Proposal Improvements:

- It will definitely consist of an optimization/debug option, one more option can be added further which recursively compiles all the files in the working directory, so that user doesn't have to sit and compile all the files of his kivy-project.
- What if user just wants to compile some of the custom kv files, he/she can just enter the file names using a space between them it will identify the multiple files and compile them one by one. That will make the compiler more handy.