



LPYTHON

Google Summer of Code 2022

Project: Implement modules from the Python Standard Library

Project Mentors: [Ondřej Čertík](#), [Naman Gera](#), [Smit Lunagariya](#)

ABSTRACT

`OBJ`

LPython is a Python compiler in heavy development. It is currently in the pre-alpha stage. The project includes discussing which modules will be needed for LPython (from a scientific computing perspective, in the beginning), creating a priority list, and then implementing each module properly. The aim of this project is to make LPython work for any Python code down the road. Some of the goals of LPython:

- The best possible performance for numerical array-oriented code
- Run on all platforms
- Compile a subset of Python and be Python-compatible
- Explore how to design it so that it can be eventually used with any Python code
- Fast compilation

My prime focus during the project period would be on:

- Implementing priority modules needed for Lpython
- Creating extensive integration tests for respective functions in modules
- Zero bugs - Fix the currently identified bugs
- Make the documentation more user and developer-friendly

CONTENTS

1.	About Me	3
1.1.	<i>Programming Experience</i>	4
1.2.	<i>Contact Info And timezone(S)</i>	5
1.3.	<i>Availability And Commitments</i>	5
2.	Essential Prerequisites	6
2.1.	<i>Compile Lpython On Ubuntu 20.04.2 Lts.....</i>	6
2.2.	<i>Successfully Run Tests</i>	7
2.3.	<i>Run Integration Tests.....</i>	8
3.	The Project	8
4.	Timeline.....	12
5.	Previous Contribution	14
5.1.	<i>Pull requests.....</i>	14
5.2.	<i>Issues.....</i>	14
6.	My Motivation	14
7.	Expectations From Mentor	15
8.	Post GSoC.....	15

1) ABOUT ME

Full name	Madhav Mittal
University	Indian Institute of Technology (BHU), Varanasi
Course Major	Mathematics and Computing
Year	Third
Expected Year of Graduation	2024
Degree	Integrated Dual Degree
GitHub	https://github.com/Madhav2310
LinkedIn	https://www.linkedin.com/in/madhav-mittal-599569192/
Resume	https://drive.google.com/file/d/1XJG2hAtMUBf8nDjelXbTTzW1r8Ekyg5R/view?usp=sharing

Curiosity to learn new things has always driven me. Whether it be logical games in school life or rigorous algorithmic problems in competitive coding contests, every experience has helped me structure my thought process.

Being a technology enthusiast even before I entered college, I always enjoyed the time I spent debugging my code and PC. During my first year, I was amazed at how genuinely vast the world of Computer Science is, and there is so much for me to learn, which I genuinely enjoy. My first interaction with open source was Linux and NPM; it was a whole new realization. I was using a project someone else developed, and it has helped me scale up my projects. The idea of collaborating with so many people working together to build amazing things attracted me to open source.

For about 2.5 years now, I have been expressly coding in C++ and python. And in this time, I've very much fallen in love with python. I was lucky to come across this amalgamation of opportunities to work on a compiler for Python while working on Open Source, and I got fascinated by Lpython. Since then, I've had a great time talking to the mentors and understanding the codebase so as to start contributing.

1.1) PROGRAMMING EXPERIENCE

The areas that capture my interests are Data Structures, Algorithms, Operating Systems, Natural Language Processing, and Information & Security. For most of my programming journey, I have worked primarily with Web-based technologies(my niche) and C/C++ programs, and I have recently been diving into Deep Learning as well.

Relevant Coursework for the Project:

- Operating Systems
- Algorithms
- Data Structures
- Computer System and Architecture
- Object-Oriented Programming
- Client and Server Architecture

I started out in competitive programming, wherein I used to code in C++ and C. After that, I got fascinated with Computer Vision and Machine Learning, for which I had to use Python3. Eventually, I ended up developing an interest in Software Development. Starting off with basics (HTML, CSS, and Javascript), I later moved to frameworks and technologies built around Django and Django REST Framework. I am a quick learner and very keen on

expanding my horizons to newer technologies. If the project requires a skill I need to learn, I will give my 100% to practicing it.

1.2) CONTACT INFO AND TIMEZONE(S)

Primary Email: madhav.mittal.mat19@itbhu.ac.in

Secondary Email: madhavmittal52@gmail.com

Phone No.: +91-9667474952

I am reachable on the following platforms: Slack, Github, Linkedin, Discord, Zulip Chat, email, or any other meeting app with scheduled meetings.

Generally, the time slots which suit me are:

- 15:30 - 18:30 IST (UTC 10:00 - 13:00)
- 22:30 - 02:30 IST (UTC 17:00 - 21:00)
- 10:00 - 14:00 IST (UTC 04:30 - 08:30)

I can start my day a couple of hours early, even around IST 0600 (UTC 0130) in the morning and can end it late until IST 0200 (UTC 2030) if it helps communicate with other developers.

1.3) AVAILABILITY AND COMMITMENTS

I have made no prior commitments to anyone so I will be working towards the project during the summer and will update the mentors regularly about the progress via any medium they prefer. During this time, I will be available for around 35 hours per week. When my college reopens in Mid July, I will be able to denote approximately 30 hrs per week.

2) ESSENTIAL PRE-REQUISITES

1. I am able to compile LPython on my Ubuntu 20.04.2 LTS

```
(lp) madhav@madhav-ThinkPad-E590:~/lpython$ cmake -DCMAKE_BUILD_TYPE=Debug -DW
ITH_LLVM=yes -DWITH_STACKTRACE=yes -DWITH_LFORTRAN_BINARY_MODFILES=no .
-- Found LLVM 11.0.1
-- Using LLVMConfig.cmake in: /home/madhav/mambaforge/envs/lp/lib/cmake/llvm

Configuration results
-----
LPYTHON_VERSION: 0.2.0-218-g6a0384230-dirty
CPACK_PACKAGE_FILE_NAME: lpython-0.2.0-218-g6a0384230-dirty-Linux
C compiler      : /usr/bin/cc
C++ compiler    : /usr/bin/c++
Build type: Debug
C compiler flags      : -g
C++ compiler flags   : -Wall -Wextra -g -ggdb
Installation prefix: /usr/local
WITH_LFORTRAN_ASSERT: yes
LPYTHON_STATIC_BIN: no
WITH_STACKTRACE: yes
WITH_UNWIND: yes
WITH_BFD: yes
WITH_DWARFDUMP: no
WITH_LINKH: yes
WITH_MACHO: no
HAVE_LFORTRAN_DEMANGLE: yes
WITH_LLVM: yes
WITH_XEUS: no
WITH_JSON: no
WITH_FMT: no
WITH_LFORTRAN_BINARY_MODFILES: no
WITH_RUNTIME_LIBRARY: YES
WITH_TARGET_AARCH64: no
WITH_TARGET_X86: yes
-- Configuring done
-- Generating done
-- Build files have been written to: /home/madhav/lpython
```

2. I am able to successfully run tests:

```
(lp) madhav@madhav-ThinkPad-E590:~/lpython$ cmake --build . -j16
Consolidate compiler generated dependencies of target lpython_runtime_static
Consolidate compiler generated dependencies of target lpython_runtime
Consolidate compiler generated dependencies of target doctest
[ 3%] Built target lpython_runtime
[ 6%] Built target lpython_runtime_static
[ 9%] Built target doctest
Consolidate compiler generated dependencies of target asr
[ 12%] Building CXX object src/libasr/CMakeFiles/asr.dir/codegen/asr_to_cpp.cpp.o
[ 12%] Building CXX object src/libasr/CMakeFiles/asr.dir/codegen/asr_to_py.cpp.o
[ 13%] Building CXX object src/libasr/CMakeFiles/asr.dir/codegen/x86_assembler.cpp.o
[ 15%] Building CXX object src/libasr/CMakeFiles/asr.dir/codegen/asr_to_x86.cpp.o
[ 16%] Building CXX object src/libasr/CMakeFiles/asr.dir/pass/do_loops.cpp.o
[ 18%] Building CXX object src/libasr/CMakeFiles/asr.dir/pass/global_stmts_program.cpp.o
[ 20%] Building CXX object src/libasr/CMakeFiles/asr.dir/pass/param_to_const.cpp.o

[ 90%] Building CXX object src/lpython/tests/CMakeFiles/test_lpython.dir/test_parse.cpp.o
[ 90%] Building CXX object src/lpython/tests/CMakeFiles/test_lpython.dir/test_stacktrace2.cpp.o
[ 92%] Building CXX object src/lpython/tests/CMakeFiles/test_lpython.dir/test_serialization.cpp.o
[ 93%] Building CXX object src/lpython/tests/CMakeFiles/test_lpython.dir/test_llvm.cpp.o
[ 95%] Linking CXX executable test_stacktrace
[ 95%] Built target test_stacktrace
[ 96%] Linking CXX executable test_lpython
[ 98%] Linking CXX executable lpython
[100%] Built target test_lpython
[100%] Built target lpython
```

```
(lp) madhav@madhav-ThinkPad-E590:~/lpython$ ./run_tests.py
TEST: doconcurrentloop_01.py
* ast ✓
* asr ✓
* cpp ✓

TEST: complex1.py
* ast ✓
* asr ✓

TEST: complex2.py

TEST: constants1.py
* ast ✓
* asr ✓
```

3. I am able to successfully run Integration tests:

```
(lp) madhav@madhav-ThinkPad-E590:~/lpython$ ./integration_tests/run_tests.py
Compiling LPython tests...
Running: ../src/bin/lpython exit_01.py -o tmp/exit_01
Running: ../src/bin/lpython expr_01.py -o tmp/expr_01
Running: ../src/bin/lpython expr_02.py -o tmp/expr_02
Running: ../src/bin/lpython expr_03.py -o tmp/expr_03
Running: ../src/bin/lpython expr_04.py -o tmp/expr_04
Running: ../src/bin/lpython expr_05.py -o tmp/expr_05
Running: ../src/bin/lpython test_types_01.py -o tmp/test_types_01
Running: ../src/bin/lpython test_str_01.py -o tmp/test_str_01
Running: ../src/bin/lpython test_str_02.py -o tmp/test_str_02
Running: ../src/bin/lpython modules_01.py -o tmp/modules_01
Running: ../src/bin/lpython test_math.py -o tmp/test_math
Running: ../src/bin/lpython test_numpy_01.py -o tmp/test_numpy_01
Running: ../src/bin/lpython test_numpy_02.py -o tmp/test_numpy_02

Running: integration_tests/tmp/test_complex
Running: PYTHONPATH=src/runtime/ltypes python integration_tests/test_complex.py
Running: integration_tests/tmp/test_max_min
Running: PYTHONPATH=src/runtime/ltypes python integration_tests/test_max_min.py
Running CPython tests...
Running: PYTHONPATH=src/runtime/ltypes python integration_tests/test_builtin_bin.py
ALL TESTS PASSED
```

3) THE PROJECT

In this section, I will provide the details and explanation regarding my project during this summer.

The task is to develop modules such that LPython works for any Python code down the road at the fastest compilation time.

The key elements that will be of significant concern during the project include:-

1. Prioritizing Modules and constructing functions for the same
2. Writing good test cases for each created function in the modules
3. Documentation to ease future developers in the project.

In the beginning, it is vital that we recognize the modules to be started with, along with setting the project priority for the same. The higher priority modules are those that would be greatly useful in making other modules too.

High Priority	Mid Priority	Low Priority
Python Built-in	Datetime	Pickle
Math	Time	Argparse
String	Itertools	Glob
Numbers	OS	Json
Statistics	Functools	Tempfile
Decimal	Cmath	Platform
Random	CSV	AST
Sys	Fractions	Logging

My first priority would be to familiarize myself with the declaring and calling of various data types (Example: Integer(i8, i16..i64), Char, list, string, etc.). Some are still buggy, so I would work on identifying and debugging those.

Once that is sorted, I would resume implementing modules, aiming for at least two a month. The process followed is that I take inspiration and reference from the LFortran codebase, the [Python Standard Library Documentation](#), and the mental logic built over the years, and use the templates that we decide in the LPython community.

Math.py, an essential library, is one with several functions implemented such as factorial, isqrt, perm, comb, degrees, radians, exp, pow, ldexp, fabs, gcd, lcm, floor, ceil, remainder, expm1, fmod, log1p, trunc.

I was successful in implementing the trunc() function in the following manner:

```
@overload
def trunc(x: f64) -> i64:
    """
    Return x with the fractional part removed, leaving the integer part.
    """
    if x>0:
        return floor(x)
    else:
        return ceil(x)

@overload
def trunc(x: f32) -> i32:
    """
    Return x with the fractional part removed, leaving the integer part.
    """
    if x>0:
        return floor(x)
    else:
        return ceil(x)
```

For every function implemented, it is essential to have integration tests in place to check the validity of the functions. The following is the integration test created for `trunc()`.

```
def test_trunc():
    i: i64
    i = trunc(3.5)
    assert i == 3
    i = trunc(-4.5)
    assert i == -4
    i = trunc(5.5)
    assert i == 5
    i = trunc(-4.5)
    assert i == -4
```

Similarly I have begun working on the `string.py` module. The first function I implemented `capitalize()` is designed as:

```
def capitalize(s: str) -> str:
    """
    Return a copy of the string with its first character
    capitalized and the rest lowercased.
    """
    result: str
    result = s[0].upper() + s[1:]
    return result
```

As necessary, the integration test for the function is:

```

def test_capitalize():
    i: str
    i = capitalize("lpython")
    assert i == "Lpython"
    i: str
    i = capitalize("development")
    assert i == "Development"

```

These all take a singular variable in the form of float or integer. Currently, it is not possible to take in a list as an argument, but once implemented I would implement the fsum() function of the math.py module in the following manner:

The pseudocode for any list or array type iterable to find the full precision summation.

```

def fsum(iterable):
    """
    Full precision summation using multiple floats for intermediate values
    """
    partials = [] # sorted, non-overlapping partial sums
    for x in iterable:
        i = 0
        for y in partials:
            if abs(x) < abs(y):
                x, y = y, x
            hi = x + y
            lo = y - (hi - x)
            if lo:
                partials[i] = lo
                i += 1
        x = hi
        partials[i:] = [x]
    return sum(partials)

```

Fsum takes an iterable as input and returns an integer. Prod() is another such function to be implemented by taking as input an iterable and returning an integer or float value.

The integration test for fsum would be coded as:

```
def test_fsum():  
    i = f64  
  
    i = fsum([1, 4, 6])  
    assert i == 11.0  
  
    i = fsum([2.5, 2.4, 3.09])  
    assert i == 7.99
```

I would similarly work towards implementing all high-priority modules over the period of these 22 weeks. As soon as I'm done, I'd begin working towards the middle and low priority modules.

4) TIMELINE

I will strictly adhere to the following timeline. The tasks would be completed before the deadline.

Time period	Tasks
<i>Community Bonding Period</i>	
May 20 - June 12	<ul style="list-style-type: none">● Discuss and get pending pull requests merged● Discuss with the mentors on what communication medium would they prefer for updates regarding the project● Understand the codebase more thoroughly
<i>Coding period begins (from June 13)</i>	
June 13-June 30	<ul style="list-style-type: none">● Discuss function designs with mentors.● Implement remaining LPython Built-in module functions● Write integration/compile time tests for the same.
June 30- July 15	<ul style="list-style-type: none">● Implement remaining Math.py module functions.● Write integration/compile time tests for the same.
July 15- July 25	<ul style="list-style-type: none">● Implement the String.py module functions.● Write integration/compile time tests for the same.● Update the documentation with the latest GIFs and Images
<i>Phase-1 Evaluation (July 25- July 29)</i>	
July 25- Aug 4	<ul style="list-style-type: none">● Buffer week to complete the pending tasks and get them reviewed
Aug 5 - Aug 25	<ul style="list-style-type: none">● Implement the String.py and Numbers.py module functions.● Write integration/compile time tests for the same.

	<ul style="list-style-type: none"> ● Look back and fix bugs
Aug 26- Sept 11	<ul style="list-style-type: none"> ● Implement the Statistics.py module functions. ● Write integration/compile time tests for the same.
Sept 12 - Sept 27	<ul style="list-style-type: none"> ● Create good first issues to allow other contributors to contribute ● Fix bugs ● Implement the Decimal.py module functions. ● Write integration/compile time tests for the same.
Sept 28 - Oct 13	<ul style="list-style-type: none"> ● Implement the Random.py module functions. ● Write integration/compile time tests for the same.
Oct 14 - Oct 25	<ul style="list-style-type: none"> ● Implement the Time.py and Datetime.py modules functions. ● Write integration/compile time tests for the same. ● Create issues, guide students beginning to join for GSoC
Oct 26 - Nov 5	<ul style="list-style-type: none"> ● Complete the implementation of any previously remaining modules. ● Get started on implementing the Sys.py and OS.py module functions. ● Write integration/compile time tests for the same.
Nov 6 - Nov 13	<ul style="list-style-type: none"> ● Get pending pull requests merged. ● Discuss the future scope of the compiler ● Write a blog about my Summer of Code journey
<i>Evaluation (Nov 21- Nov 28)</i>	

5) PREVIOUS CONTRIBUTIONS

5.1) Pull requests


PR Number	Description	Status
#377	Implemented the trunc() function in math.py module for lpython compiler	Merged
#397	Made string.py and implemented capitalize() function	Open

5.2) Issues

Issue Number	Description
#376	Missing trunc() function in math.py module
#395	How to pass a list iterable in a function?

6) MY MOTIVATION

Python is a programming language with so many advantages that we don't even need to list them. It is also commonly acknowledged that code performance has been a major challenge for it. Many approaches to speeding up Python have been offered over time in response to this challenge. The purpose of these approaches is to inject a small amount of compiled code into Python to make it as fast as feasible with the least amount of effort on the programmer's part. I have been captivated by this idea and have always wanted to contribute to it and improve it from my end



since the first day I delved into the codebase of LPython. Thanks to this project, I'll be able to do so while being guided by such skilled and experienced mentors.

7) EXPECTATIONS FROM MENTOR

- Help me choose the best possible way when I have more than one way of implementation.
- Explaining to me some part of the codebase if I am unable to understand it on my own.
- Feedback for the work implemented.

8) POST GSoC

Post GSoC, I would be connected with the organization. I would keep contributing with the same zeal because LPython is more than an organization to me. From creating module functions from scratch to expanding on the documentation, I've been in touch with the project for enough time to understand why it was built and what impact it can have. GSoC would be a perfect opportunity to learn more while giving back to the community. While contributing seemed really daunting at the start, over time as I went through the codebase, issues, and PRs, it became more and more intuitive. I'll ensure that future members of the community are able to traverse easier.