



Project : Python Client Expansion

Organisation : InterMine

Google Summer of Code, 2018

About Me:

Username and Contact Information

- **Name:** Nupur Gunwant
- **Gender:** Female
- **Time Zone:** IST (UTC + 5:30)
- **Discord handle:** Patronus
- **Github id:** [nupurgunwant](#)
- **Twitter handle:** [nupur_gunwant_](#)
- **Email:** nupurgunwant2911@gmail.com
- **Medium handle:** [Nymeria](#)
- **Mobile:** +91 843-691-0222

Education

- **University:** [Indian Institute of Technology, Kharagpur](#)
- **Degree:** Integrated Master of Science
- **Field of Study:** Mathematics and Computing
- **Current year:** 3rd year(6th Semester ongoing)
- **Expected Graduation cum Post Graduation:** 2020

Personal Background

Hello, I am **Nupur Gunwant**, a third year undergraduate student at **IIT Kharagpur, India**. I'm pursuing a 5 Yr. Integrated M. Sc. degree in **Mathematics and Computing**. I am **proficient in C, C++, Java and Python**. I use a MacBook Pro working on MacOS Sierra, version 10.2.6. My primary text editor is Atom, as not only is it deeply programmable but also remarkably hackable.

I love sharing my ideas and find it immensely joyful to hear other's views. I believe that creating things is one of the biggest assets of mankind and thus, I feel open source is a great platform for people who love to share their work collaboratively and make the most of human mind to build a progressive society.

I have taken courses in **Programming and Data structures, Algorithms, Object-oriented programming** as a part of my curriculum. I started my journey in Python through an online course and later read a couple of books, **Python by O'Reilly** being my favourite. I kept on learning by working on projects, reading and I still keep doing so. Recently, I attended the '**Open Source Summit**' held in my college where there was an enormous discussion about advanced topics in Python.

I have been contributing to a number of projects which can be seen on my [Github profile](#). The most recent **projects in Python** I have contributed to are:

- **WHAT'S-IN-THE-NEWS-GEEK**: A CLI tool to bring all the latest news to your terminal.
- **SoCLI**: Stack Overflow command line written in python. Using SoCLI you can search and browse Stack Overflow without leaving the terminal. SoCLI was one of the trending projects on Github. It now has thousands of users around the world.

Apart from that, I am an avid reader and a lover of art, which make up the most of my conversations. I hope that my career someday lies in the intersection of my cognitive and creative abilities.

About the Project:

Abstract

InterMine integrates biological data sources, making it easy to query and analyse data. These data sources, more specifically data warehouses can be analyzed to perform queries, manage lists and are used to do functioning of biological data.

Python Client is a software intended to make use of these data warehouses of InterMine in a more efficient manner. It is composed of a number of client libraries which allow users to write scripts to access InterMine data directly via the command line. It is rudimentary for biologists and other researchers who need such information on a frequent basis.

Because of its huge application, it is crucial that features such as query management and easier access of different mines should be made possible using this client. The motive of this project will be to focus on all such issues and thus make the Python Client even more compliant with the needs of its users.

Benefits to the Community

InterMine is one of the most famous open source data warehouse built specifically for the integration and analysis of complex biological data. It is not just a collection of sophisticated web query tools but is also a major storehouse of the largest biological databases in the world. The user friendly and flexible web interface as well as the powerful and programmable API of InterMine have been making it popular since ever.

Python Client is an important part of InterMine and has huge amount of visibility on Bioconda and Depsy(total 13,000 downloads). A lot of users prefer it because it is much easier to write commands on the terminal, especially for frequent users.

The versatility of Python Client is remarkable and is another reason it has been getting popular with time. Adding a layer of features that are much needed by the users will not just make things easier for its present users but will allow a greater fraction of population to exploit its features and usage.

Contributions to InterMine

- [Addition of Loop constraints](#) : Loop constraints are used widely by all InterMine users and are an important type of constraint. Their addition in the Jupyter tutorials is quite helpful to all the users who wish to access them via command line.
- [Addition of Range constraints](#) : Range constraints are another variety of constraints that I added in the Jupyter tutorials.
- [Addition of get_list\(\) function](#) : The use of get_list() function that accesses a list from the user's InterMine account was another thing I added.
- Suggested a way to fix the unavailability of Python Client on Bioconda : Bioconda was displaying InterMine's Python Client as a package but was redirecting the user to a wrong link. I suggested a change in one of those files after testing on my machine, which resolved the issue.

Problems and Implementations:

I. Bridging organisms with their InterMines

Problem:

InterMine's custom Query Builder helps a user to create and save searches. In Python Client, every query is treated like an object and defines what we need to extract from the database. However, the user can make a query from only one database at a time.

Suppose, a researcher needs to extract genes of Yeast and Fruit fly. He will have to find and enter the URL of YeastMine and FlyMine separately and extract the desired results one by one. The process of query building can be done only within a single mine and there isn't a way yet through which we can extend this search amongst a collection of mines but we can make it easier.

Apart from that, if someone needs to retrieve some data about an organism and isn't sure about its mine, he will need the URL every time he checks a mine for his desired data. As a matter of fact, there have been instances where users have gone for the wrong URLs (bovinemine.org instead of bovinegenome.org)


We can work upon the solution by breaking this problem into parts and realizing the specific needs of the user:

⇒ Suppose a user has a query to make but he hasn't got a clue about the URL of the mine(s) to which the organism of his query belong. What we need in this case is a way to retrieve the mine(s) by entering a particular organism.

⇒ Suppose the user has a particular mine's name and wants to know which organisms are associated to it & other information about that mine (like URL, API version etc.). We need a function that extracts this information.

Solution and Implementation:

To satisfy the previously mentioned needs of the user, we have to get access to InterMine Registry. InterMine Registry is a place where all the up-to-date instances information are stored and can be consumed by applications. Suppose, I click on BeanMine in the [registry](#), this is what I get:

BeanMine

Description

A mine with common bean data from the Legume Info tripal.chado database

URL: <https://mines.legumeinfo.org/beanmine>


API Version: 26

Release Version: 0.8

Intermine Version: 1.8.5

Organisms: A. ipaensis, A. duranensis, A. thaliana, C. arietinum, G. max, M. truncatula, P. vulgaris

Neighbours: Plants

 LegumeFed

CLOSE

We will use requests library in Python along with the GET request to InterMine Registry RESTful API and access the data from InterMine Registry. Shown below is an example of a tested code that gives the following output:

Input:

```
import requests
r=requests.get("http://registry.intermine.org/service/instances/flymine")
print(r.text)
```

Output:

```
{
  "instance": {
    "_id": "5992e1ba0649be0011b21dfe",
    "id": "35",
    "name": "FlyMine",
    "url": "http://www.flymine.org/flymine",
    "created_at": "2017-08-15T11:57:46.104Z",
    "last_time_updated": "2018-03-10T00:00:15.925Z",
    "status": "Running",
    "isProduction": true,
    "twitter": "intermineorg",
    "description": " An integrated database for Drosophila genomics",
    "api_version": "25",
    "release_version": "45.1 2017 August",
    "intermine_version": "1.8.5",
    "__v": 0,
    "images": {
      "logo": "http://www.flymine.org/flymine/model/images/logo.png"
    },
    "colors": {
      "header": {
        "text": "#fff",
        "main": "#5c0075"
      }
    },
    "location": {
      "latitude": "52.199236",
      "longitude": "0.122280"
    },
    "organisms": [
      "D. melanogaster"
    ],
    "neighbours": [
      "MODs"
    ]
  },
  "statusCode": 200,
  "executionTime": "2018-3-10 11:07:26"
}
```

Now that we have access to this data, we can find the Mine(s) of a particular organism and we can find the data corresponding to a particular Mine. This data can therefore be used to define the following functions:

`getMine(species)` will return the Mine(s) to which the organism (that the user inputs) belongs :

```
>>getMine(A. duranensis)
BeanMine, ChickpeaMine, CowpeaMine, LegumeMine, PeanutMine,
SoyMine
```

Note: If no arguments are passed to `getMine()`, it will return the list of all the Mines.

`getData(Mine)` will return the data corresponding to a given Mine:

```
>>getData(BeaMine)
Description: A mine with common bean data from the Legume Info
tripal.chado database
URL: https://mines.legumeinfo.org/beanmine
API Version: 26
Release Version: 0.8
Intermine Version: 1.8.5
Organisms: A. ipaensis, A. duranensis, A. thaliana, C. arietinum,
G. max, M. truncatula, P. vulgaris
Neighbours: Plants
```

Coming back to the original problems that were faced in the past as described in the problem statement, we can now tackle them as follows:

⇒ Extraction of URL and other information about Bovine:
(We can get the desired URL of Bouvine Mine now)

```
>>getData(BouvineMine)
Description: An integrated data warehouse for the Bovine Genome
Database
URL: http://bovinegenome.org/bovinemine
API Version: 25
Release Version: 1.4
Intermine Version: 1.8.4
Organisms: B. taurus, C. hircus, O. aries
```


II. Creating a Query Manager

Problem:

Python Client provides the user with a feature to manage the lists using `list_manager`. This list manager can not just be used to create and access lists, but can also be used to obtain lists that are composed as a union, intersection or subtraction of the desired lists.

A similar way, however, does not exist for managing queries. The only feature currently in Python Client is to create a query with desired constraints and columns. Suppose the user wants to use a query he had generated some time ago, he will have to repeat the entire process that he had done previously. This happens to be quite inconvenient.

Queries are the basis of all research and being able to manage them will definitely be an asset to behold. With a query manager, the user can obtain a collection of all the saved queries, access the desired one and perform various other operations with queries. This will save an enormous amount of time and effort.

Solution and Implementation:

Every query is treated like an object and has two principle sets of properties(attributes):

- **its view:** the set of output columns
- **its constraints:** the set of rules for what to include

We will create a `Query_manager` class(that works similar to `list_manager`) to manage query contents and operations. A `Query_manager` instance will be a dictionary containing the query name as the key and query object as its value.

object --+

|
QueryManager

This class will contain all the useful methods which delegates to this class, while other methods are more easily accessed through the query objects themselves.

We will accomplish this by making use of [InterMine API](#). The major functions that it will contain are as follows:

⇒ `qm` is here an instance of the class `Query_manager`. The `create_query()` method of this class will create a new query and store it in the InterMine account of the user. We will make use of `POST /user/queries` to save the query in InterMine account of the user and thus the query will remain available at later sessions. The arguments of `create_query()` are corresponding to the attributes of a query object. Here, 'view' is used to add columns to one specific query we are creating in `Query_manager` (just like `add_view()` is used to add columns to the query).

```
qm = service.Query_manager()
qm.create_query(query_type="Gene",name="my query 1",
view=["Gene.symbol","Gene.primaryIdentifier", "Gene.length"])

//creates a query in query manager
```

⇒ By using `get_all_query_names()`, we can get the names of all the queries we had created and saved in the past. We shall make use of “`GET /user/queries`” and “`GET /queries`” to achieve this.

```
qm.get_all_query_names()
//returns a list of all queries
```

⇒ By using `get_query()`, we can make use of a previously generated query that had been saved in the user's account. This will be done again by using “`GET /user/queries`” and “`GET /queries`”.

```
q = qm.get_query("my query 1")
//accesses query named 'my query 1'
```

⇒ We can create more functions such as to delete the queries from the user account by using `DELETE /user/queries/name`.

⇒ A query can also be used to export queries in BED, FASTA and GFF3, csv, tsv format using the corresponding API requests.

In this way, the user can not only save time to generate a previously used query again but will also be given the independence to operate with queries.

III. Adding Visual features

Problem:

At present there is no visual representation of the output that is generated by the Python Client. If we are able to present the output in a more understandable way, it shall make the Client even more useful.

Solution and Implementation:

Proposed way: Using Matplotlib

[Matplotlib](#) is said to be the grandfather of all Python visualization packages that we use at present. It is the oldest python visualization library and to top it off, it is open source too. This will be a plus point in its future maintainability.

The only steps to use Matplotlib will be:

(For user)

- Export the .csv file.
NB: This step can be done via command line once we complete the implementation of the last part of the previous problem, ie “saving queries in csv format”.

(For developer)

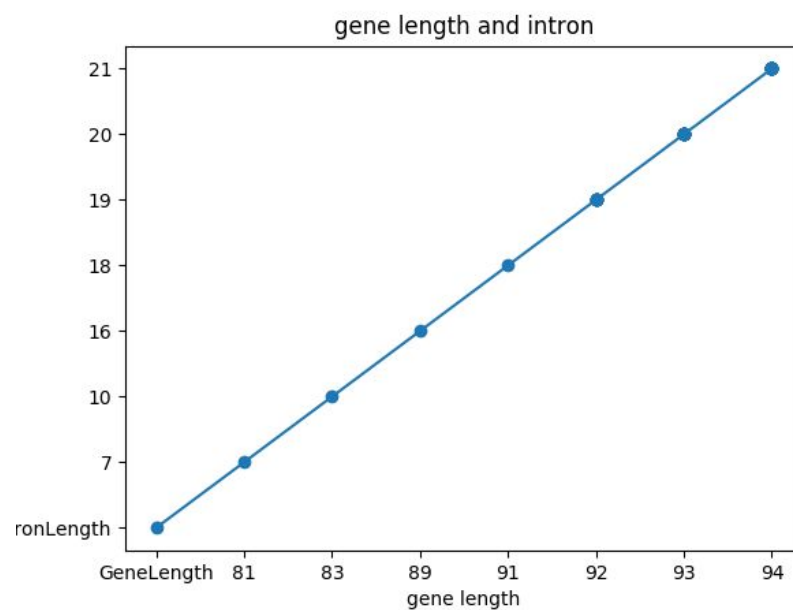
- Add matplotlib as a dependency.

For the data in the table above, the following graph was created from its corresponding .csv file by using this code. Note that here a function called `plot_graph()` can be defined, which will enclose the below Input code. In similar way, we can create `bar_plot()`, `pie_chart()` and other plots as per the needs.

Input:

```
import matplotlib
matplotlib.use('TkAgg')
import matplotlib.pyplot as plt
import csv
x=[]
y=[]
with open('results.csv', 'r') as csvfile:
    plots= csv.reader(csvfile, delimiter=',')
    for row in plots:
        x.append(row[0])
        y.append(row[1])
plt.plot(x,y, marker='o')
plt.title('gene length and intron')
plt.xlabel('gene length')
plt.ylabel('intron')
plt.show()
```

Output:



Alternative way: Using Plotly

To achieve this, we need to introduce [Plotly](#) as a third party program to support visualization. We will use the Python API of Plotly to fulfill our purpose. Plotly is also a well documented and widely used Python visualization library.

This feature needs a few dependencies to be included and a few steps to be done by the user. Even though this is an extra step from both the sides, the benefits of being able to visualise data outweigh the little efforts. The steps are as follows:

(For the user)

- Sign up on Plotly. It is free.
- Open profile and regenerate an API_key
- Set credentials:

```
import plotly
plotly.tools.set_credentials_file(username='DemoAccount',
api_key='lr1c37zw81')
```

- Export the query/list from Flymine(or any other InterMine) in csv format.

(For the Developer)

- Installation of plotly, pandas:

```
pip install plotly
pip install pandas
```

After doing these steps Plotly can be used from the terminal to represent the data stored in the .csv file as per our need. The user needs to make sure that his current folder on the terminal is the one in which the csv file is store. After the above steps, plotly can be used to make more complex plots. However, the plots will be shown in a new browser window of the user's Plotly account, unlike Matplotlib where its shown offline. Here is a tested example of code that creates a table of statistics if the given data:

```
import plotly.plotly as py
import plotly.graph_objs as go
import plotly.figure_factory as FF
import numpy as np
import pandas as pd
df = pd.read_csv('results.csv')
sample_data_table=FF.create_table(df.head())
py.plot(sample_data_table, filename='sample-data-table')
```

Output:

GeneLength	IntronLength
81	7
83	10
89	16
91	18
92	19

The choice of plots can vary as per the needs of the user. We can add more complex plots. But the priority will be the quality of functions (that plot data) and their documentation, rather than creating a lot of functions without a proper documentation. The final list of plots that should be included as well the final library(Matplotlib or Plotly) will be finalised before the coding period starts.

IV. Making region search compatible(Optional)

Problem:

The region search feature helps a user to search for features that overlap a list of given genome coordinates. The region search feature is present in the web application of InterMine and has been written in Python as well to serve as an extension to the main repository of Python Client. However, it hasn't been updated to work for Python 3.x.

Apart from this, a lot of people are not aware about the extension which decreases its usage. So, converting it from an extension to a feature in main repository will be better and more user-friendly. This will ensure region search via Python Client without any additional requirements.

Solution and Implementation:

The region search feature is currently not accessible to the Python Client because after the addition of Strand Specific feature, the API was not propagated and is hence in need of a fix in the core InterMine repository.

If during the GSoC period, this issue gets fixed then Region search can be added to the Python Client using the InterMine API.

Documentation and Testing:

Documentation

The documentation will consist of two parts:

1. **Docstrings:** A docstring is a string literal specified in source code that is used, like a comment, to document a specific segment of code/ function. These will be added while writing the code.
2. **HTML Documentation:** This will consist of files written in reStructuredText(**RST**) format. The documentation can be generated from RST source using [Sphinx](#). This process is managed by [ReadTheDocs](#), an online service for maintaining documentation. It will be further added into [im-docs](#), a repository maintaining all the HTML documentations of InterMine.

This documentation will consist of the following:

- What feature/change the code brings.
 - Installation instructions.
 - A small code example explaining the use.
 - How to get support (invite link to support channel / contact email)
 - Information for people who want to contribute back
3. **Jupyter Notebooks:** This will consist of files written in ipynb format, describing every feature in detail. The documentation can be generated from using Jupyter notebooks, online or offline. These files will be added in [intermine-ws-python-docs](#), which is a repository of python tutorials till date. The steps for setting up Jupyter notebook are described in the repository's README file.

The reasons for using Jupyter notebook is the fact that they are not only easy to use and have a friendly user interface, but also that most of the features of Python Client have been described through them. So, a user can find everything about the Python Client in one single place.

Testing

We will be using [unittest](#) library of Python testing framework for serving our purpose. The unittest module can be used from the command line to run tests from modules, classes or even individual test methods:

```
python -m unittest test_module1 test_module2
python -m unittest test_module.TestClass
python -m unittest test_module.TestClass.test_method
```

Thus, the unittest module provides a rich set of tools for constructing and running tests and fits our requirements.

Timeline:

Community Bonding period(April 23- May 14)

The most important task I will be committing to during this time is getting familiar with the codebase and community. I am already pretty familiar with the Python client, and have been using all its existing features. However, there are some functions that were written long time ago and haven't been used much or aren't compliant with the updates and hence, getting to know their usage (or removing them if undesirable) is another major task.

I will also spend this time to gain experience in **writing tests** and will devote a large part of the month for this. I will **constantly stay in touch** with the mentors and the InterMine community and will make sure I **clear my doubts** regarding any issues I face that time as well as discuss about the factors that may lead to increasing the usability of the Client.

By the end of this period, my aims are to:

- Learn thoroughly about writing tests.
- Learn about the codebase of Python Client deeply and cover most of it.
- Discuss with the mentors about what all can be done to increase the reach of Python Client, other than addition of features.
- Finalise the plots that should be included in Python Client's upcoming visual features.

Coding Period begins(May 14- August 14)

Week 1(May 14 - May 20): Feature I

This period will be devoted to writing code for the first feature i.e. bridging organisms with their InterMine. To, accomplish this, I will access data from the InterMine Registry using RESTful API and GET requests. I will be writing the following functions by making the desired extractions from this data during this period.

1. **getMine(species)**: A function that will be used to access the InterMine of a given species.
2. **getData(Mine)**: A function that will be used to access the Data corresponding to an InterMine.

Week 2 and 3(May 21 - June 3): Feature II

This period will be devoted to writing code for the second feature i.e. adding a query manager to Python Client. During this time, I will focus on how different [InterMine API](#) can be helpful in managing queries. I am planning to make a Query_manager class that contains the following functions to make query management easier:

1. **create_query(*args)**: This function will be used to define and save a new query.
2. **get_all_query_names()**: This function will be used to see all the queries that have been saved.
3. **get_query(query_name)**: This function will be used to access a particular query.
4. **delete_query(query_name)**: This function will be used to delete a particular query.
5. **convert_BED(query_name)**: This function will be used to access a particular query and convert it to the given format(here BED).

Week 4(June 4 - June 12): Improvements from feedback

In this period:

1. I will focus on the making changes and **improving my code** as per the feedback from the mentors on the work done up till now.
2. I will **add any extra features** if required.
3. Apart from that, I will make sure to **complete everything I had planned** during the course of these four weeks and ensure it is **devoid of bugs**.

Phase 1 evaluation

Week 5 and 6(June 15 - June 28): Tests for Feature I and II

After Phase 1 Evaluation, other major tasks for 2 week period will be:

1. Covering all the internal functions with **tests**.
2. Finalize the code written so far for **rigorous final testing** and **code cleanup**.

I will make sure the tests are **well documented**, so that they can be well understood by developers trying to improve the tests. I will also ensure their **maximum coverage**.

Week 7 and 8(June 29 - July 12): Feature III

I will be working on the visualization feature during this time. I will integrate Plotly/Matplotlib with the Python Client to embrace the usefulness of the output data in the form of various plots Plotly/Matplotlib provides. The major tasks in this time period will be to:

1. **Incorporate the various plots** that can be used to represent different queries by adding functions for each of them. This will let the user plot the query using Python Client.
2. Use **Jupyter tutorials** to describe the use of commands to make respective plots.

Phase 2 evaluation

Week 9 and 10(July 14 - July 27): Documentation

After all this, the next task will be to document the code written uptill now. This is a very important task as it will lay foundation for the developers who try to understand and modify the code in future. I will focus on two things during this period:

- I will make sure all the features covered up till now are **well documented** to enhance their future value.
- I will use **Jupyter notebooks** to cover tutorials for the above features so that the users can easily utilize them for their benefit.

Week 11 and 12(July 28 - August 10): Bug feedback / Optional task

By this time if everything went well, I will have completed all my targets. In these two weeks, I will work on the optional task of adding the Region search feature in the Python Client. If I realize any further modifications that I feel might enhance the usability or maintainability of the Client, I will discuss them with my mentors and work on them in this period and further, if possible.

GSoC Period ends

Apart from the above mentioned schedule, I will be:

- **Pushing code to my fork daily** so that my mentors can evaluate and keep track of whatever work I am doing.
- **Blogging every week about the progress and related experiences** in the said week so that mentors and others can get an overall summary of my week's work.
- **Sending a detailed Pull Request** to the main master branch as soon as the code is ready and cleaned-up. I will make sure it is well-documented and constitutes of unit tests, preferably before each evaluation deadline.

Software packages to be used:

Languages:

Python

Modules:

intermine-ws-python, intermine-ws-python-docs, unittest, plotly/ matplotlib, sphinx

GSoC:

Have you participated previously in GSoC? When? Under which project?

No, I have not participated in GSoC before. This is the first time I am participating in GSoC.

Are you also applying to other projects?

No, I am not applying for any other project in any other organisation.

Commitments

I have no commitments from 25th April 2018, the day my University holidays start.

My classes for the next semester will begin on 13th July, 2018.

Due to very less academic load during the start of the semester, I will be able to dedicate around 7 hours everyday for the last 4 weeks of the coding period.

Nevertheless, I have distributed my tasks in such a way that the last 4 weeks will only be given to documentation and will act as a buffer period.

Eligibility

I am eligible to participate in Google Summer of Code. For any queries, clarifications or further explanations, feel free to contact nupurgunwant2911@gmail.com.
