Programming for Science Fairs

A student's guide to resources, usage and display

Rajalakshmi Kollengode

Contents

Preface

There is a wide variety of both free and paid resources available on the web. Many of them focus on gaming as a tool to learn elements of programming. Many other sites focus on coding for content creation (as in blogs), graphics design, collaborating and marketing. ***Not many of these are directly relevant to Science Fairs***. While doing a science fair project, the student's programming needs depend on the category and field of study the project belongs to. We can broadly divide the projects into ***two*** groups: (a) computational sciences (software engineering, robotics, informatics etc.) and (b) non-computational sciences (behavioral science, biochemistry, biology, chemistry, physics, engineering and mathematics excluding algorithms for numerical methods). For computational sciences, the main line is programming irrespective of field of study. In this case, the project would need a great amount of coding. If the project is not on computational sciences, some elements of programming can be included in the project: For e.g. most projects would involve some sort of data gathering (through experiments and/or through questionnaires) and this data is bound to contain some noise. Software resources are available to mitigate the noise issues.

The goals here are (i) to cut through the clutter to provide the resources that are directly applicable to the student's needs in carrying out their science fair projects and (ii) to provide some guidelines on how to utilize programming for non-computational projects such as those in physics, chemistry, biology, biochemistry and social studies.

Those who are proficient in programming can skip section 1.1 and go to 1.2. And those who are familiar with available software libraries can skip section 1.2 (and up to section 1.7) as well. If you have suggestions on additional resources, please email me at sciencefaircoding@yahoo.com with the title "Software Resources for Science Fairs".

Rajalakshmi Kollengode

January 2019

Sunnyvale, CA.

1. Overview of Available Resources

We can broadly divide the resources into *two* categories: (i) learning resources and (ii) library/component resources. Learning resources help students learn the basics of programming. The fundamentals of programming can be broken into three parts: (i) computer science basics (fundamental algorithms, data structures), (ii) programming language basics, and (iii) tool usage basics (editors, compilers, debuggers, IDEs etc). These resources are outlined in section 1.1.

The component/library resources provide ready-made components that can be targeted using the APIs (Application Program Interfaces) provided. Such resources are very helpful in carrying out science fair projects: for e.g. say the project is on a new algorithm for detecting the type of a plant by doing an analysis on the shape of the veins of the leaves. To input shapes of leaves into this new algorithm, to analyze the correlation of patterns on the leaves, and to output the identification results many ready-made components can be used. These resources are listed in sections 1.2 through 1.7.

1.1 Learning Resources to Learn the Fundamental Concepts

The top resources for learning to code include Codeacademy, Coursera and Udemy. We recommend learning the basics first: At Codeacademy, the Computer Science Path (https://www.codecademy.com/paths/computer-science/) provides the essentials of programming using Python as the language. At Coursera, several introductory classes are available to learn most common computer languages and algorithms. Python, Java, R, C/C++ are the top programming languages for usage in science fairs. And the reason is that a lot of readymade components applicable to a wide variety of scientific applications are available in these languages. Almost all classes at Udemy are paid: when they run 'sales', many classes are available for less than $15.00. To this list we must also add GNU Octave (free) and Matlab/Mathematica (paid) for quick experimentation with scientific concepts.

To get started it is best to use online tools to edit, compile, run and debug the program. Deferring learning about development tools (editors, compilers, debuggers, IDEs etc.) to a later time keeps the focus on learning to code and eliminates distractions. Few of the online tools are

https://www.tutorialspoint.com/python3_terminal_online.php

https://py3.codeskulptor.org/

https://www.tutorialspoint.com/compile_java_online.php

https://www.onlinegdb.com/online_c++_compiler

https://rextester.com/l/r_online_compiler

The main learning goals are threefold: (i) on the language front, the main parts include syntax, basic input/output, loops, classes, inheritance, interfaces; (ii) on the computer science (CS) front, the goal is proficiency in basic data structures (D/S) and algorithms including arrays, linked lists, stacks, queues, trees, graphs, sorting, searching and hashing; (iii) on the tool usage front, proficiency in using editors, compilers, debuggers, IDEs, and project management. At this level of knowledge, students are ready to include some elements of programming in their science fair projects.

After using an online tool to get proficient in **all the three** elements of programming in one programming language, we recommend downloading a full-fledged IDE such as Eclipse (https://www.eclipse.org/downloads/) or VisualStudio (https://visualstudio.microsoft.com/vs/). Eventually for collaborating with others, Github (https://github.com/) is very helpful. And packages are available to integrate Github with Eclipse and VisualStudio.

After attaining proficiency in basics, the next step would be to move onto more advanced programming by using readily available components: For e.g. as to the data cleaning operation mentioned in the preface, ready-made components mentioned in the next section can be used.

1.2 Other Resources - Libraries and Packages

Advanced software applications for science fairs almost always use existing libraries as building blocks since many of the advanced algorithms are coded and readily available. These libraries are available mostly free of cost. We list below resources for a wide spectrum of applications.

Both Java and Python languages have built-in support for basic data structures like lists, stacks and queues. Some common libraries in Java include Apache Commons, and Guava. Additional packages are available in Colt (https://dst.lbl.gov/ACSSoftware/colt/).

In C++, the Standard Template Library (STL) has support for all basic data structures. If sets and maps in STL are not sufficient, additional tree classes from boost library (https://www.boost.org/) can be considered.

Most of the science, math, and machine learning packages are in Python, R, C++, and FORTRAN. We recommend **Python and C/C++**.

In Python the advanced math and science APIs are available in NumPy, SymPy and SciPy. Sci-kit learn (https://scikit-learn.org/stable/), Microsoft Cognitive Toolkit (https://www.microsoft.com/en-us/cognitive-toolkit/) and Google's Tensorflow (https://www.tensorflow.org/) make it easy to develop machine learning applications.

There is another class of programming commonly used in science fair projects: microcontroller programming as in Arduino programming. And most microcontroller vendors (Arduino, Qualcomm, TI etc.) do provide a wide array of library functions.

Occasionally, third party libraries may have to be imported into these microcontroller programming environments. And most microcontrollers including Arduino provide such a facility. For additional details please see https://www.arduino.cc/en/hacking/libraries. Typically microcontrollers have limited memories, and C/C++ APIs might have to be tuned for memory consumption.

For other topics such as math, biology, chemistry, physics, and social studies we list below some of the fundamental software resources.

1.3 Resources for Math

Aside from NumPy mentioned earlier, additional resources include ScientificPython (https://pypi.org/project/ScientificPython/), AlgLib (http://www.alglib.net/), Intel's MKL (in C-Language), Eigen (http://eigen.tuxfamily.org/), Gnu Scientific Library

(https://www.gnu.org/software/gsl/), and dlib (http://dlib.net/). Datamelt (https://jwork.org/dmelt/) has good collections of mathematical APIs in Java and can be called from Python and Ruby.

1.4 Resources for Physics

Astropy, ProjectChrono (https://projectchrono.org/), Open Dynamics Engine (https://www.ode.org/), Computer Physics Communications Library (http://www.cpc.cs.qub.ac.uk/) are some commonly used resources.

1.5 Resources for Biology

ScientificPython, BioPython, GenoCad (https://genocad.com/), Vcell (http://vcell.org/), BioConductor for BioInformatics (http://www.bioconductor.org/), BioJava (https://biojava.org/wiki/Main_Page/) cover a wide spectrum of applications in biology.

1.6 Resources for Chemistry

PyMOL (http://www.pymol.org/), OpenBabel (http://openbabel.org/wiki/Main_Page), OpenChemistry (https://www.openchemistry.org/) Apache Chemistry (http://chemistry.apache.org/) are the leading packages.

1.7 Resources for Social Studies

PsychoPy, BeautifulSoup (to crawl the web and gather data), R (R Foundation for Statistical Computing), GNU PSPP (https://www.gnu.org/software/pspp/pspp.html), IBM's SPSS (paid), GraphPad Prism (https://www.graphpad.com/ , paid) are recommended.

2. Usage Considerations

Here we discuss some ways to include programming in science fair projects. The usage considerations do depend on the project. We attempt to include a broad array of projects for considering the usage of programming.

2.1 Usage for Computational Projects

For computational projects, the application of programming is direct. The first step is to pick the appropriate libraries for the development environment intended for the project. The next step is to test the libraries before using them for the project. *Beware of bugs, bugs, bugs!* Every API or function used should be tested with a few known examples. This process of doing unit testing (in addition to what others have done on their systems) is akin to calibrating experimental apparatus: Given the complexity of software systems, some bugs may be exposed in one particular combination of operating system, downloaded library's version and hardware. And other users may not encounter these bugs because of differences in hardware and/or operating systems etc. Picture this: the API used is buggy and the results shown on the board are incorrect and a smart judge figures it out – the entire effort comes down crushing and that definitely is a not a great situation to be in. Testing the software components used in the project is well worth the effort.

And if the standard verification tests do not pass, an older version of the software that presumably is more robust can be tested for consideration. Or an alternate package can be considered. After completing these two steps, the student can pursue the goals of the project.

2.1 Usage for Non-Computational Projects

In many cases, the application of programming is not straight forward for projects in physics, chemistry, biology, mathematics and social studies. We provide a few guidelines on how programming can be included for non-computational projects.

The key idea is this: *opportunities exist almost at every step to include coding. However, locating these opportunities need some thought.* Some of the phases that are conducive to applying programming include: exploratory phase (simulation of the theory), data collection phase (sensor programming, noise mitigation), data analysis phase (statistics, AI etc.), and correlation phase (regression fits can help find the relationship between variables, and help compare and contrast with other studies). Software is powerful and we can utilize software for

every repeatable task/procedure. Software can furthermore provide some elements of inference and correlation.

In every phase in which coding can be included, there are two ways to use available software resources: (i) use standalone software packages like GNU Octave and Matlab or (ii) create new programs in a typical programming language like C/C++, Java, Python, and R, and these new programs can use libraries outlined in sections 1.2 through 1.7. Unit testing needs to be done on both the functions used in standalone packages and the APIs used in the new program. As to choosing between a standalone package and writing a new application, our recommendation is to build on the student's existing knowledge and pick one set of components that is closely related to the current level of programming knowledge the student has: the goal for the students is to challenge themselves to learn some new methodologies without sacrificing accuracy. We next illustrate how to include coding using some examples.

In many projects some sort of data collection happens. The data collection can happen from experiments or from surveys. All data are likely to have some noise and this noise can be mitigated using one of the several filters available in many of the packages we discussed including SciPy (in signals), AlgLib etc. Alternately, a standalone package like GNU Octave can be used to filter the noise.

For mathematical projects, say the project is about a new approximate solution to a set of equations: this project can use simulations to verify. Simulations can also help during the exploratory phase by providing a feel for the solution: say the project is on finding an approximate solution for the trajectory of a rocket as it approaches say a planetary body. The shape of potential trajectories can be visualized using simulations of Newton's equations of motion. The attempted approximate solution should mimic this simulated shape.

As an example of an engineering project, let's consider a project that determines the terminal velocity of a freely falling object say, a cheetah jumping to catch its prey. This project can use software to simulate such falls in addition to the theoretical considerations. Alternately, the student can write a program that provides the terminal velocity given the drag coefficient and other parameters. This program can then be used to project several scenarios: for e.g. what drag coefficient is needed such that the impact speed is less than a certain speed so as to reduce the risk of bodily injuries.

Say the project is in biology for e.g. to find the impact of some substance/organism on plant growth: the collected data can be filtered and processed using regression fits. Regression fits are particularly useful for correlating data when their relationship is non-linear. Next let us consider a project in social studies that attempts to capture online bullying behavior. Here,

there is a need to crawl the web and collect data: Many libraries like ScraPy and BeautifulSoup can be used. And the collected data can be analyzed using GNU PSPP.

One needs to use caution while scraping for data on sites in which an account is needed: as part of the account sign-up process, such websites might place restrictions on scraping. Facebook is one such company that places restrictions.

3. Coding Display Considerations

Now all the hard work is accomplished and it is time to display the project in all its glory. *What aspects of the programming parts should be displayed on the project board, and by the board*? The answer depends on the impact of the algorithms/software on the project.

3.1 Display Guidelines for Computational Projects

If the project is heavy on computational sciences (Bioinformatics, Software Engineering, Numerical Methods etc.) several pages on the board would refer to it: objective/hypothesis portion, the procedure portion, the design considerations portion, and the results portion.

As an example of a predominantly software oriented project, let us consider a project on an innovative shape detection algorithm to identify the type of plants discussed earier. For this project elements of programming would show up across the full spectrum: motivation, goals, data gathering, data analysis, and results.

The next question is this: **how much of the code should be printed out and listed next to the board**? The main goal is that the materials displayed should reveal the full scope of the project. For computational projects *all* code that forms the core component of the project should be listed: This core component would consist of code written by the student and potentially some code written by others. *The code the student wrote should be demarcated from the code others wrote.*

Only the name of the libraries and packages used should be listed. Code from the libraries should not be listed.

3.2 Display Guidelines for Non-Computational Projects

For non-computational projects, the scope for displaying programming is limited: limited to the portion in which programming was used. Say for a mathematical project some initial

explorations were done with programming, the display of programming related material is limited to the motivation and goal sections.

As to listing code, the same recommendations outlined for computational projects apply: code written by the student should be listed and clearly demarcated from code written by others for the project. Many of these non-computational projects may use a standalone package instead of developing a new application. In such cases, code written to use these packages should be listed.

There is no need to list the lines of code from libraries/packages used by the project. Listing just the name of the libraries/packages is recommended.

Afterword

Dear Student,

I want you to have fun exploring new topics and learn a thing here, a thing there and eventually everything would fall in place. The key here is to get a growth mindset. It is ok to fail: just that we should learn from failures as well as successes. Hope this guide helps you and if you have additional suggestions, please email me at sciencefaircoding@yahoo.com and title it as "Software Resources for Science Fairs".

Code away to decode the mysteries Big Bang has provided us in 13.8 billion years.

Rajalakshmi Kollengode