INTERNATIONAL JOURNAL OF EDUCATION AND INFORMATION TECHNOLOGIES
DOI: 10.46300/9109.2021.15.13 Volume 15, 2021

Python Computational Web Apps for STEM Engineering Education

V. F. Ochkov*, A. A. Sutchenkov*, A. I. Tikhonov*
* Moscow Power Engineering Institute (MPEI), Moscow, Russian Federation

Received: June 13, 2021. Revised: July 5, 2021. Accepted: July 12, 2021. Published: July 15, 2021.

Abstract - **The article discusses STEM technologies and tools used in engineering education for scientific and technical calculations in Python, which allow to make classes visual and fun for students. An integrated environment that supports all the stages of solving computational scientific problems from their formulation, solution to their sharing is considered. Dash, Panel, Voilà and Streamlit technologies for publishing computational web applications for multivariate calculations are discussed, a comparison of these technologies for use in the educational process is made. Web applications allow computational experiments, but prevent changes to the source code, eliminating the appearance of errors. The integrated environment includes a generator of static sites for publishing smart tutorials with embedded web applications. This allows publishing student-developed applications along with learning materials with a minimum of effort. The integrated environment is convenient for both face-to-face and distant learning.**

Keywords - STEM, Python ecosystem, computational web apps, Jupyter, Dash, Panel, Voilà, smart tutorials

I. Introduction

The application of STEM technologies in engineering education [1, 2] involves creating a large number of calculation apps used as visual demonstrations for studying new material, multivariant calculations and computational experiments, solving inverse problems – identification of input parameters by known output parameters.

The acronym STEM may be accompanied with symbol A, turning it into STEAM (Science-Technology-Engineering-Art-Mathematics), adding creative elements to the learning process [3-7].

STEM-technology support for solving scientific problems should include tools for their description, application development, multivariate calculations and computational experiments, presentation of the results in both tabular and graphical form and publication.

Nowadays proprietary software packages Mathematica, Maple, Mathcad, Matlab and freely distributed Octave, Scilab, Smath are used as such platforms. All of them are based on high-level programming language, integrated development environments (IDE), application libraries, scientific visualization tools. The listed software packages differ considerably in functionality, some of them use concept of computational document – a uniform environment, allowing to describe a problem, to make calculations, to present results. Such environments are either integrated into the IDE or use Jupyter Project [8].

The Moscow Power Engineering Institute (MPEI) mainly uses Mathcad, Matlab, Scilab, and the Python ecosystem in educational process. The advantages of the latter platform are free of charge, growing popularity, and an extensive set of libraries. The difficulty of mastering scientific and technical calculations in Python is comparable with Matlab and higher than in Mathcad and Smath, that in turn requires crash course on Python programming, acquiring skills in work with Jupyter Notebook, NumPy and matplotlib [9].

In our opinion, the best choice for the development environment is the Python ecosystem due to the huge number of available libraries and Jupyter Notebook (JN) and JupyterLab (JL). JN and JL allow embedding formatted text, formulas, images, videos, application source code, calculation results including those in the form of graphics and animation into computational documents - notebooks. JN and JL are great technologies for developing computational applications, but the ability to modify source code leads to bugs and sometimes vandalism. This makes it necessary, when publishing applications, to turn to tools that preserve the functionality and user interface of the applications, but exclude their modification.

As an example, an interactive web app for simulation crystal growth using the Diffusion-limited aggregation (DLA) model [10, 11] (Fig.1) implemented in Python in Jupyter Notebook [9], Voilà [12] and Streamlit [13]. The app makes it possible not only to demonstrate the growth of fractal aggregates, but also to investigate the influence of diffusing particle concentration, deposition conditions, and analyze the fractal dimensionality of aggregates. The use of NumPy library to process arrays, SciPy to operate with aggregate neighborhood and diffusing particles, matplotlib library for visualization and animation, ipywidgets to build user interface allowed to make the application compact (140 lines of source text – DLA modeling, 30 lines – user interface and animation), to perform simultaneous simulation of system with thousands of diffusing particles, in contrast to original method [10]. This example reflects as in a drop of water the problems that arise when using STEM technologies in teaching science and engineering disciplines:

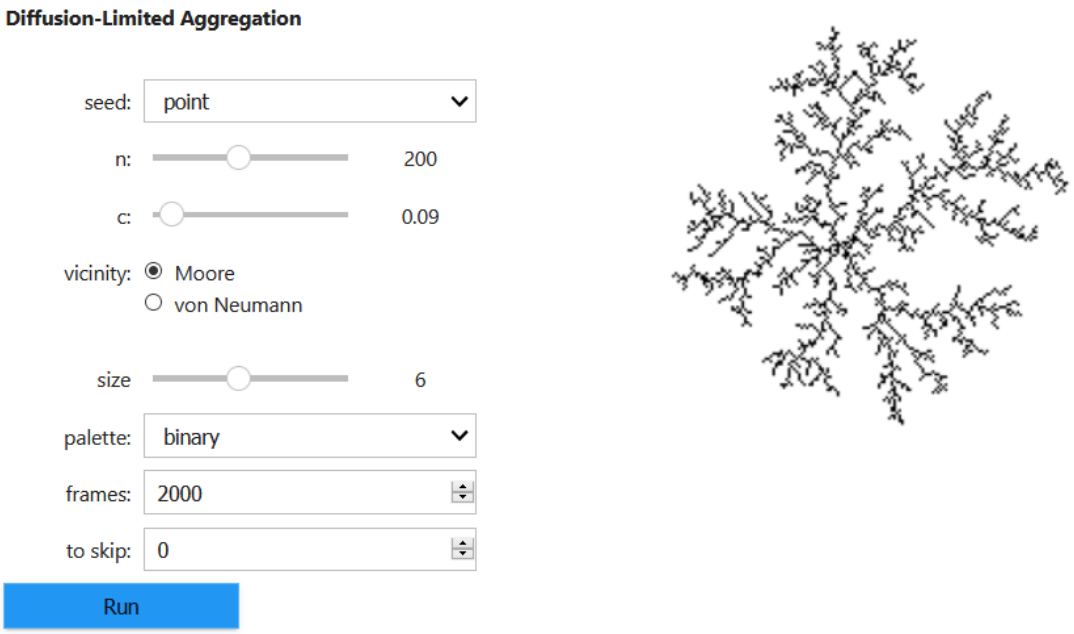

Figure 1. Interactive web app for simulation Diffusion-limited aggregation (DLA)

II. Computational Web Apps in the Educational Process

As IDEs in the learning process are mainly used Jupyter Notebook (JN), Jupyter Lab (JL), and starting since 2019 Visual Studio Code [14], which supports running JN, but at the same time allows to debug Python modules.

JN supports full development cycle. Markdown markup languages and, if necessary, HTML provide work with formatted text, images, hyperlinks, videos, Latex with mathematical formulas. By installing additional modules, it is possible to pretty format the source text of the application at the click of a button [15] or to turn a Jupyter notebook into a managed slideshow [16] with running applications to use in lectures (Fig.2).

To create a slideshow, it is sufficient to make a simple markup of the notepad cells that determines the direction of the slide transitions. It made possible to refuse PowerPoint presentations, as JN slideshows allow you to combine the problem description, the implementation of source code, if necessary, the problem solution.

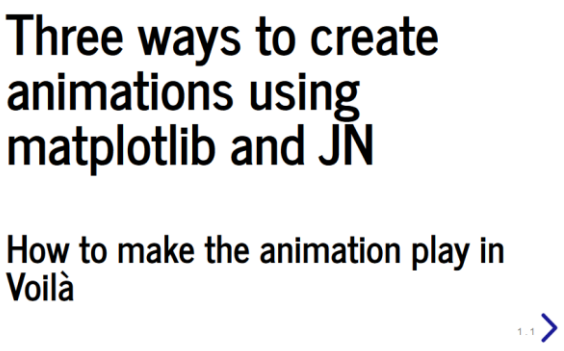

Figure 2. Slideshow in Jupyter Notebook

As a result, we get fully designed interactive applications that enable us to perform computational experiments on our computers.

In the context of the Covid-19 pandemic, the question of sharing interactive web apps, and in some cases use them in virtual laboratories, becomes acute. In what follows, the transformation of developed computational applications into standalone web apps will be referred to as *publication*.

The publication assumes the minimum labor intensity of transferring the functionality, user interface, hiding the source code of the applications, controlling access to the applications. Hiding the source code is conditioned by providing protection against unauthorized changes, additions and vandalism.

Most of the currently used platforms for scientific and technical calculations have subsystems for publishing web apps, for example, PTC Mathcad Gateway [17] is used for Mathcad, and MATLAB Web App Server [18] is used with Matlab. Publishing web apps with application servers is quite transparent and involves adding widgets to the application. In the authors' opinion, the most flexible and balanced tool for publication is Shiny [19], developed for the system of statistical calculations R.

Specialized tools for publishing computational web apps appeared in the Python ecosystem in 2018. Until then, it was necessary to use general-purpose web frameworks such as Flask and Django [20] for this task. In the context of educational process this approach is unacceptable due to the fact that the time of transformation into a web app exceeds the time of creating a computational application in Python and requires mastering additional technologies, including HTML, CSS, JavaScript.

III. Dash, Panel, Voilà, Streamlit

Let's take a look at the current Python ecosystem technologies for publishing computational web apps in terms of their use in the learning process and compare them to each other.

A. *Plotly Dash*

Dash [21, 22] is the most mature technology for publishing computational web apps built on top of the Flask web framework. Dash has built-in interactive visualization tools based on the plotly library, but other visualization tools, such as matplotlib, can also be used. It has an extensible set of user interface widgets. From the point of view of educational process this is quite important, because the possibility of creating non-standard components allows to use the technology for virtual laboratory works and for gamification of calculation applications. Non-standard components are written using React [23].

Unlike "pure" Flask, the creation of the user interface and interaction with the user is written entirely in Python without the need for additional technologies. For each event, for example, the passing of the mouse cursor over a diagram element, it is necessary to write a callback function. There is a library [24] that allows you to develop Dash applications in the JN environment, visualization with plotly library [25] is well integrated into Jupyter notebooks.

It should be noted that Dash web apps are well scalable client-server applications, a significant part of them functions on the client, the built-in visualization reacts to the actions of the user without the need to contact the server. Moreover, Dash applications can be embedded in Django-based applications, and data exchange between Dash applications is quite easy [25].

B. *PyViz Panel*

PyViz Panel [26] technology includes a set of widgets for creating user interfaces that can be used in JN, as well as the Bokeh application server. A distinctive feature of this technology is an extensive set of interactive visualization tools functioning in web applications and including Bokeh web graphics library, GeoViews – tools for manipulating geo-graphical data, HoloViews – high-level tools for declarative data visualization.

In addition to creating user interfaces for standalone web apps, Panel allows to develop applications that function in Jupyter notebooks. However, the Panel does not have any significant advantages over the JN in this respect.

C. *Voilà*

This technology [12] is tightly integrated with JN, allowing notebooks to be launched either from the command line or by pressing a button on the JN toolbar. In this case, cells with source code are not displayed.

Unlike traditional web apps, Voilà applications are executed on behalf of the user who runs them, the application is stateful, and a network connection is established for the entire time the application is running. This approach makes scalability difficult, and we should also note the long start-up time of Voilà applications.

All of these disadvantages are compensated by the fact that Voilà allows to run Jupyter notebooks without any modifications. When you start the application, you can specify a layout template in the command line. This in turn makes it possible, for instance, to show manageable slideshows as well as dashboards whose layouts can be customized at runtime. Currently, Voilà applications don't support animations created with the FuncAnimation of matplotlib library, so the animation has to be created manually by erasing the previous frame with IPython.display.clear_output(), forming and displaying the next animation frame.

All of this makes the use of technology attractive in the educational process.

D. *Streamlit*

Streamlit [13] provides an easy and fast way to build web user interfaces for Python apps. The user interface can be displayed both on the web page and on the removable sidebar, saving space on the screen. Streamlit has a rich set of user interface widgets, and there are possibilities of creating your own user interface components [27]. The application is launched from the command line, for example:

```
streamlit run dla04.py
```

Streamlit apps support formatted text using Markdown as well as mathematical formulas using Latex.

Streamlit apps are single-page, stateless, application state has to be saved in the cash. The single-page limitation can be relaxed by reserving space on a web page using placeholders. Start-up time of Streamlit apps is significantly less than Voilà, applications are executed on behalf of the user who launched them. An interesting feature of the technology is built-in means to record and save the screen-cast of the running app.

E. *Comparison of technologies for publishing interactive web applications*

The technologies discussed above can be divided into two groups. The first group includes only Plotly Dash. Dash allows to create traditional web applications that function as a sequence of client requests and server responses. The strength of Dash is the ability to create components on the client side, which in turn allows to reduce the amount of data transferred between the client and the server, as well as to create virtual labs. At the same time, it is the most complex of the technologies discussed in this article.

Common for the second group of technologies is the need to maintain a constant connection between the server and the client, which can be terminated if the user is not active for a long time and this, in turn, requires application restart. The most attractive technology of the second group for educational process is Voilà, because it allows to run Jupyter notebooks as web apps without any additional modifications. Panel does not have any advantages over Voilà, it uses its own set of widgets to

create user interfaces needs a separate application server. The use of Panel in the educational process is reasonable only if geographical data are used intensively.

Streamlit is an intensively developing technology with an extensible set of user interface components for transforming Python modules into interactive web applications. Its ease of use has proven to be very attractive to students, although the technology has not been discussed in details in class.

IV. Course Structure

The main goal of our courses is to gain skills in solving practical problems in students' subject area with the Python ecosystem.

We begin with a crash course in Python programming. We use JN as our primary environment. This allows us to describe the problem using formatted text and formulas, implement code to solve it, conduct computational experiments, and present the results in a visual form in the same notebook.

Visualization is used already in the initial stages of learning by using our own implementation of turtle graphics on top of the matplotlib library to embed drawings directly into the notebook.

Python loops are illustrated by drawing polygons, conditional statements by drawing stars, and recursion by fractals.

To solve a problem, it is necessary to decompose it into subproblems, find ecosystem tools to solve the subproblems, "glue" the results of solving the subproblems by writing program code, and present the results in a user-friendly form. In class, we analyze a large number of examples of solving such problems.

In one of examples the problem of analyzing nonlinear electric circuits is investigated. This requires solving the subtasks of drawing the circuit using SchemDraw library, adding new elements to it, solving nonlinear algebraic equations, and graphical representation of the results.

Most of the Python ecosystem libraries are available in source code; reading and, if necessary, making additions provides extra skills for working with Python program code.

Mandatory components of the course are experimental

Figure 3. Gray Scott model of reaction diffusion

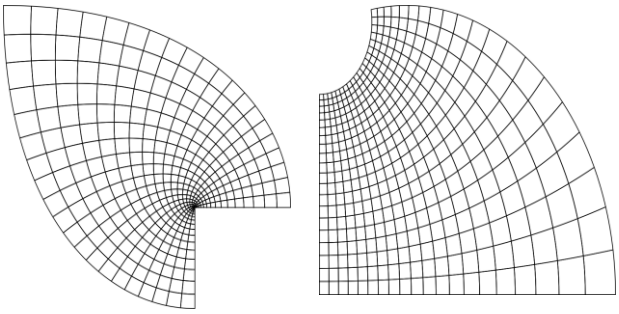

Figure 4. Conformal mappings using functions z^3 and th z

data processing and presentation by means of ecosystem, interpolation and approximation, linear algebra, solving systems of algebraic and ordinary differential equations.

With the help of NumPy the implementation of difference schemes for solving partial derivative equations, including nonlinear ones (Fig. 3), is considered [28].

Optimization problems are solved using both traditional methods and genetic algorithms. In addition, problems related to statistical modeling are considered. We show how to use optimizations to solve inverse problems.

Practical problems from students' subject areas are solved in separate blocks of course.

The solution of almost any problem is accompanied by a scientific visualization, for example, conformal mappings (Fig. 4). As more complex example of visualization are considered Riemann surfaces u+iv=f(x+iy), in this case values on one of coordinate axes in four-dimensional space are displayed by color (Fig.5).

V. Patterns and Fractals in Python

To improve the skills of algorithmic thinking and scientific visualization it is necessary to solve a large number of problems. To increase students' interest, some of these problems were formulated as creating patterns with a given symmetry. We used our own implementation L system [29, 30], which is a kind of Domain Specific Language. This allowed us to integrate patterns into JN, and to formulate the algorithms for drawing patterns as a set of axioms and rules on a set of turtle graphics commands. The drawing of the patterns was organized as a competition (Fig. 6). To draw a pattern, it is enough to set the initial state (axiom) – a sequence of symbols –

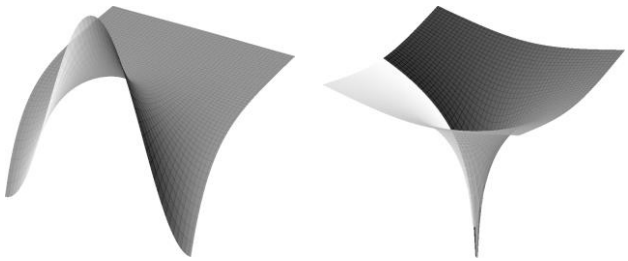

Figure 5. Riemann surfaces for the functions e^{iz} and ln z

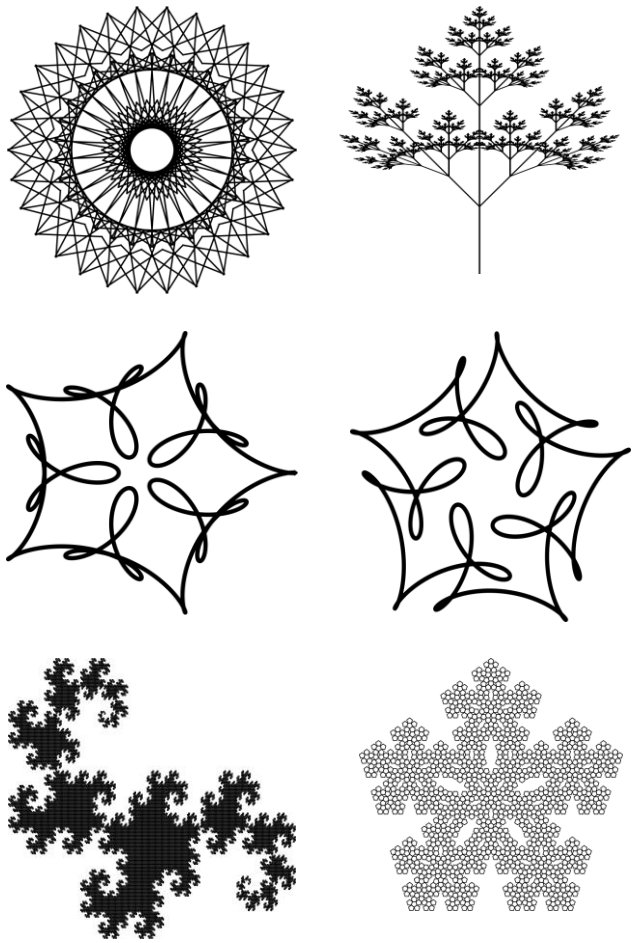

Figure 6. Patterns and fractals

commands to draw the initial part of the pattern and rules to transform the symbols. The initial state for a Koch curve, is F – command to move forward, and the only rule: F→F+F--F+F, where +, − turns to left and right, respectively. The rules can be applied an arbitrary number of times, the result will always be a string of commands to draw. Students wrote pattern generators and then selected the most attractive patterns, allowing the A symbol (Art) to be added to the STEAM acronym. The best patterns were published at the application server.

VI. Integrated Environment for Developing and Publishing Computational Web Apps

To use the considered technologies in the educational process, a simple application server (AS) was developed based on the VirtualBox virtual machine, on which Ubuntu Server was installed, web server Nginx operates with static content and as a reverse proxy. JN and JL, as well as web apps, run using The Littlest JupyterHub (TLJH) [31]. That allows both shared and private user web apps. User authentication is mandatory.

AS supports remote work with JN, JL without the need to install Python ecosystem applications on local computers, but main purpose of AS is to run Dash, Streamlit, Voilà web apps. Panel support was removed due to its lack of use by students.

Administrative web utilities for content and user management were developed. Tools have been added for rapid publication of static sites with the ability to embed interactive web apps into them (Fig. 7). This supports easy publishing of smart tutorials with interactive active content embedded in web pages using iframes.

Students can solve problems and develop web apps by connecting to the AS. Developed web applications can be placed in shared directories and embedded into the static site for all students to share.

In the Covid-19 pandemic AS with published learning materials was used as a unified flipped classroom environment, aimed at problem-solving during classes.

For example, when studying the topic devoted to the solution of ordinary differential equations, students are introduced to the capabilities of the library scipy.integrate, master the composition of simple systems of differential equations, work with the functions of the library before the class begins. In class they solve complex problems, draw phase portraits of systems of ordinary differential equations, build animations of problems solutions, save them as videos.

The application server includes a static site generator that allows to publish smart tutorials and student portfolios by assembling them from heterogeneous content, including html and pdf files, source code, external Internet resources, images, videos, including those published on YouTube, Python applications using Voilà, Streamlit, and Dash technologies. Active content (Python applications running on the AS) is displayed in the content pane in a floating frame. To publish, you need to upload the files to the shared content folder of the application server and prepare a spreadsheet in Microsoft Excel with the table of contents of the static site. Each line of the table of contents specifies the name of the fragment, its type, the unified resource locator, and the fragment level in the hierarchy. The table of contents induces a linear-hierarchical structure of the site, which is formed when compiling the site. During compilation, fragments and unified resource pointers are checked.

Loading a new version of the table of contents triggers the compiler program to form a new version of the site. This approach makes it possible to quickly publish instructional materials before class. As mentioned above, students primarily work with JN because of the ease of publishing Voilà applications. An additional result is that students have mastered the Markdown markup language for text and Latex for formulas needed to publish assignments.

The use of a static site makes it easier to conduct classes, to find educational materials and applications. Interactive Voilà utilities built into the application server simplify the publication of additional materials and resources. In particular, students' portfolios are published with the help of these tools.

VII. Conclusion

We have considered educational technologies and tools to support courses on scientific and technical calculations in Python. Visualizing the results of solving

almost every problem, publishing web applications to conduct computational experiments increases the clarity, allows gamification of classes, and competitions between students. Students found it quite important to see the results of their problem solving in the form of published web applications, it is possible to discuss together.

Python ecosystem which has a uniquely wide set of freeware libraries for solving applied problems puts an influence on the technology of problem solving, when the main thing is to decompose the problem to be solved into a sequence of subproblems, the tools for solving them are available in the libraries. At the same time, it is important to search for and master these tools, "glue" the subproblems into a program for solving it, plan the computational experiments.

The developed integrated environment that supports all stages of the learning process includes remote support for working with JN and JL, running computational web applications based of Voilà, Dash, Streamlit technologies. Static website generator provides preparation of learning materials with minimum labor intensity in the form of smart tutorials with embedded computational web applications. The tools used are so simple that they allow students to publish their assignments as computational web applications.

The considered bundle of technologies allows practically seamless transition from face-to-face classroom technology to distance learning when Cisco Webex or Zoom were used. This approach turned out to be productive in face-to-face training as well for operative consultations without the need to insert them into the schedule of computer classes.

REFERENCES

[1] G. Strimel, M.E. Grubbs, "Positioning Technology and Engineering Education as a Key Force in STEM Education," *Journal of Technology Education*, vol. 27(2), pp. 21–36, 2016.

[2] V. Ochkov, 2^5 Problems for STEM Education. 1st edn, Chapman and Hall/CRC, London, 2020.

[3] H. P. Halvorsen, R. Tretjakova, J. Timmerberg, J. M. Thiriet, S. Mylvaganam, "STEAM for STEM - Include "Art" in STEM (Science, Technology, Engineering and Mathematics)," in *29th Annual Conference of the European Association for Education in Electrical and Information Engineering (EAEEIE)*, pp. 1-7, 2019.

[4] C. Liao, "From Interdisciplinary to Transdisciplinary: An Arts-Integrated Approach to STEAM Education," *Art Education*, vol. 69, no. 6, pp. 44-49, 2016.

[5] Do-Young Lee, Jong-In Chung, "The Effects of Middle School Mathematical Statistics Area and Python Programming STEAM Instruction on Problem Solving Ability and Curriculum Interest," *Journal of the Korea Academia-Industrial cooperation Society*, vol. 20, no. 4 , pp. 336-344, 2019.

[6] Rafeek Mamdouh, Hazem M. El-Bakry, Alaa Riad, Nashaat El-Khamisy, Converting 2D-Medical Image Files "DICOM" into 3D- Models, based on Image Processing, and Analysing their Results with Python Programming, WSEAS Transactions on Computers, ISSN / E-ISSN: 1109-275.

[7] Lucien Ngalamu, A Visual Learning Tool for Effective Student Engagements in Computer Engineering Education: Case of Digital Logic Instruction, WSEAS Transactions on Computers, ISSN / E-ISSN: 1109-2750 / 2224-2872, Volume 19, 2020, Art. #15, pp. 111-114.

[8] Jupyter, available at: https://jupyter.org/, [accessed: 27 December 2020].

[9] D. Toomey, Jupyter Cookbook. Packt, Birmingham-Mumbai, 2018.

[10] T.A Witten, L. M. Sander, "Diffusion-limited aggregation," *Phys. Rev.* B 27(9), 5686–5697, 1983.

[11] L.M. Sander, "Diffusion-limited aggregation: A kinetic critical phenomenon," *Cont. Phys.*, vol. 41(4), pp. 203–218, 2000.

[12] And Voilà!, available at: https://blog.jupyter.org/and-voil%C3%A0-f6a2c08a4a93, [accessed: 27 December 2020].

[13] Awesome Streamlit, available at: https://awesome-streamlit.readthedocs.io, [accessed: 27 December 2020].

[14] M.A.K. Ovais, H. Khusro, Developing Multi-Platform Aps with Visual Studio Code. 1st edn., Packt, Birmingham-Mumbai, 2020.

[15] Jupyter Black [Black formatter for Jupyter Notebook], available at: https://github.com/drillan/jupyter-blackm, [accessed: 27 December 2020].

[16] Rise, available at: https://github.com/damianavila/RISE, [accessed: 27 December 2020].

[17] Mathcad Gateway, available at: https://community.ptc.com/t5/PTC-Mathcad/Mathcad-Gateway-new-calculation-server-from-PTC/td-p/11253?attachment-id=38240, [accessed: 27 December 2020].

[18] MATLAB Web App Server, available at: https://www.mathworks.com/products/matlab-web-app-server.html, accessed: 2020/12/28.

[19] C. Beely, S.R. Sukhdeve, Web Application Development with R using Shiny. 3rd edn, Packt, Birmingham-Mumbai, 2020.

[20] D. Ashley, Foundation Dynamic Web Pages with Python. 1st edn, Apress, New York, 2020.

[21] Dash User Guide, available at: https://dash.plotly.com/, last accessed: 2020/12/28.

[22] A. Tikhonov, "Dash as a platform for development of computational web applications in Python", 1st edn, LAP Lambert Academic Publishing, Saarbrucken, 2018 (in Russian).

[23] A. Bakns, E. Porcello, Learning React. 2nd edn, O'Reilly Media, Sebastopol, USA, 2020.

[24] jupyter-dash, available at: https://github.com/plotly/jupyter-dash, [accessed: 27 December 2020].

[25] django-plotly-dash, available at: https://github.com/GibbsConsulting/django-plotly-dash, accessed: 2020/12/28.

[26] Panel. A high-level app and dashboarding solution for Python, available at: https://panel.pyviz.org/, [accessed: 27 December 2020].

[27] R. Zwitch, "Introducing Streamlit Components", available at: https://dzone.com/articles/introducing-streamlit-components, accessed: 27 December 2020.

[28] Gray Scott Model of Reaction Diffusion, available at: http://groups.csail.mit.edu/mac/projects/amorphous/GrayScott/, accessed: 2021/04/02.

[29] P. Prusinkiewicz, A. Lindenmayer, The Algorithmic Beauty of Plants. Berlin, Springer Verlag, 2004, 240 p.

[30] A. Lindenmayer, "Mathematical models for cellular interaction in development," *J. Theoret. Biology*, vol. 18, pp. 280-315, 1968. Accessed: Oct. 28, 2020. [Online]. Avaiable: http://www0.cs.ucl.ac.uk/staff/p.bentley/teaching/L6_reading/lsystems.

[31] The Littlest JupyterHub (TLJH), available at: https://tljh.jupyter.org/en/latest/, accessed: 2020/12/28.

Creative Commons Attribution License 4.0 (Attribution 4.0 International, CC BY 4.0)

This article is published under the terms of the Creative Commons Attribution License 4.0
https://creativecommons.org/licenses/by/4.0/deed.en_US