Developing Interactive Learning Resources for Python and Machine Learning

Amaris Vélez Candelaria
Master in Computer Engineering
Alfredo Cruz, Ph.D.
Electrical and Computer Engineering and Computer Science Department
Polytechnic University of Puerto Rico

Abstract *– This paper describes the process of developing learning resources for students in computing and offers various examples of the learning resources that were successfully developed as a result. Two main learning resources were developed from the project described in this paper: A supplementary technical guide and interactive workshops. The technical guide was found to be beneficial for different types of students such as students who are interested in the field of computing but, have not yet started their formal education on the field, students in computing who wish to review concepts they were previously taught, and students who wish to mentor other students in programming concepts. On the other hand, the interactive workshops written as Jupyter notebooks were found to be beneficial for community outreach activities since they contain both examples and exercises that presenters may easily discuss while the audience is interacting with code relating to each example and exercise.*

Key terms *– Jupyter Lab, Machine Learning, Pythontechnical guide, and Workshops*

INTRODUCTION

Proper technical preparation will allow students to be able to secure the opportunities that are available to them in the field of computing. There are many ways in which students could improve their technical knowledge in this field such as: taking online and in-class courses, watching educational videos, reading articles and textbooks, and participating in workshops and competitions. However, for each topic in this field of study, the student must be able to review a vast amount of resources and filter them to ensure that only those which are most relevant are used. This filtering task can be time consuming and, at times, some students may become overwhelmed by the subject that they wish to study.

To this end, this paper describes the details from a project that aims to develop interactive learning resources for students in the field of computing to facilitate the student's task of understanding and practicing specific topics in computing. The interactive learning resources were devised by using Jupyter Lab; a next-generation web-based user interface for Project Jupyter. Jupyter Lab allows users to work with kernel-backed documents that enables code in any text file (Markdown, Python, R, C++, etc.) to be run interactively in any Jupyter kernel. These kernel-backed documents are known as Jupyter notebooks and for the project, they served as workshop content with which students could read documentation on a topic and practice the concept by interactively running code in the same document. From another point of view, before devising the interactive content, a supplementary technical guide was devised to provide additional learning resources to students using the interactive workshops. The supplementary technical guide discussed the following topics: compiler theory and practical programming, algorithms and data structures, and data science for machine learning.

JUSTIFICATION

The Digital Revolution and continuing technological advances have affected the job outlook for almost every type of position available across the world has been observed. [1] found that in the US, a positive job outlook has emerged for computer scientists and engineers and the need for computer scientists can be found in nearly every major industry in the United States (US). According to the Bureau of Labor Statistics, jobs in computing and information technology are expected to grow nearly as twice as fast as the average for all careers, adding

more than half a million new jobs from 2016 to 2026 [2]. As a result, the development of these interactive learning resources for students in computing would be highly beneficial due to the current and future demand for computer engineers and scientists.

From another point of view, the topics that were selected to be discussed through the devised learning resources were practical programming and machine learning. These specific topics were selected due to their current importance in the job market for computer scientists and engineers. The topic of programming has always been essential for the computer scientists and engineers. Programming provides the necessary tools to solve any problem that may be computationally solved. On the other hand, the topic of machine learning, which is a sub-topic of artificial intelligence (AI), is important due to how the topic of AI has been projected to transform the global economy [3]. Furthermore, computer scientists and engineers have the responsibility of predicting and adapting to the changes in the industry due to the increasing demand for machine learning.

DESIGN PROCESS

The design process of the project may be divided in three phases: environment setup, developing supplementary technical guide, and developing interactive workshops. Each phase is individually described below.

Environment Setup

To execute the different scripts offered throughout the supplementary technical guide and the developed interactive learning material for Python and Machine Learning, a Jupyter Lab environment had to be set-up. This section will specify the necessary steps to replicate such environment. Note that Jupyter Lab is a next-generation web-based user interface for Project Jupyter. It allows users to work with documents kernel-backed documents which enable code in any text file (Markdown, Python, R, C++, etc.) to be run interactively in any Jupyter kernel.

This section focuses on describing the steps to be executed to be able to program in C++ and Python using Jupyter Lab. Note that the Jupyter kernel for C++ is not supported by the Windows operating system. For this reason, aside from showing how to download and install Jupyter Lab with the necessary kernels, this guide will show how to create a virtual machine running on the latest version of the Ubuntu operating system which supports the Jupyter kernel for C++. The steps to setup the environment are shown below:

1. Downloading and installing VirtualBox: Go to link https://www.virtualbox.org/ and download the latest version of Virtual Box. Once the download has been completed, proceed to installing virtual box in your computer. Note that VirtualBox is a free and open source virtualization software from Oracle that allows for the installation of other operating systems in virtual machines. On the other hand, the computer system must have the virtualization feature enabled for VirtualBox to work. To ensure that virtualization is enabled, reboot the system and as soon as it powers up press F2, F10, or F12 to access the BIOS settings. Look for the virtualization option in the BIOS settings and ensure that it is enabled.
2. Creating a virtual machine running on the latest version of Ubuntu: Go to the link https://ubuntu.com/desktop and download the latest version of Ubuntu. Once the download has been completed, proceed to installing Ubuntu using VirtualBox by executing the following steps:
    - Start VirtualBox and click on the New symbol. Pop-up will appear to specify name, type (Linux), and version (Ubuntu 64-bit). After specifying name, type, and version press next.
    - Specify the memory size (RAM) to be allocated to the virtual machine. Allocate at minimum 2GB of RAM. After specifying memory size, press next.

- Create a virtual hard disk and specify the hard disk file type as VDI.
- Choose either the “Dynamically allocated” or the “Fixed size” option for creating the virtual hard disk and press next.
- Specify the file location and the virtual hard disk size. Specify a size that is equal or greater (if possible) than 10GB. After specifying the virtual hard disk size, press create.
- Boot and install the downloaded Linux ISO by clicking on the created virtual machine and selecting the downloaded ISO. Note that if the ISO does not appear in the pop-up, it may be found by clinking on the folder icon and searching for the ISO in the file explorer.
- Ubuntu will start to boot and the option to install Ubuntu should appear. Select the install Ubuntu option and then, press continue.
- Select Erase disk and install Ubuntu and then, press install.
- Specify time zone, keyboard layout, and password.
- Once installation finishes, restart virtual system by pressing Restart now.

3. Downloading and installing Anaconda: Open the web browser and enter the following link: https://www.anaconda.com/distribution/ and copy the installer bash script for Linux. Then, execute the following commands on the Linux terminal:
   - Move into the /tmp directory: $ cd /tmp
   - Use curl to download copied link from the Anaconda website: $ curl -0 https://repo.anaconda.com/archive/Anaconda3-2020.02-Linux-x86_64.sh
   - Run the Anaconda script: $ bash Anaconda3-2020.02-Linux-x86_64.sh
   - Review Anaconda’s license agreement and press ENTER until the end is reached and type yes if you agree with the license
   - Press ENTER to accept the default location or specify a different location
   - Once the installation is complete, select the option to use the conda command: Type yes
   - Activate the installation: $ source ~/.bashrc
4. Downloading and installing cling. -To download and install cling, execute the following commands on the Linux terminal:
   - Install the C++ interpreter cling: (base) $ conda install -c conda-forge cling
   - Create new environment named cling: (base) $ conda create -n cling
   - Activate cling environment: (base) $ conda activate cling
5. Installing with Jupyter lab using Anaconda. -To download and install cling, execute the following commands on the Linux terminal:
   - Install Jupyter Lab in the cling environment: (cling) $ conda install -c conda-forge jupyterlab
   - Open Jupyter Lab (Python kernel should now be available) and click on the outputted url to open the web-based interface: (cling) $ jupyter lab
6. Installing xeus-cling using cling - To download and install xeus-cling, execute the following commands on the Linux terminal:
   - Install the Jupyter kernel for C++ in the cling environment: (cling) $ conda install xeus-cling conda-forge
   - Open Jupyter Lab (Python and C++ kernels should now be available) and click on the outputted url to open the web-based interface: (cling) $ condajupyter lab

Developing Supplementary Technical Guide

Before proceeding to develop learning resources for students in the field of computing, the term computing had to be defined. The field of computing involves the study of computer technology and how it is used to manage, process, and communicate information to accomplish a given task. It encompasses both the design and development of the hardware and software necessary

to the computing system [4]. This is a vast field of study and the development of a complete technical guide would be a complex task to fulfill. For this reason, only three sub-fields in computing were chosen for the creation of the guide. As previously mentioned, these topics were selected according to their current importance in the job market for computer scientists and engineers. The selected sub-fields to be discussed in the technical guide are the following:

- Programming languages: In theory and in practice
  - Historical background
  - Compiler theory
  - Practical programming
- Algorithms and data structures
  - Greedy algorithm
  - Divide and conquer
  - Dynamic programming
  - Searching & Sorting techniques
  - Linked lists
  - Stacks & Queues
  - Graphs
  - Trees
- Data science for machine learning
  - Introduction to machine learning
  - The five tribes of machine learning

Developing interactive Jupyter Lab notebooks

Once the supplementary technical guide was devised, two workshops were created. One workshop covered the topic of programming basics with Python and another covered the topic of supervised machine learning with Python. Each workshop was composed of four main sections and for each section there were a series of examples and exercises. Additionally, these workshops have been made available in a GitHub repository.

RESULTS

The project of developing learning resources for students in computing produced two main results: the supplementary technical guide and the interactive workshops. Each result is briefly presented in this section.

Supplementary Technical Guide

As previously mentioned, the technical guide summarized concepts from three topics in computing: compiler theory and practical programming, algorithms and data structures, and machine learning. An example of the content that was presented for each of the guide's topics is shown below.

The first topic that was introduced in the technical guide was related to compiler theory and practical programming. The 7 phases of the compiler according to [5] were individually discussed. The compiler phases are shown in Figure 1.

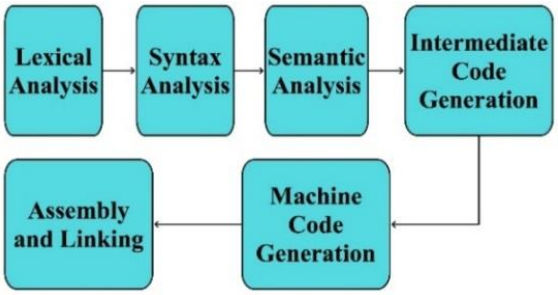


Figure 1
Phases of the Compiler

Then, after offering students a small theoretical background on compilers, basic programming concepts were taught using C++ and Python. These concepts included: the three phases to develop a program, inputs and outputs, variables, operators, control flow, arrays, pointers, references, functions, classes, and objects. In the guide, C++ and Python were taught in parallel by offering examples of written code in both languages for each of the discussed programming concepts.

Concerning the Concept of Program Inputs, the following Content was Presented

Whenever a program requires user input, an input statement must be declared. The syntax for an input statement varies according to the programming language. In C++, cin is used to receive inputs. In Python, the input( ) function is used to receive inputs. The behavior of the input statement when the

program reaches its execution at runtime is generally as follows:

- Program is halted and waits for input data.
- After user enters the data by pressing ENTER, program reads value and stores it in a specified variable.

Figure 2 and Figure 3 offer an example of an input statement in C++ and Python, respectively. Additionally, the figures show how a program remains halted when expecting inputs.

```
#include <iostream>
using namespace std;

int age = 0;

cout << "Enter your age: ";
cin >> age;
cout << endl << "Age received!";
```

```
Enter your age:
```

Figure 2
Waiting for user Input in C++ Program

```
age = input("Enter your age: ")
print("\nAge received!")
```

```
Enter your age:
```

Figure 3
Waiting for user Input in Python Program

Concerning the Programming Concept related to the Three Phases to Develop a Program, the following Content was presented

The three phases to develop a program are the following: analyze problem, write program, and validate program. The phase of analyzing a problem refers to the act of clearly defining the problem and identifying the input and output data that will be needed to solve the problem. The phase of writing the problem refers to the act of taking into consideration the data to be expected as input and output, identifying the program variables, and implementing the operations to be executed on the data using a specific programming language syntax. Finally, the phase of validating the program refers to the act of ensuring that the program correctly solves the defined problem by validating each line of code and comparing the output data with the expected result.

To provide an example of these three phases consider the case in which the area of a circle had to be calculated. Each phase would be executed as follows:

- Phase #1: Problem analysis
  - Expected inputs and outputs: To calculate the area of a circle (output), the circle's radius (input) must be known.
  - Operations: Once the radius of the circle has been obtained, its area may be calculated as the square of the circle's radiusmultiplied by the constant pi.
- Phase #2: Write program

Taking into consideration the previous analysis, Figure 4 and Figure 5 show the written program in Python and C++, respectively.

Phase #3: Validate program

If the area of the circle is manually calculated for the same radius as in previously executed pieces of code, the following result is obtained:

$$Area_{cir} = r^2\pi = 4.5^2\pi = 63.6173 \qquad (1)$$

As it may be observed, the calculated result is equal to the results obtained by running the written scripts. Thus, the program has been validated.

```
from math import pi

#Requesting circle radius
r = float(input("Enter the"
" circle's radius in"" ft: \n"))

#Calculating and
#outputting circle's area
area = pi*r**2
print("\nArea of the circle"
" in square ft: \n", area)
```

```
Enter the circle's radius in ft:
 4.5

Area of the circle in square ft:
 63.61725123519331
```

Figure 4
Finding the Area of a Circle in Python

```cpp
#include <iostream>
#include <math.h>
using namespace std;

//Declaring variables
//according to expected
//inputs and outputs
float r = 0.0;
float area = 0.0;

//Requesting circle radius
std::cout << "Enter the "
    << "circle's radius in ft: ";
std::cin >> r;

//Calculating and
//outputting circle's area
area = pow(r, 2)*M_PI;
std::cout << endl
<< "Area of the circle in "
<< "square ft: \n" << area;
```

```
Enter the circle's radius in ft:
 4.5

Area of the circle in square ft:
63.6173
```

Figure 5
Finding the Area of a Circle in C++

Concerning the Concept of Classes and Objects, a Example of the Written Content is presented Below

Everything in C++ and Python is associated with classes and objects. This is due to the fact that C++ and Python are both object-oriented programming (OOP) languages. The object-oriented programming approach solve problems by creating objects using classes. Objects can be viewed as a collection of data and functions that act on that data. As a result, objects can be defined through the following two characteristics: attributes and behavior. On the other hand, a class can be viewed as the object's blueprint. In more detail, classes are user defined datatypes that specify the data members (attributes) and member functions (behavior) for an object.

To provide an example for classes and objects in C++, consider a dog as object in a program. The dog class may be defined as shown in Figure 6. Then, the dog object may be declared and its member functions may be implemented as shown in Figure 7.

On the other side, in Python, the dog class may be defined as shown in Figure 8. Then, the dog object may be declared and its member functions may be implemented as shown in Figure 9.

```cpp
#include <iostream>
using namespace std;

class dog
{
    private:

    //Data members
    std::string breed;

    public:
    //Member functions

    //Default constructor
    dog()
    {
        breed = "Dalmation";
    }

    //Parametrized constructor
    dog(std::string s_breed)
    {
        breed = s_breed;
    }

    //Functions to assign
    //values to object attributes
    void setBreed(std::string s_breed)
    {
        breed = s_breed;
    }

    //Functions to get values
    //from object attributes
    std::string getBreed()
    {
        return breed;
    }

    //Functions to print values
    //of object attributes
    void printBreed()
    {
        std::cout<<breed<<'\n';
    }

    //Functions to imitate dog
    //behavior
    void bark()
    {
        std::cout<<"Woof!\n";
    }

    void sleep()
    {
        std::cout<<"Zzzzzz\n";
    }

    void eat()
    {
        std::cout<<"Munch Munch\n";
    }
};
```

Figure 6
Defining Dog Class in C++

```cpp
#include <iostream>
using namespace std;

//Using default constructor
//to declare object
dog Spike;

//Using parametrized
//contructor to declare object
dog Marley("Labrador");

//Printing dog breeds
std::cout<<"Spike's breed is: ";
Spike.printBreed();
std::cout<<"Marley's breed is: ";
Marley.printBreed();

//Calling dog functions
Spike.bark();
Marley.sleep();
Spike.eat();
```

```
Spike's breed is: Dalmation
Marley's breed is: Labrador
Woof!
Zzzzzz
Munch Munch
```

Figure 7

Declaring and Implementing Dog Object in C++

```python
class dog:

    #Constructor
    def __init__(self, sbreed="Dalmation"):
        #Data members
        self.breed = sbreed

    #Member functions
    #Functions to assign values to
    #object attributes
    def setBreed(self, sbreed):
        self.breed = sbreed

    #Functions to imitate dog behavior
    def bark(self):
        print("Woof!")

    def sleep(self):
        print("Zzzzzz")

    def eat(self):
        print("Munch munch")
```

Figure 8

Defining Dog Class in Python

```python
#Using default constructor
#to declare object
Spike = dog()

#Using parametrized contructor
#to declare object
Marley = dog("Labrador")

#Printing dog breeds
print("Spike's breed is: ",
      Spike.breed)
print("Marley's breed is: ",
      Marley.breed)

#Calling dog functions
Spike.bark()
Marley.sleep()
Spike.eat()
```

```
Spike's breed is:  Dalmation
Marley's breed is:  Labrador
Woof!
Zzzzzz
Munch munch
```

Figure 9

Declaring and Implementing the Dog Class in Python

Concerning the Concept of Control Flow, an Example of the Written Content is presented Below

The if… else statement is used whenever there are two possible execution paths in which the program could proceed. Then, the execution path to be taken depends on the result from an evaluated condition. If the result is true then, the code in the body of the if will be executed. If the result is false then, the code in the body of the else is executed. The flow diagram for this statement is shown in Figure 10.

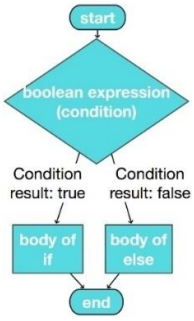


Figure 10

Flow diagram for the if… else Statement

Both C++ and Python incorporate the if… else statement. However, there is a slight variation in syntax between the two languages. Figure 11 and Figure 12 show this variation by offering an example implementation of the while statement in C++ and in Python, respectively.

```cpp
#include <iostream>
using namespace std;

int grade = 80;

if(grade > 89)
std::cout <<
"You got an A!";
else
    if(grade > 79)
    std::cout <<
    "You got an B!";
    else
        if(grade > 69)
        std::cout <<
        "You got an C!";
        else
            if(grade > 59)
            std::cout <<
            "You got an D!";
            else
                std::cout <<
                "You got an F!";
```

```
You got an B!
```

Figure 11

Implementation of the if… else Statement in C++

```
grade = 80
if grade > 89:
    print("You "
    "got an A!")
else:
    if grade > 79:
        print("You "
        "got an B!")
    else:
        if grade > 69:
            print("You "
            "got an C!")
        else:
            if grade > 59:
                print("You "
                "got an D!")
            else:
                print("You "
                "got an F!")
```

```
You got an B!
```

Figure 12
Implementation of the if… else Statement in Python

The second topic that was introduced in the guide was related to algorithms and data structures.

Concerning the Topic of Algorithms, an Example of the Written Content is presented Below

An algorithm may be defined as a sequence of precise and unambiguous instructions telling a computer what to do. It's important to note that not any sequence of instructions may be defined as an algorithm. According to [6], to be able to define a sequence of instructions as an algorithm, the sequence must have the following characteristics:

- Clear and unambiguous: The sequence should be clear and unambiguous. The repeated execution of a sequence using the same starting parameters should always lead to the same result.
- Input clarity: If a sequence requires any inputs, these inputs must be clearly defined.
- Output clarity: The sequence must clearly define the outputs to be obtained. Also, the amount of outputs should be greater or equal to one.
- Finite-ness: The sequence should end after a finite number of steps.
- Feasible: The execution of the sequence should not depend on future technology and should by achievable with the currently available resources.
- Language independent: The instructions in the sequence should be independent of any programming language.

Computers are made of billions of switches. The simplest algorithm would instruct a computer to flip a switch (1 or 0). However, this section will focus on more complex algorithms, particularly those involved when implementing data structures. These algorithms may be divided into the following categories according to their main functionality in the data structure:

- Search algorithm: Used to search for an item in the data structure
- Sort algorithm: Used to sort items in the data structure
- Insert algorithm: Used to insert an item into the data structure
- Update algorithm: Used to update an item already inside the data structure
- Delete algorithm: Used to delete an item already inside the data structure

Concerning the Topic of Data Structures, an Example of the Written Content is presented Below

A data structure can be defined as a programmatic way of collecting, organizing, and managing data. Any structure capable of storing data may be referred to as a data structure. The main goal of the data structure is to increase efficiency and reduce complexity whenever operations are performed on data. There are different types of data structures and these may be classified based on the following characteristics: linear, non-linear, homogenous, non-homogenous, static, or dynamic. Linear data structures, such as arrays, are characterized by using a linear sequence to arrange data items. Non-linear data structures, such as trees and graphs, are characterized by using a non-linear sequence to arrange data items. Homogenous data structures, such as arrays, are characterized by containing only elements of the same data type. Non-

homogenous data structures, such as lists, are characterized by containing elements of any data type. Static data structures, such as arrays, are characterized by having a fixed size at compile time. Finally, dynamic data structures, such as linked lists created using pointers, are characterized by having a size that may increase or decrease depending on the program's needs during its execution.Furthermore, there are two groups in which data structures may be divided which are primitive data structures and non-primitive as shown in Figure 13. Primitive data structures are those which directly operate upon machine instructions. On the other hand, non-primitive data structures are more complex data structures that are derived from primitive data structures. The following sections in the guide focused on discussing the theory behind non-primitive data structures.

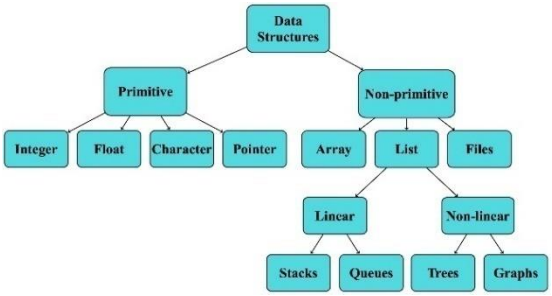


Figure 13
Types of Data Structures

For each data structure a diagram was created to describe it. Figure 14 offers an example of the created diagrams by showing the diagram created to describe the tree data structure.

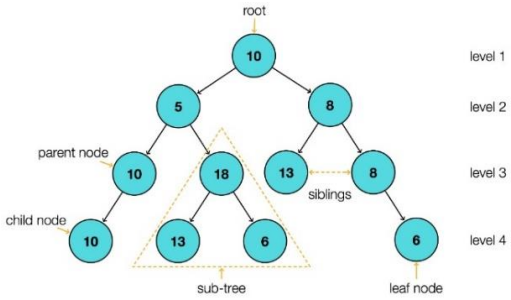


Figure 14
Tree data structure

The third and final topic that was introduced in the guide was related to the five tribes of machine learning which provides a small theoretical background on the topic.

Learning algorithms aim to learn an algorithm that solves any problem given the appropriate data relating to the problem. Thus, any learning algorithm could approximate any function arbitrarily closely given enough data. However, enough data could be infinite. To counteract this problem, learning algorithms have to make assumptions and these assumptions vary from one learning algorithm to another. In the best case scenario, the learning algorithm's assumptions allow it to function as a "master algorithm"; which in [7] Domingos defines as an algorithm capable of solving any problem given a finite amount of data relating to the problem.

In [7] Domingos established that machine learning experts have devised five schools of thought in their journey to find the master algorithm, although no such algorithm still exists. The five schools of thought are the following: symbolists, connectionists, evolutionaries, bayesians, and analogizers. Each of these schools of thought and their respective "master algorithm" were individually discussed in the guide. Table 1 summarizes the key factors to consider for each of the schools of thought that were introduced in the guide.

Table 1
The Five Tribes of Machine Learning

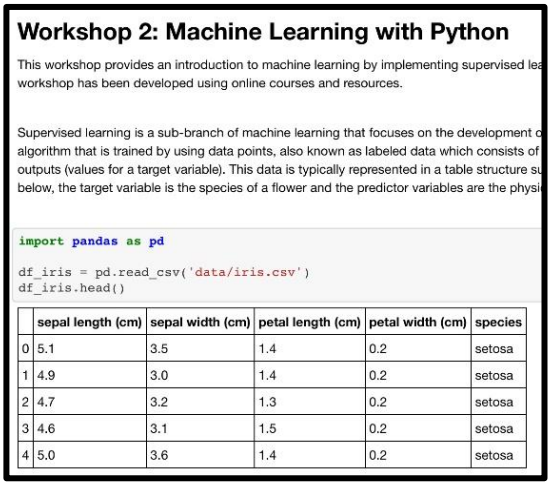
Tribe	Problem	Solution
Symbolists	Knowledge composition	Inverse deduction
Connectionists	Credit assignment	Backpropagation
Evolutionaries	Structure discovery	Genetic programming
Bayesians	Uncertainty	Probabilistic inference
Analogizers	Similarity	Kernel machine

Note that due to its length, the complete guide cannot be described in this paper. Only examples of the written content was presented in this paper. However, the complete version of the guide may be

reviewed and downloaded from the following GitHub repository: https://github.com/avelez-cloud/Interactive-Learning-Resources-for-Python-and-ML.

Interactive Workshops

Apart from the supplementary technical guide, another of the results obtained from the project were the developed workshops which allow students to interactively learn and practice concepts in computing. As previously mentioned, the project focused in developing interactive learning resources for two specific topics in computing: Python programming (workshop #1) and Machine Learning with Python (workshop #2). The starting sections of one of the workshops is shown in Figure 15.



Workshop 2: Machine Learning with Python

This workshop provides an introduction to machine learning by implementing supervised lea
workshop has been developed using online courses and resources.

Supervised learning is a sub-branch of machine learning that focuses on the development o
algorithm that is trained by using data points, also known as labeled data which consists of
outputs (values for a target variable). This data is typically represented in a table structure su
below, the target variable is the species of a flower and the predictor variables are the physi

```
import pandas as pd

df_iris = pd.read_csv('data/iris.csv')
df_iris.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

Figure 15
Starting Section of the Machine LearningWorkshop

Note that due to their extensive length, the complete workshop has not been included in this paper. However, the complete version of the workshops may be reviewed in the following GitHub repository: https://github.com/avelez-cloud/Interactive-Learning-Resources-for-Python-and-ML.

DISCUSSION OF RESULTS

Through the project two main learning resources were developed which were the supplementary technical guide and the interactive Jupyter notebook workshops. The supplementary technical guide was included in the results section of the report. This guide allows students to read and learn about the following concepts in computing: compiler theory, programming in C++ and Python and the comparison between them, algorithms and data structure definitions, and the five tribes of machine learning which provides a general idea of the different types of algorithms that are being implemented in machine learning.

As a result, this guide may be used by different types of students such as students who are interested in the field of computing but, have not yet started their formal education on the field, students in computing who wish to review concepts they were previously taught, and students who wish to mentor other students in programming concepts may use the guide as a reference.

From another point of view, the developed workshops are expected to be highly beneficial for community outreach activities since they contain both examples and exercises that presenters may easily discuss while the audience is interacting with code relating to each example and exercise. In the results section, only a partial demonstration of the workshops were given due to their length. The complete pdf version of the interactive Jupyter notebook workshops may be review in the previously specified GitHub repository.

FUTURE WORK

Developing a technical guide and workshops for topics in computing proved to be a lengthy task due to the large amount of information that could be discussed with simply one sub-topic. For future work, the development of technical guides and workshops for more topics would be recommended to create a more complete guide to the field of computing. The inclusion of the following topics would make for an interesting guide: Operating Systems, Databases, Computer Architecture, Human Computer Interaction, Robotics, Networks, and Theory of computation.

CONCLUSIONS

The main goal of the project was successfully achieved by developing learning resources that will surely benefit students with the desire to learn. The developed guide and interactive workshops offer students a head start into computer engineering and computer science by teaching them how to use one of the most important tools for the computer scientist and engineer; programming. Additionally, the popular topic of machine learning is discussed to motivate students to learn about topics that are projected to transform our global economy.

REFERENCES

[1] Kowarski, I. (2019). *What Can You Do With a Computer Science Degree?*. US News & World Report. [Online] Retrieved from https://www.usnews.com/education/best-graduate-schools/articles/2019-05-02/what-can-you-do-with-a-computer-science-degree.

[2] Colino, S. (2018). *8 College Majors With Great Job Prospects*. US News & World Report. [Online] Retrieved from https://www.usnews.com/education/best-colleges/articles/2018-09-11/8-college-majors-with-great-job-prospects.

[3] Johansson, A. *How AI Is Upending Computing IEEE Computer Society*. Computer.org. [Online] Retrieved from https://www.computer.org/publications/tech-news/trends/how-ai-and-machine-learning-are-affecting-the-computer-science-industry.

[4] *What is Computing? - Definition from Techopedia. Techopedia.com.* (2019). [Online] Retrieved from https://www.techopedia.com/definition/6597/computing.

[5] Mogensen, T. (2010). "Basics of compiler design" (pp. 2-3). [Verlag nichtermittelbar].

[6] Bhatt, S. (2019). *Characteristics of an Algorithm. Medium*. [Online] Retrieved from https://medium.com/@bhattshlok12/characteristics-of-an-algorithm-49cf4d7bcd9.

[7] Domingos, P. "The Master Algorithm". *Basic Books*. (2015).