Created by ardyseto@uny.ac.id
2023

Lab Sheet 1:

Introduction to Control Systems and Python

Learning Objectives:

1. Set up the Python environment for control system laboratory works.
2. Basic Python programming

Task 1: Setting up the Python Environment for Control System Laboratory Works

Python is a general-purpose programming language that is used for a wide variety of tasks, including data science, machine learning, and web development. It is a popular language because it is easy to learn, powerful, and versatile.

Jupyter Notebook is a web-based interactive environment that allows you to run Python code, write text, and create visualizations. It is a great tool for prototyping, exploring data, and sharing your work with others. The step of setting up the environment:

1. Download and install Miniconda
   Go to the Miniconda website: https://docs.conda.io/en/latest/miniconda.html and download the installer for your operating system.

   Latest - Conda 23.5.2 Python 3.11.3 released July 13, 2023

Platform	Name	SHA256 hash
Windows	Miniconda3 Windows 64-bit	00e8370542836862d4c790aa8966f1d7344a8addd4b766004febcb23f40e2914
	Miniconda3 Windows 32-bit	4fb64e6c9c28b88beab16994bfba4829110ea3145baa60bda5344174ab65d462
macOS	Miniconda3 macOS Intel x86 64-bit bash	1622e7a0fa60a7d3d892c2d8153b54cd6ffe3e6b979d931320ba56bd52581d4b
	Miniconda3 macOS Intel x86 64-bit pkg	2236a243b6cbe6f16ec324ecc9e631102494c031d41791b44612bbb6a7a1a6b4
	Miniconda3 macOS Apple M1 64-bit bash	c8f436dbde130f171d39dd7b4fca669c223f130ba7789b83959adc1611a35644
	Miniconda3 macOS Apple M1 64-bit pkg	837371f3b6e8ae2b65bdfc8370e6be812b564ff9f40bcd4eb0b22f84bf9b4fe5
Linux	Miniconda3 Linux 64-bit	634d76df5e489c44ade4085552b97bebc786d49245ed1a830022b0b406de5817
	Miniconda3 Linux-aarch64 64-bit	3962738cfac270ae4ff30da0e382aecf6b3305a12064b196457747b157749a7a
	Miniconda3 Linux-ppc64le 64-bit	92237cb2a443dd15005ec004f2f744b14de02cd5513a00983c2f191eb43d1b29
	Miniconda3 Linux-s390x 64-bit	221a4cd7f0a9275c3263efa07fa37385746de884f4306bb5d1fe5733ca770550

   Run the installer and follow the on-screen instructions.
   - Windows: https://youtu.be/r7XPcSNmWu4
   - Linux : https://youtu.be/SJBLJTGdj-g
   - Mac OS : https://youtu.be/4qS7qw9v884
2. Create a new Python environment
   Open a terminal window and type the following command to create a new Python environment named control:

```
Unset
conda create -n control
```

Activate the new environment:

```
Unset
conda activate control
```

3. Install the python control and jupyterlab packages
   In the activated environment, type the following command to install the python-control packages:

```
Unset
conda install -c conda-forge control slycot
```

In the activated environment, type the following command to install the jupyter notebook packages:

```
Unset
pip install jupyterlab
```

4. Start JupyterLab
   Type the following command to start JupyterLab:

```
Unset
jupyter-lab
```

This will open a web browser window with a JupyterLab.

That's it! You have now successfully installed miniconda and set up a Python environment for control systems, including Jupyter Notebook.

Additional resources:

- Miniconda documentation: https://docs.conda.io/en/latest/
- python-control documentation: https://python-control.readthedocs.io/en/latest/
- matplotlib documentation: https://matplotlib.org/
- Jupyter Notebook documentation: https://jupyter.org/

Task 2: Basic Jupyter Notebook

Here are some basic Jupyter Notebook concepts that you should know:

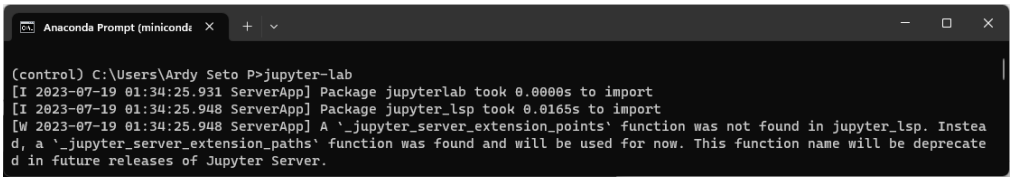
- Cells: Cells are the basic building blocks of a Jupyter Notebook. You can write code, text, or visualizations in a cell.
- Markdown: Markdown is a lightweight markup language that is used to format text in Jupyter Notebooks.
- Code cells: Code cells are used to run Python code. You can run a code cell by clicking on it and pressing Shift+Enter.

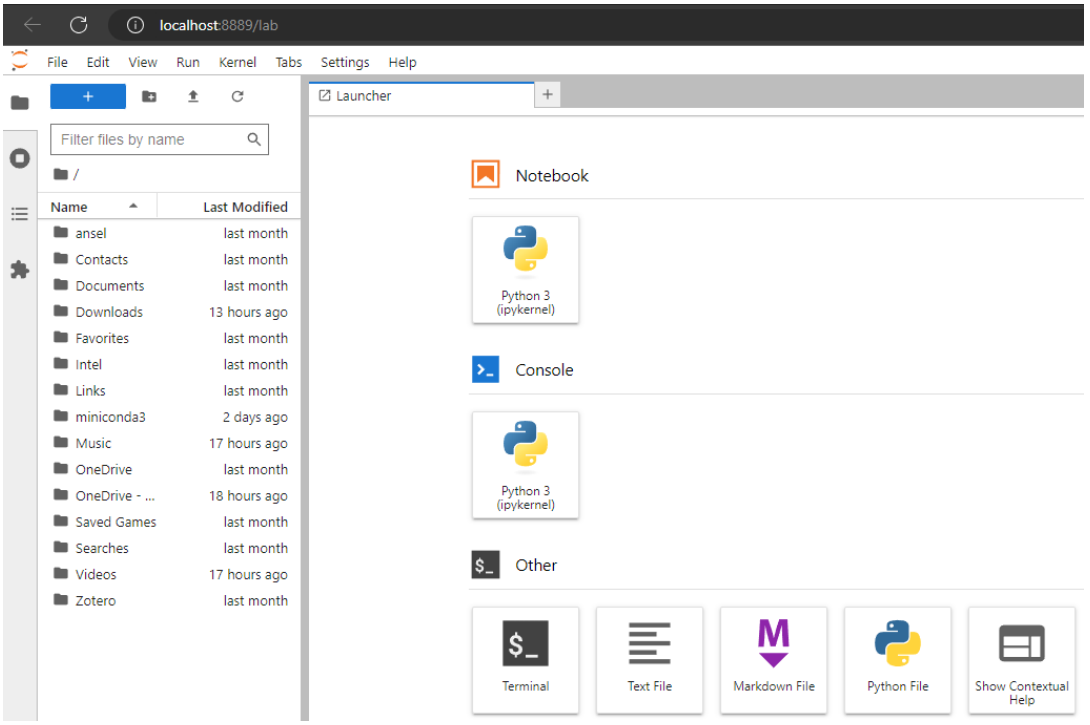
- Output: The output of a code cell is displayed below the cell.
- Visualizations: You can create visualizations in Jupyter Notebooks using the matplotlib library.

1. Starting the Jupyter Notebook Server

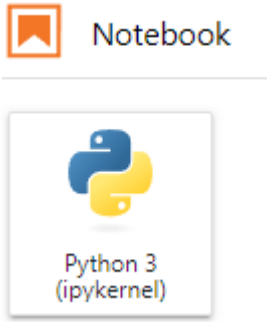
```
Unset
jupyter-lab
```

Keep the command prompt open

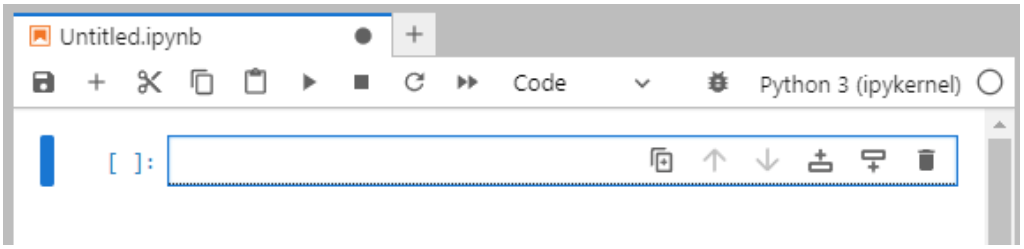




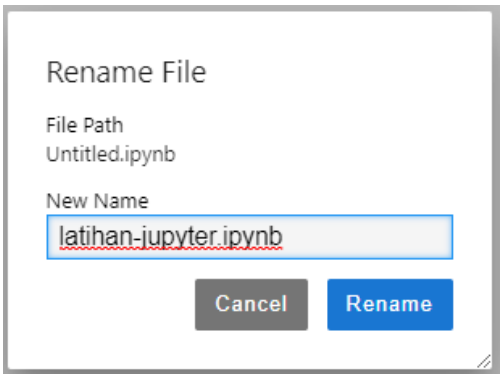
2. Creating a Notebook with click on the Python 3 logo (ipykernel)



View of jupyter notebook worksheet



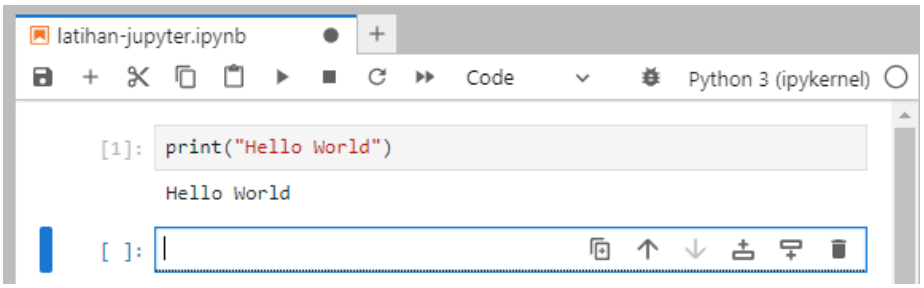
3. Rename notebook because default name Untitled that isn't a very descriptive name



4. Running Cells

   A Notebook's cell defaults to using code whenever you first create one, and that cell uses the kernel that you chose when you started your Notebook. In this case, you started yours with Python 3 as your kernel, so that means you can write Python code in your code cells. Since your initial Notebook has only one empty cell in it, the Notebook can't really do anything. Thus, to verify that everything is working as it should, you can add some Python code to the cell and try running its contents.

   Running a cell means that you will execute the cell's contents. To execute a cell, you can just select the cell and click the Run button that is in the row of buttons along the top. It's towards the middle. If you prefer using your keyboard, you can just press SHIFT + ENTER.



Task 3: Basic Python Programming Language

Here are some basic Python concepts that you should know:

- Variables: Variables are used to store data in Python. You can declare a variable by using the var_name = value syntax.
- Data types: Python has a variety of data types, including integers, floats, strings, lists, and dictionaries.

- Operators: Operators are used to perform mathematical operations on data. Some common operators include +, -, *, /, and %.
- Control flow: Control flow statements allow you to control the order in which your code executes. Some common control flow statements include if, else, for, and while.

1. Variable

```python
# Variables
my_variable = 10
print(my_variable)
```

2. Data types

```python
# Data types
integer = 10
float = 10.0
string = "Hello, world!"
list = [1, 2, 3]
dictionary = {"key1": "value1", "key2": "value2"}

print(integer, type(integer))
print(float, type(float))
print(string, type(string))
print(list, type(list))
print(dictionary, type(dictionary))
```

3. Operators

```python
# Operators
print(10 + 20)
print(10 - 20)
print(10 * 20)
print(10 / 20)
print(10 % 20)
```

4. Control flow

```python
# Control flow
if 10 > 20:
    print("10 is greater than 20")
else:
    print("10 is not greater than 20")

for i in range(10):
    print(i)

while i < 10:
    print(i)
    i += 1
```

5. Function

```python
# Function
def greet(name):
    print(f"Hello, {name}!")

greet("Alice")
```

```python
def multiply_numbers(num1, num2=1):
    return num1 * num2

result = multiply_numbers(5)
print(result)
```

Additional Resource:

- https://www.pythontutorial.net/python-basics/
- https://www.programiz.com/python-programming
- https://www.tutorialspoint.com/python/python_basic_syntax.htm

---

Task 4: Importing the python control and matplotlib modules

In this exercise, you will learn how to import the python-control and matplotlib modules into your Python code.

1. Open a Jupyter Notebook file.
2. Import the python control module and matplotlib:

```python
import control
import matplotlib
```

3. Verify that the modules have been imported correctly by printing the following code:

```python
print(control.__version__)
print(matplotlib.__version__)
```

Task 5: Plotting with Matplotlib in JupyterLab

Matplotlib is a widely used plotting library in Python that provides a comprehensive set of tools for creating various types of plots and visualizations. In this tutorial, we will explore the basics of plotting with Matplotlib in JupyterLab.

1. Importing the Matplotlib Library
   Open a JupyterLab notebook or create a new one. Import the necessary Matplotlib library by running the following code in a code cell:

```python
import matplotlib.pyplot as plt
```

2. Creating Basic Plots
   Matplotlib provides multiple types of plots, such as line plots, scatter plots, bar plots, histograms, and more. Let's start with a line plot. Create two lists representing the x and y coordinates of the points:

```python
x = [1, 2, 3, 4, 5]
y = [2, 4, 6, 8, 10]
```

   Plot the line graph using the plot() and show() function:

```python
plt.plot(x, y)
plt.show()
```

   You will see a new window pop up displaying the line plot.

3. Customizing the Plot
   You can customize various aspects of the plot, such as the title, labels, axes limits, and more. Let's add some customizations to the line plot:

```python
plt.plot(x, y)
plt.title('Line Plot')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.xlim(0, 6)
plt.ylim(0, 12)
plt.grid(True)
plt.show()
```

   Run the code cell to display the customized line plot.

4. Creating Other Types of Plots
   Matplotlib supports a wide range of plot types. Let's create a scatter plot and a bar plot as examples. For the scatter plot, define two lists representing the x and y coordinates of the points and Create the scatter plot using the scatter() function:

```python
x = [1, 2, 3, 4, 5]
y = [2, 4, 6, 8, 10]

plt.scatter(x, y)
plt.title('Scatter Plot')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.grid(True)
```

```
plt.show()
```

For the bar plot, define a list of categories and their corresponding values and create the bar plot using the bar() function:

```
Python
categories = ['A', 'B', 'C', 'D', 'E']
values = [20, 35, 30, 15, 25]

plt.bar(categories, values)
plt.title('Bar Plot')
plt.xlabel('Categories')
plt.ylabel('Values')
plt.grid(True)
plt.show()
```

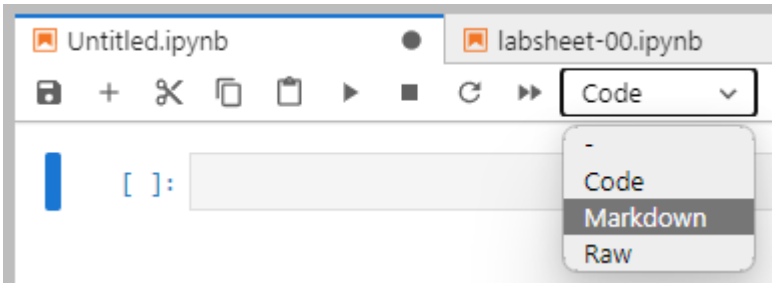
5. Saving the Plot to an Image File
   You can save the plot as an image file directly from JupyterLab. For example, to save the bar plot as a PNG image, add the following code:

```
Python
plt.bar(categories, values)
plt.title('Bar Plot')
plt.xlabel('Categories')
plt.ylabel('Values')
plt.grid(True)
plt.savefig('bar_plot.png')
```

Run the code cell, and the bar plot will be saved as a PNG file in the current directory.

Task 6: More about Jupyterlab

1. Markdown on Jupyterlab
   Markdown is a lightweight markup language that allows you to format text and incorporate elements like headings, lists, images, links, and more in a simple and readable manner. Jupyter Notebook supports Markdown cells, enabling you to create richly formatted documentation, reports, and presentations within your notebooks. In this tutorial, we will explore the basic syntax and features of Markdown in Jupyter Notebook.



````
Unset
Markdown supports tables:

| Column 1 | Column 2 |
| -------- | -------- |
| Row 1, Column 1 | Row 1, Column 2 |
| Row 2, Column 1 | Row 2, Column 2 |

You can display code blocks with syntax highlighting:
```Python
def multiply_numbers(num1, num2=1):
    return num1 * num2

result = multiply_numbers(5)
print(result)  # Output: 5
```

You can include mathematical equations using LaTeX syntax:

$$
m_n = k_p*e_n + \frac{k_e*T}{T_{reset}}\sum_{i=0}^{n}e_i + k_d\frac{e_n - e_{n-1}}{\delta t} + m_{R}
$$

You can use basic formatting options like *italic*, **bold**, and `code` highlighting.

Markdown supports both ordered and unordered lists:

- Unordered List Item 1
- Unordered List Item 2
- Unordered List Item 3

1. Ordered List Item 1
2. Ordered List Item 2
3. Ordered List Item 3

You can add links to websites or local files:

- [Google Website](https://google.com)
- [labsheet-00.ipynb](labsheet-00.ipynb)

Embedding images is also possible:
![Google Logo](google.png)
````

Additional Resource:

- https://www.datacamp.com/tutorial/markdown-in-jupyter-notebook
- https://saturncloud.io/blog/jupyter-markdown-cheat-sheet-a-quick-guide-for-data-scientists/

2. Ipywidgets

   JupyterLab provides an interactive environment where you can create dynamic and interactive user interfaces using widgets. Widgets allow users to interact with code

and visualize data in real-time. In this tutorial, we will explore the basics of using widgets in JupyterLab.

Before we start, ensure that you have the necessary packages installed. Open a terminal and run the following command to install the required packages:

```
Unset
pip install ipywidgets
```

Here's an example code that demonstrates the usage of a few commonly used ipywidgets:

```
Python
import ipywidgets as widgets
from IPython.display import display

# Create a text input widget
text_input = widgets.Text(value='Hello', description='Enter text:')

# Create a button widget
button = widgets.Button(description='Click me!')

# Create a dropdown menu widget
dropdown = widgets.Dropdown(options=['Option 1', 'Option 2', 'Option 3'],
description='Select an option:')

# Create a slider widget
slider = widgets.IntSlider(value=50, min=0, max=100, step=1,
description='Slider:')

# Create an output widget
output = widgets.Output()

# Event handler for the button click
def on_button_click(button):
    with output:
        print(f"Text: {text_input.value}")
        print(f"Selected option: {dropdown.value}")
        print(f"Slider value: {slider.value}")

# Assign the event handler to the button's on_click event
button.on_click(on_button_click)

# Display the widgets
display(text_input)
display(button)
display(dropdown)
display(slider)
display(output)
```

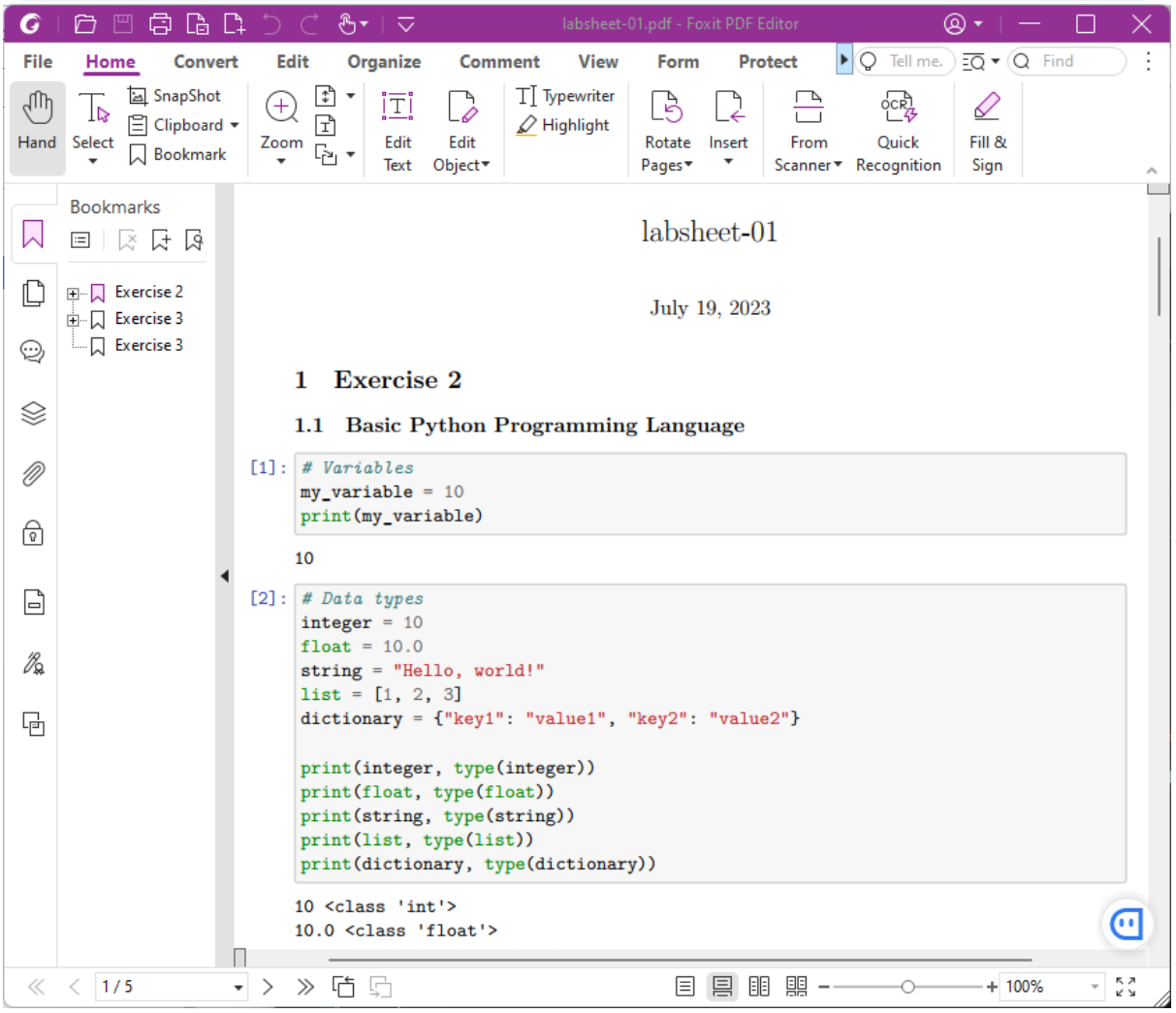
3. Exporting Jupyter Notebook (ipynb) to PDF
   Jupyter Notebook provides a simple and convenient way to export your notebooks to PDF format. In this tutorial, we will walk through the steps to export a Jupyter Notebook as a PDF document.

Before we start, ensure that you have the required packages installed. First download pandoc here: https://pandoc.org/installing.html and install on your machine. Second, open a terminal or command prompt and run the following command:

```
Unset
pip install nbconvert
```

Open a terminal or command prompt and navigate to the directory where your Jupyter Notebook (.ipynb file) is located. Run the following command to convert the notebook to PDF:

```
Unset
jupyter nbconvert --to pdf notebook.ipynb
```



Assignment:

1. Write a Python program that calculates the sum of all numbers from 1 to a given number (inclusive). The program should take a number as input from the user and print the sum.

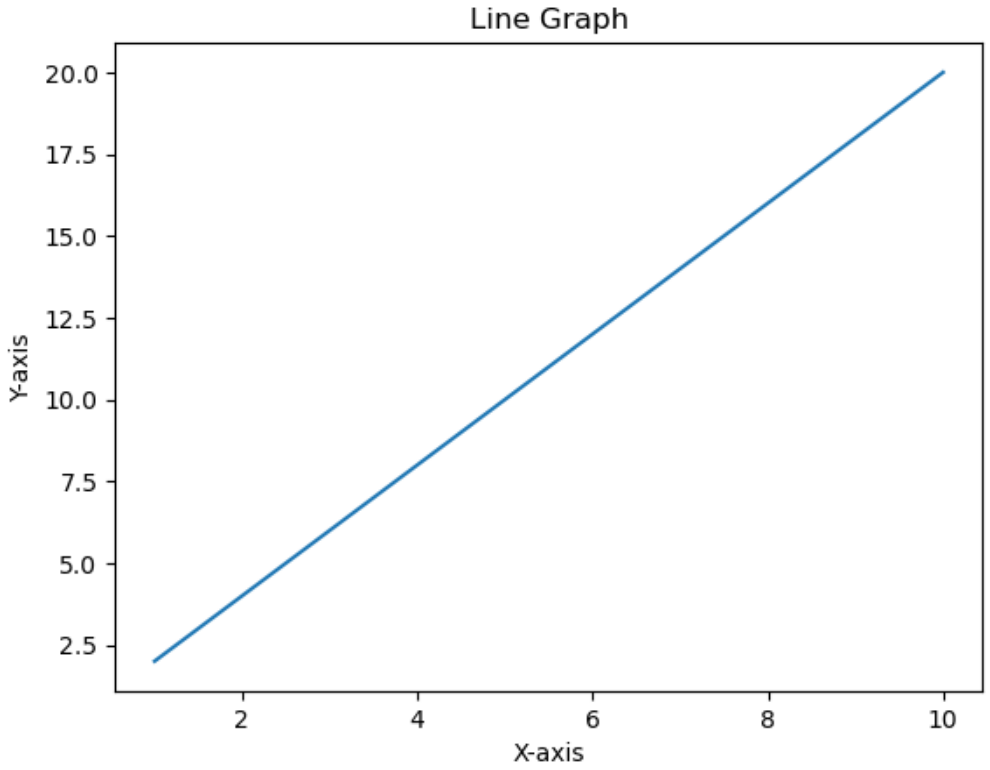
```
Enter a number:  100
The sum of numbers from 1 to 100 is: 5050
```

2. Write a Python program that checks if a given number is prime. The program should take a number as input from the user and print "Prime" or "Not Prime" accordingly.

```
Enter a number:  99
99 is not a prime number.
```

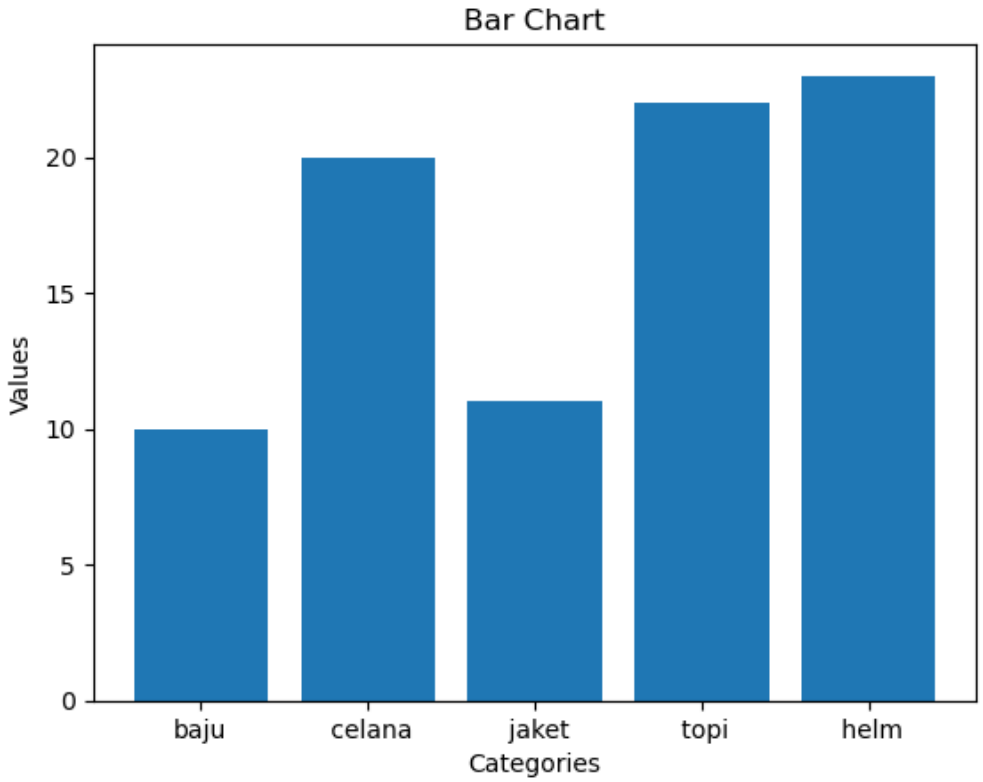
3. Write a Python program that plots a line graph of a given list of x and y coordinates. The program should take the x and y coordinates as input from the user and display the line graph using Matplotlib.

```
Enter the x-coordinates (comma-separated):  1,2,3,4,5,6,7,8,9,10
Enter the y-coordinates (comma-separated):  2,4,6,8,10,12,14,16,18,20
```

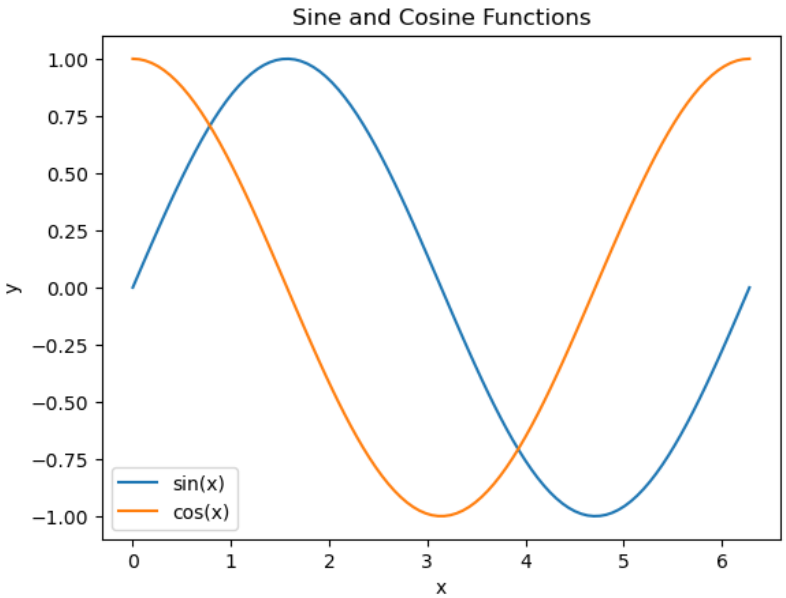


4. Write a Python program that plots a bar chart of a given list of categories and their corresponding values. The program should take the categories and values as input from the user and display the bar chart using Matplotlib.

```
Enter the categories (comma-separated):  baju, celana, jaket, topi, helm
Enter the values (comma-separated):  10, 20, 11, 22, 23
```



5. Write a Python program that plots the sine and cosine functions on the same figure using Matplotlib. The program should generate x values from 0 to 2π and calculate the corresponding y values for the sine and cosine functions. After plotting the functions, the program should display the figure with appropriate labels, title, and a legend.



6. Write a Python program in Jupyter Notebook that creates an interactive plot of the cosine function using Matplotlib. The program should include a slider widget that allows the user to adjust the frequency and amplitude of the cosine function

dynamically. As the user adjusts the slider, the plot should update in real-time to reflect the changes.