LAPORAN PRAKTIKUM POSTTEST 2
ALGORITMA DAN PEMROGRAMAN DASAR

Muhammad Zidane Abdul Kadir **2509106021**
Kelas (A1’25)

PROGRAM STUDI INFORMATIKA

UNIVERSITAS MULAWARMAN

SAMARINDA

2025

BACKGROUND

Case Study :

Daffa membuka layanan kesehatan dasar. Ia ingin membuat program kecil untuk membantu pasien mengecek status berat badan berdasarkan tinggi badan dan berat badan. Program ini akan membantu pasien tau apakah dirinya termasuk kelebihan berat badan atau tidak.

Program harus memiliki :
DILARANG MENGGUNAKAN PERCABANGAN (IF ELSE)

Input user

- Nama pasien (string)
- Tinggi badan dalam cm (float/int)
- Berat badan dalam kg (float / int)

Rumus :

- beratIdeal = (tinggiBadan - 100)
- isKelebihan = beratBadan > beratIdeal

Clue :
Gunakan list untuk mempermudah proses print status
status = statusList[int(isKelebihan)] #cara print status

Contoh hasil output :

```
--------------------------------------------------------------------------------
|        HASIL CEK BERAT BADAN                                                 |
--------------------------------------------------------------------------------
| Nama Pasien       : Dapupu                                                   |
| Tinggi Badan      : 170 cm                                                   |
| Berat Badan       : 80 kg                                                    |
| Berat Ideal       : 70 kg                                                    |
| Status            : Kelebihan Berat Badan                                    |
--------------------------------------------------------------------------------
```

Point plus++:

- jika output rapi (menggunakan tabel)
- menggunakan f-string pada print (ex, print(f"Nama Pasien {nama}")

Buatlah Flowchart dan programnya, kemudian buat repository baru dengan nama praktikum-apd

From the case study above, we can conclude that we are ask to create a program that helps patients check their weight status based on their height and weight, without using conditional statements (if else). First, we are required to build a flowchart and a python program that calculates the ideal body weight using the given formula. Then, we must determine whether the patient is overweight by utilizing a list and indexing technique instead of conditional branching.

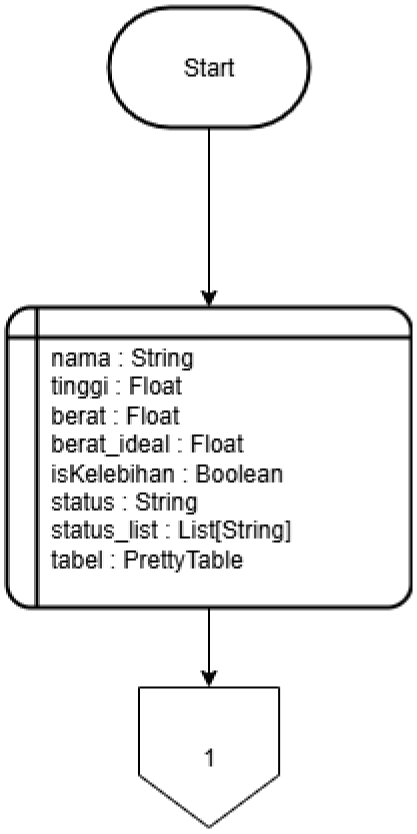
Next, the program should display the results neatly in a formatted table using f-strings to ensure the output is well organized. Finally, the task also requires us to place the project into a new repository named “praktikum-apd”

SOLUTIONS

My solution is to first create a flowchart for the case study above, and then develop the program using the Python programming language.

A. Flowchart

Before creating the program, we need to make the flowchart first, which is useful for deciding and understanding the algorithm of the program we are going to make.

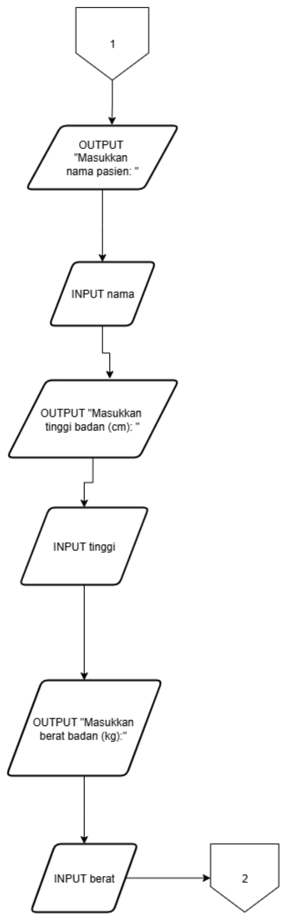


Flowchart Picture 1

From **Flowchart Picture 1,** in the first step of the flowchart begins with the declaration of variables and their corresponding data types. These variables are essential for storing and processing user input, intermediate values, and output results.

- nama : String → Stores the patient's name.
- tinggi : Float → Stores the patient's height in centimeters.
- berat : Float → Stores the patient's weight in kilograms.
- berat_ideal : Float → Stores the calculated ideal body weight based on the given formula.
- isKelebihan : Boolean → A logical variable to determine whether the patient is overweight or not.
- status : String → Stores the weight status description.
- status_list : List[String] → Stores a predefined list of possible weight status categories.
- tabel : PrettyTable → Used to display the final results in a neatly formatted table.

This step ensures that all required variables are properly defined before proceeding with user input and calculations.

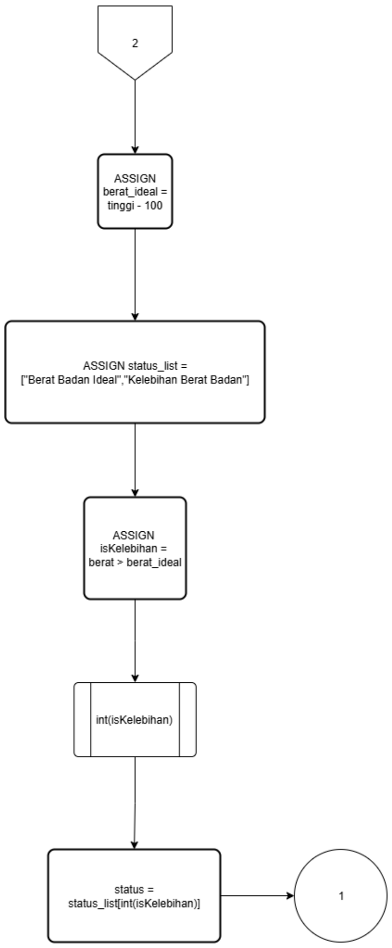

1
OUTPUT "Masukkan nama pasien: "
INPUT nama
OUTPUT "Masukkan tinggi badan (cm): "
INPUT tinggi
OUTPUT "Masukkan berat badan (kg):"
INPUT berat
2

Flowchart Picture 2

In this step, the program begins interacting with the user by requesting input data. The sequence of operations is as follows:

1. The program displays a message: **"Masukkan nama pasien:"** to prompt the user.
   - The input is then stored in the variable **nama**.
2. Next, the program asks: **"Masukkan tinggi badan (cm):"**.
   - The input is stored in the variable **tinggi** as a floating-point number (representing centimeters).
3. Finally, the program outputs: **"Masukkan berat badan (kg):"**.
   - The input is stored in the variable **berat** as a floating-point number (representing kilograms).

At the end of this stage, the program has collected all the necessary personal data (name, height, and weight) from the user, which will be used for further calculations in the next steps of the flowchart.


2
ASSIGN berat_ideal = tinggi - 100
ASSIGN status_list = ["Berat Badan Ideal","Kelebihan Berat Badan"]
ASSIGN isKelebihan = berat > berat_ideal
int(isKelebihan)
status = status_list[int(isKelebihan)]
1

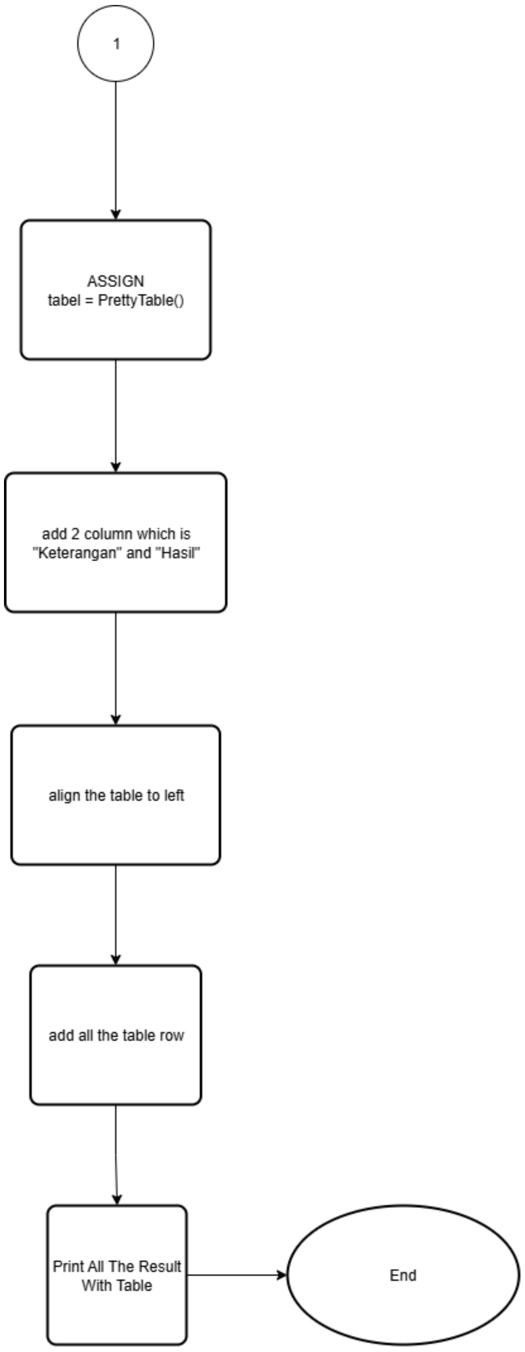
Flowchart Picture 3

From **Flowchart Picture 3**, the program performs the calculation process to determine the patient's weight status:

1. **Calculate Ideal Weight**
   - The variable **berat_ideal** is calculated using the formula:
     berat_ideal = tinggi − 100
2. This represents a simple estimation of the patient's ideal body weight.
3. **Prepare Status Options**
   - The variable **status_list** is defined as a list containing two possible outcomes:
     - *"Berat Badan Ideal"*
     - *"Kelebihan Berat Badan"*
4. **Check Overweight Condition**
   - The program evaluates the condition:
     isKelebihan = berat > berat_ideal
   - If the patient's actual weight is greater than the ideal weight, **isKelebihan** will be True. Otherwise, it will be False.
5. **Convert Boolean to Index**
   - The Boolean value of **isKelebihan** is converted into an integer (False = 0, True = 1).
6. **Select Weight Status**

- Finally, the program assigns the correct description to **status** by selecting from **status_list** using the Boolean-as-integer index.
- If False (0) → status = *“Berat Badan Ideal”*
- If True (1) → status = *“Kelebihan Berat Badan”*

This step ensures that the program can automatically determine and store the patient’s weight status based on the given inputs.


1
ASSIGN
tabel = PrettyTable()
add 2 column which is "Keterangan" and "Hasil"
align the table to left
add all the table row
Print All The Result With Table
End

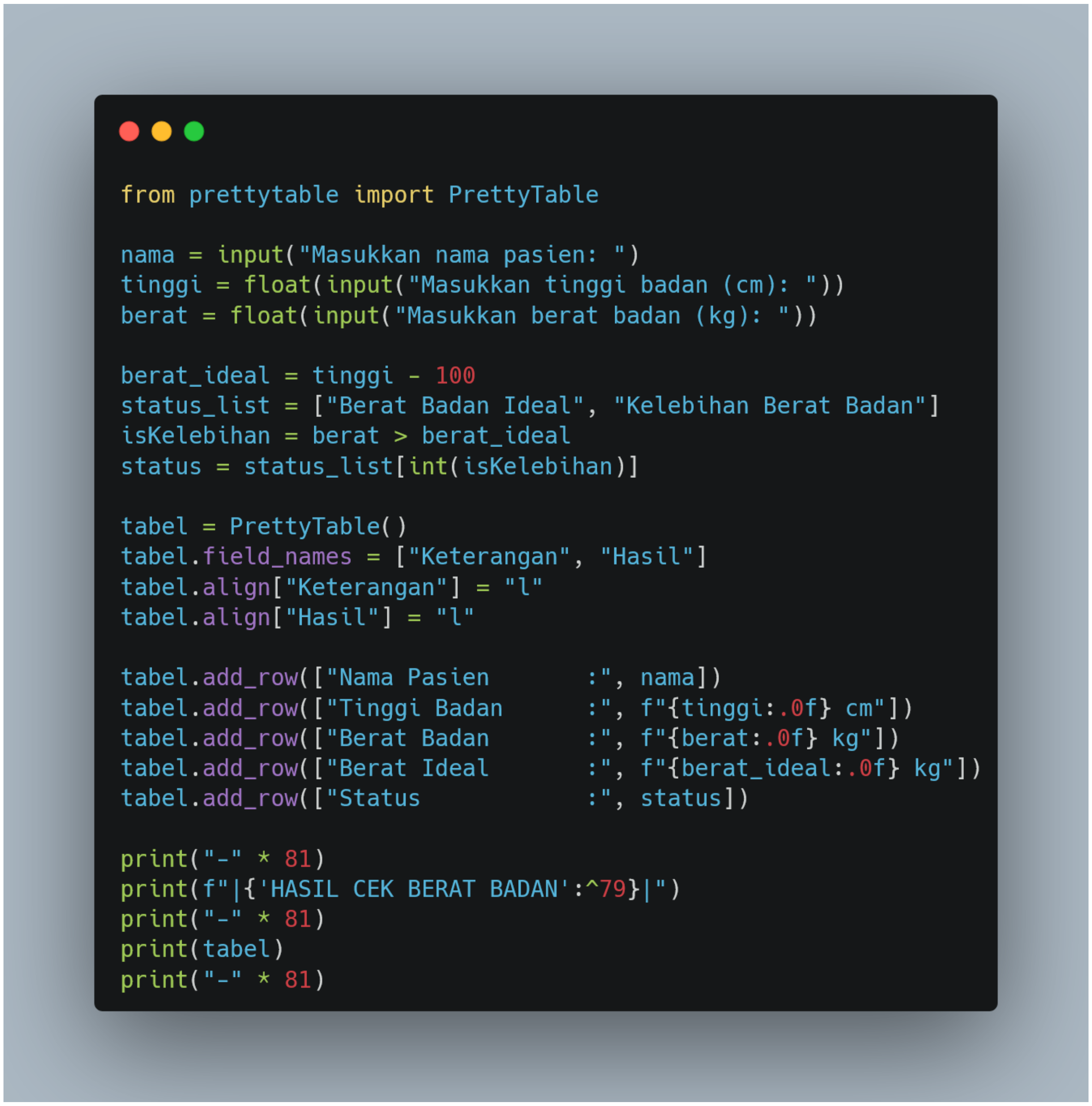
Flowchart Picture 4

From Flowchart Picture 4, it should be noted that this step introduces the use of a table structure for displaying results. While the implementation in the program makes use of a library feature, such details about external libraries and object-oriented programming (OOP) concepts are beyond the current scope of what has been taught to me even tho I already learn it myself and not using ChatGPT or some AI. For this reason, the focus here will remain on the logical process of preparing and presenting the output in a easier format :

1. **Initialize Table**
   - A table is created to format the output in a structured form.
2. **Define Columns**
   - Two columns are added:
     - **"Keterangan"** → description of the data.
     - **"Hasil"** → the actual values (such as name, height, weight, and status).
3. **Set Table Alignment**
   - The table is aligned to the left for better readability.
4. **Insert Rows**
   - The patient's data (name, height, weight, ideal weight, and status) is inserted into the table row by row.
5. **Display Results**
   - Finally, the program prints the complete table to show a clear and organized summary of the patient's information.

At this final stage, the program not only completes the calculation but also presents the results in a neat and user-friendly format.

B. Code



```python
from prettytable import PrettyTable

nama = input("Masukkan nama pasien: ")
tinggi = float(input("Masukkan tinggi badan (cm): "))
berat = float(input("Masukkan berat badan (kg): "))

berat_ideal = tinggi - 100
status_list = ["Berat Badan Ideal", "Kelebihan Berat Badan"]
isKelebihan = berat > berat_ideal
status = status_list[int(isKelebihan)]

tabel = PrettyTable()
tabel.field_names = ["Keterangan", "Hasil"]
tabel.align["Keterangan"] = "l"
tabel.align["Hasil"] = "l"

tabel.add_row(["Nama Pasien       :", nama])
tabel.add_row(["Tinggi Badan      :", f"{tinggi:.0f} cm"])
tabel.add_row(["Berat Badan       :", f"{berat:.0f} kg"])
tabel.add_row(["Berat Ideal       :", f"{berat_ideal:.0f} kg"])
tabel.add_row(["Status            :", status])

print("-" * 81)
print(f"|{'HASIL CEK BERAT BADAN':^79}|")
print("-" * 81)
print(tabel)
print("-" * 81)
```

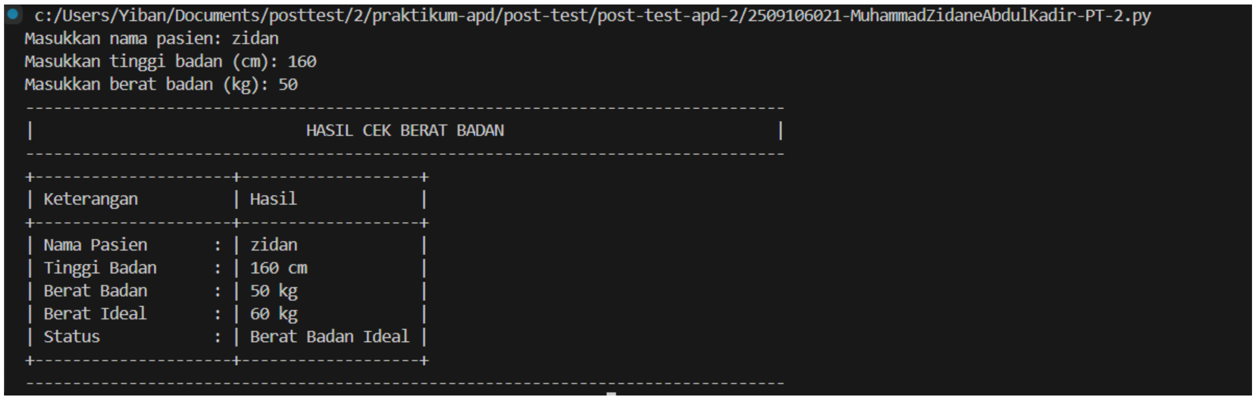
Code 1

From **Code 1**, the program is written in Python to calculate and display a patient’s weight status based on their height and weight. The program begins by importing **PrettyTable**, which is used to organize the output in a table format so that the results appear neat and easy to read.
Next, the program asks the user to input three pieces of information: the patient’s name, height in centimeters, and weight in kilograms. The height and weight values are converted into floating-point numbers to allow for numerical calculations.
After collecting the input, the program calculates the patient’s **ideal weight** using the simple formula:

Berat_ideal = tinggi - 100

It then prepares a list of possible outcomes, namely “Berat Badan Ideal” and “Kelebihan Berat Badan”. The program checks whether the actual weight is greater than the ideal weight and stores this result in a Boolean variable. This Boolean is then converted into an integer (0 or 1) to select the correct status from the list, which is stored in the variable **status**.
Once the status is determined, the program creates a table structure with two columns: **“Keterangan”** and **“Hasil”**. The table is set to align its contents to the left for better readability.
Afterwards, the program adds rows to the table one by one, containing the patient’s name, height, weight, calculated ideal weight, and the resulting status.

C. Output

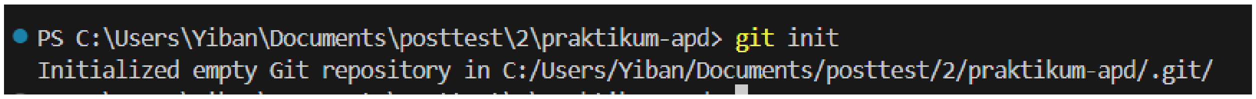


```
c:/Users/Yiban/Documents/posttest/2/praktikum-apd/post-test/post-test-apd-2/2509106021-MuhammadZidaneAbdulKadir-PT-2.py
 Masukkan nama pasien: zidan
 Masukkan tinggi badan (cm): 160
 Masukkan berat badan (kg): 50
 --------------------------------------------------------------------------------
 |                             HASIL CEK BERAT BADAN                            |
 --------------------------------------------------------------------------------
 +--------------------+-------------------+
 | Keterangan         | Hasil             |
 +--------------------+-------------------+
 | Nama Pasien      : | zidan             |
 | Tinggi Badan     : | 160 cm            |
 | Berat Badan      : | 50 kg             |
 | Berat Ideal      : | 60 kg             |
 | Status           : | Berat Badan Ideal |
 +--------------------+-------------------+
 --------------------------------------------------------------------------------
```

Finally, the program prints a formatted heading titled “HASIL CEK BERAT BADAN”, followed by the completed table. Borders made of dashes are added above and below the output to make it visually clearer and more structured.
In conclusion, from **Code 1**, the program not only performs the necessary calculations to determine whether a patient has an ideal weight or excess weight but also displays the results in a structured, professional-looking table format.

D. Git

i) Git Init



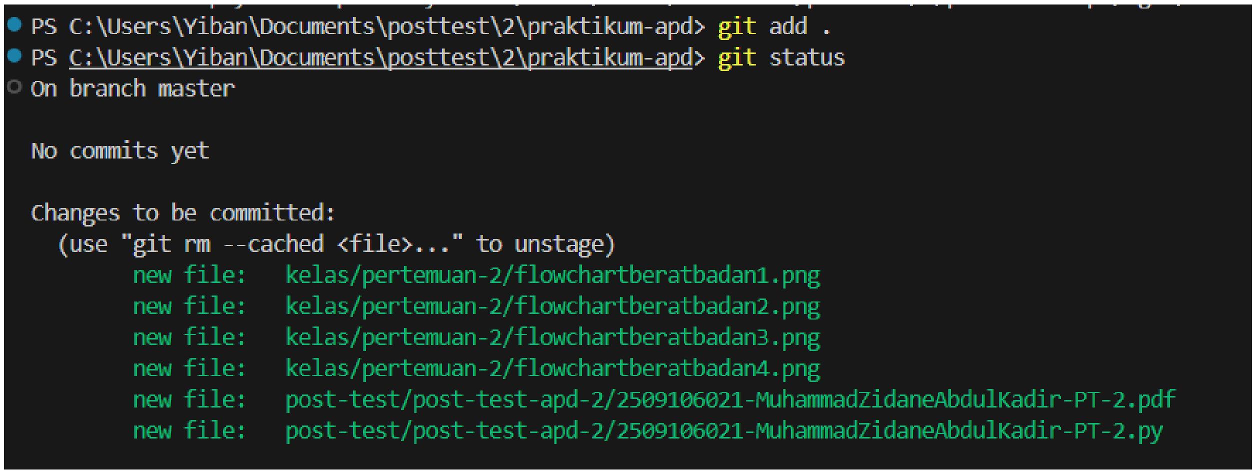
```
PS C:\Users\Yiban\Documents\posttest\2\praktikum-apd> git init
Initialized empty Git repository in C:/Users/Yiban/Documents/posttest/2/praktikum-apd/.git/
```

The command **git init** is used to create a new, empty Git repository in the specified folder. In this example, the repository was initialized in the directory:

C:\Users\Yiban\Documents\posttest\2\praktikum-apd\

After running this command, a hidden .git folder is created inside the project, which stores all version control information for tracking changes to files.

ii) Git Add



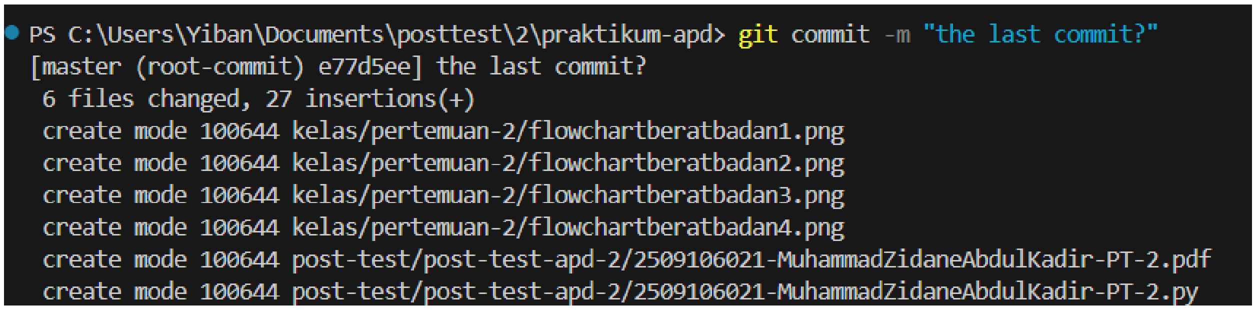
```
PS C:\Users\Yiban\Documents\posttest\2\praktikum-apd> git add .
PS C:\Users\Yiban\Documents\posttest\2\praktikum-apd> git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   kelas/pertemuan-2/flowchartberatbadan1.png
        new file:   kelas/pertemuan-2/flowchartberatbadan2.png
        new file:   kelas/pertemuan-2/flowchartberatbadan3.png
        new file:   kelas/pertemuan-2/flowchartberatbadan4.png
        new file:   post-test/post-test-apd-2/2509106021-MuhammadZidaneAbdulKadir-PT-2.pdf
        new file:   post-test/post-test-apd-2/2509106021-MuhammadZidaneAbdulKadir-PT-2.py
```

The command **git add .** is used to add all new and modified files in the current directory to the Git staging area. This means the files are prepared to be included in the next commit. After that, the command git status is used to check the current state of the repository. It shows that several new files (images, PDF, and Python script) have been staged and are ready to be committed.

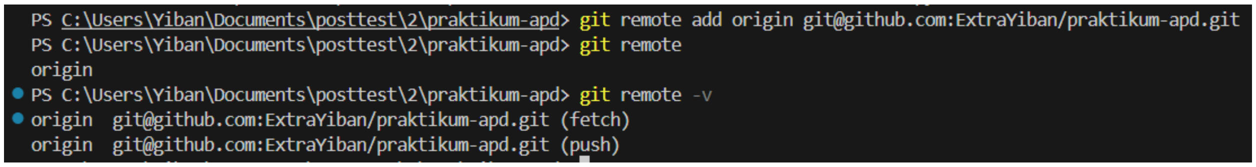
iii) Git Commit



```
PS C:\Users\Yiban\Documents\posttest\2\praktikum-apd> git commit -m "the last commit?"
[master (root-commit) e77d5ee] the last commit?
 6 files changed, 27 insertions(+)
 create mode 100644 kelas/pertemuan-2/flowchartberatbadan1.png
 create mode 100644 kelas/pertemuan-2/flowchartberatbadan2.png
 create mode 100644 kelas/pertemuan-2/flowchartberatbadan3.png
 create mode 100644 kelas/pertemuan-2/flowchartberatbadan4.png
 create mode 100644 post-test/post-test-apd-2/2509106021-MuhammadZidaneAbdulKadir-PT-2.pdf
 create mode 100644 post-test/post-test-apd-2/2509106021-MuhammadZidaneAbdulKadir-PT-2.py
```

The command **git commit -m "the last commit?"** is used to save the staged changes into the Git repository along with a descriptive message. The message helps identify the purpose of the commit. In this example, six files (images, a PDF, and a Python file) were successfully committed. This marks the first snapshot of the project in the repository.

iv) Git Remote



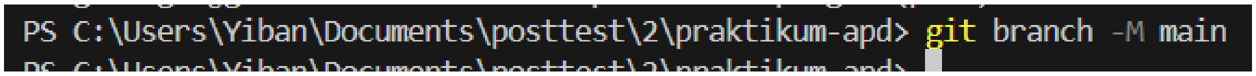
```
PS C:\Users\Yiban\Documents\posttest\2\praktikum-apd> git remote add origin git@github.com:ExtraYiban/praktikum-apd.git
PS C:\Users\Yiban\Documents\posttest\2\praktikum-apd> git remote
origin
PS C:\Users\Yiban\Documents\posttest\2\praktikum-apd> git remote -v
origin  git@github.com:ExtraYiban/praktikum-apd.git (fetch)
origin  git@github.com:ExtraYiban/praktikum-apd.git (push)
```

The command **git remote add origin git@github.com:ExtraYiban/praktikum-apd.git** is used to connect the local Git repository to a remote repository on GitHub. Here, the remote is given the name **origin**, which is the default name for the main remote repository.

The command git remote shows the list of remote names linked to the local repository, while git remote -v provides detailed information, including the URL for **fetching** and **pushing** changes.

In this example, the remote named **origin** points to the GitHub repository: **git@github.com:ExtraYiban/praktikum-apd.git**

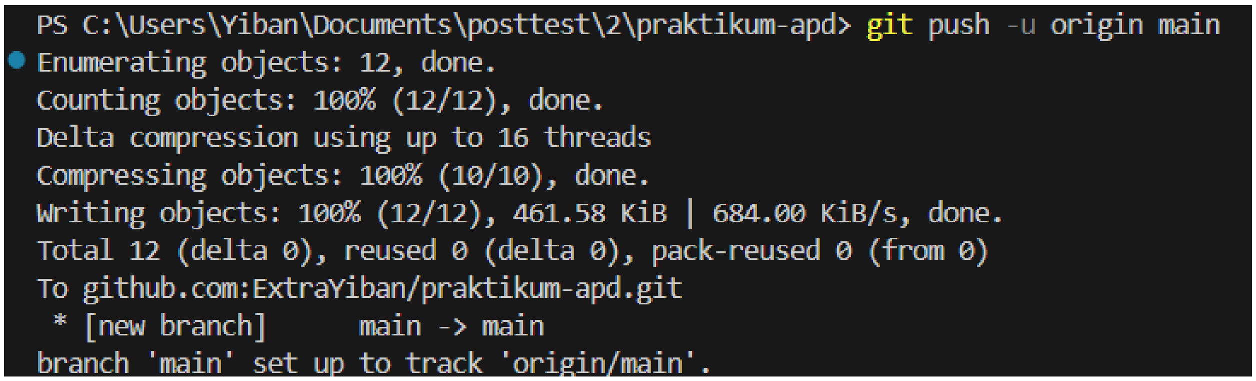
v) Git Branch



```
PS C:\Users\Yiban\Documents\posttest\2\praktikum-apd> git branch -M main
```

This command creates and switches to a new branch named **main**. If a branch with that name already exists, it will be overwritten.

vi) Git Push



```
PS C:\Users\Yiban\Documents\posttest\2\praktikum-apd> git push -u origin main
Enumerating objects: 12, done.
Counting objects: 100% (12/12), done.
Delta compression using up to 16 threads
Compressing objects: 100% (10/10), done.
Writing objects: 100% (12/12), 461.58 KiB | 684.00 KiB/s, done.
Total 12 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To github.com:ExtraYiban/praktikum-apd.git
 * [new branch]      main -> main
branch 'main' set up to track 'origin/main'.
```

This command pushes your local main branch to the remote repository (named origin) and sets up tracking so that future git pull or git push commands will automatically use origin/main. The output shows Git is compressing and sending 12 objects, then confirms the branch was successfully pushed and set to track the remote branch. This is a standard step after creating or updating a local branch to sync it with GitHub or another remote server.