**Page Replacement Algorithm Simulator**

**Introduction:**

This program simulates three-page replacement algorithms:

1. FIFO (First-In-First-Out)
2. LRU (Least Recently Used)
3. Optimal

It allows the user to:

* Input a page reference string (a sequence of page requests).
* Select an algorithm from FIFO, LRU, or Optimal.
* Set the number of frames (memory slots available for pages).
* View a graphical representation of how the pages are stored in memory over time.
* Get statistics on page faults and hit ratio.

**Importing Required Libraries**

1. import tkinter as tk

2. from tkinter import ttk, messagebox

3. import numpy as np

4. import matplotlib.pyplot as plt

* tkinter: Python’s built-in library for creating GUI applications.
* ttk: Provides themed widgets (e.g., dropdown menus, buttons) for better UI.
* messagebox: Used to show popup messages (e.g., errors, results).
* numpy (np): Used for handling arrays and numerical data.
* matplotlib.pyplot (plt): Used to plot the page replacement process.

**FIFO Page Replacement Algorithm**

1. def fifo\_page\_replacement(pages, frames):

2. frame\_list = []

3. history = []

4. fault\_positions = []

5. hits, misses = 0, 0

6. fifo\_index = 0 # Tracks which index to replace

7.

8. for i, page in enumerate(pages):

9. if page in frame\_list:

10. hits += 1

11. else:

12. misses += 1

13. if len(frame\_list) < frames:

14. frame\_list.append(page)

15. else:

16. replaced\_index = fifo\_index # Get index of oldest page

17. frame\_list[replaced\_index] = page # Replace at the same position

18. fault\_positions.append((i, page, replaced\_index)) # Store step, new page, replaced index

19. fifo\_index = (fifo\_index + 1) % frames # Move FIFO pointer

20.

21. history.append(frame\_list.copy())

22.

23. return history, hits, misses, fault\_positions

24.

* Purpose: Implements First-In-First-Out (FIFO) Page Replacement, where the oldest page is replaced when memory is full.
* Inputs:
  + pages: List of page references.
  + frames: Number of available memory slots.
* Process:
  + - Check if the page is in memory → If yes, it's a hit; otherwise, it's a miss (page fault).
    - If memory is not full, add the new page.
    - If memory is full, replace the oldest page (tracked using fifo\_index).
    - Update history to store frame contents at each step.
* Returns:
  + history → Memory state at each step.
  + hits → Number of successful page accesses.
  + misses → Number of page faults.
  + fault\_positions → Details of replaced pages.

Key Point: FIFO is simple but non-optimal as it does not consider future page requests

**LRU Page Replacement Algorithm**

1. def lru\_page\_replacement(pages, frames):

2. frame\_list = [] # Stores pages in memory

3. history = [] # Stores frame state history

4. fault\_positions = [] # Stores replaced page details

5. indexes = {} # Stores last used index of each page

6.

7. hits, misses = 0, 0

8.

9. for i, page in enumerate(pages):

10. if page in frame\_list:

11. hits += 1

12. indexes[page] = i # Update last used index

13. else:

14. misses += 1

15. if len(frame\_list) < frames:

16. frame\_list.append(page) # Add page if space available

17. else:

18. # Find the least recently used page (smallest index in 'indexes')

19. lru\_page = min(indexes, key=indexes.get) # Page with the lowest index

20. replaced\_index = frame\_list.index(lru\_page) # Find its position in the frame

21. frame\_list[replaced\_index] = page # Replace with new page

22. fault\_positions.append((i, page, replaced\_index)) # Store fault info

23. indexes.pop(lru\_page) # Remove old page from usage tracker

24.

25. indexes[page] = i # Update last used index

26.

27. history.append(frame\_list.copy()) # Store frame state

28.

29. return history, hits, misses, fault\_positions

30.

* LRU removes the least recently used page.
* Uses page\_order to track page usage.
* When memory is full:
  + It removes the page that was used the longest time ago.
  + Updates page\_order every time a page is accessed.

**Optimal Page Replacement Algorithm**

1.  def optimal\_page\_replacement(pages, frames):

2. frame\_list = [] # Stores current pages in memory

3. history = [] # Stores state of frames at each step

4. fault\_positions = [] # Stores (step, new page, replaced index)

5. hits, misses = 0, 0

6.

7. for i, page in enumerate(pages):

8. if page in frame\_list:

9. hits += 1 # Page hit

10. else:

11. misses += 1 # Page fault

12.

13. if len(frame\_list) < frames:

14. frame\_list.append(page) # Fill empty frames first

15. else:

16. # Dictionary to store the next occurrence index of each page in frame

17. future\_use = {frame: float('inf') for frame in frame\_list}

18.

19. for frame in frame\_list:

20. if frame in pages[i+1:]: # Check if frame appears in future

21. future\_use[frame] = pages[i+1:].index(frame) + i + 1 # Absolute index

22.

23. # Find the page that is used farthest in the future

24. page\_to\_replace = max(future\_use, key=future\_use.get)

25. replaced\_index = frame\_list.index(page\_to\_replace)

26.

27. # Replace the page

28. frame\_list[replaced\_index] = page

29. fault\_positions.append((i, page, replaced\_index)) # Store replacement step

30.

31. history.append(frame\_list.copy())

32.

33. return history, hits, misses, fault\_positions

34.

* Optimal removes the page that will be used farthest in the future.
* It scans ahead to determine which page will be needed last.
* This results in the fewest page faults but is impractical in real-world systems.

**Visualisation :-**

1.  def visualize\_page\_replacement(algorithm, pages, frames, history, fault\_positions, hits, misses):

2. fig, ax = plt.subplots(figsize=(12, 6))

3. ax.set\_title(f'{algorithm} Page Replacement Visualization', fontsize=16, fontweight='bold')

4.

5. num\_steps = len(pages)

6. num\_frames = frames

7.

8. # Create table data

9. table\_data = [[''] \* num\_steps for \_ in range(num\_frames)]

10. cell\_colors = [['white'] \* num\_steps for \_ in range(num\_frames)]

11.

12. # Store replacement steps for proper visualization

13. replaced\_positions = {pos[0]: (pos[1], pos[2]) for pos in fault\_positions} # (step: (new\_page, position))

14.

15. for col, page in enumerate(pages):

16. frame\_state = history[col] # Get frame content at current step

17.

18. for row in range(num\_frames):

19. if row < len(frame\_state):

20. table\_data[row][col] = str(frame\_state[row])

21.

22. # Highlight replaced pages in red

23. if col in replaced\_positions and row == replaced\_positions[col][1]:

24. cell\_colors[row][col] = 'red'

25. else:

26. cell\_colors[row][col] = 'lightblue'

27.

28. table\_data = np.array(table\_data)

29.

30. ax.axis('tight')

31. ax.axis('off')

32. table = ax.table(cellText=table\_data, cellLoc='center', loc='center',

33. cellColours=cell\_colors, colLabels=[str(p) for p in pages],

34. colColours=['lightblue'] \* num\_steps)

35.

36. plt.show()

37.

This function, visualize\_page\_replacement, is designed to visualize the execution of a page replacement algorithm using a table format. Below are the main points explaining its functionality:

1. Function Purpose

* The function generates a graphical representation of page replacement using Matplotlib.
* It visually shows the step-by-step execution of a page replacement algorithm.
* The visualization includes page hits, misses, and replaced pages.

2. Parameters

| Parameter | Description |
| --- | --- |
| algorithm | Name of the page replacement algorithm (e.g., FIFO, LRU, Optimal). |
| pages | List of page references (sequence of page requests). |
| frames | Number of available frames in memory. |
| history | A list of lists representing the state of memory frames at each step. |
| fault\_positions | List of tuples (step, new\_page, position), indicating where page faults occur and which page is replaced. |
| hits | Total number of page hits (not directly used in visualization). |
| misses | Total number of page faults (not directly used in visualization). |

3. Steps in the Code

1. Initialize the Plot
   * Uses plt.subplots() to create a figure.
   * Sets the title based on the algorithm name.
2. Create Table Data
   * table\_data stores the page content of each memory frame at each step.
   * cell\_colors assigns background colors to cells for visualization.
3. Track Page Replacement
   * replaced\_positions is a dictionary that maps each step to (new\_page, position) where a replacement occurred.
4. Fill Table Data
   * Iterates through the page request sequence (pages).
   * Updates the table\_data with memory frame contents at each step.
   * Colors replaced pages in red and other occupied frames in light blue.
5. Display the Table
   * Uses ax.table() to create a table visualization.
   * Labels columns with page request numbers.
   * Calls plt.show() to render the visualization.

4. Key Features

✅ Shows memory frame state at each step  
✅ Highlights replaced pages in red for clarity  
✅ Allows easy comparison between different page replacement algorithms  
✅ Provides an intuitive way to analyze hits and misses visually

This function helps in understanding how a page replacement algorithm handles memory and which pages get replaced at each step.  
  
**GUI Implementation:**

1. def run\_simulation():

2. try:

3. page\_refs = list(map(int, entry\_pages.get().split()))

4. num\_frames = int(entry\_frames.get())

5. selected\_algo = algo\_var.get()

6.

7. if num\_frames <= 0:

8. messagebox.showerror("Error", "Number of frames must be greater than 0!")

9. return

10.

11. if not page\_refs:

12. messagebox.showerror("Error", "Reference string cannot be empty!")

13. return

14.

15. if selected\_algo == "FIFO":

16. history, hits, misses, fault\_positions = fifo\_page\_replacement(page\_refs, num\_frames)

17. elif selected\_algo == "LRU":

18. history, hits, misses, fault\_positions = lru\_page\_replacement(page\_refs, num\_frames)

19. elif selected\_algo == "Optimal":

20. history, hits, misses, fault\_positions = optimal\_page\_replacement(page\_refs, num\_frames)

21. else:

22. messagebox.showerror("Error", "Invalid algorithm selected!")

23. return

24.

25. # Computation Outline

26. result\_text.set(

27. f"Algorithm: {selected\_algo}\n"

28. f"Frames: {num\_frames}\n"

29. f"Reference Length: {len(page\_refs)}\n"

30. f"Reference String: {page\_refs}\n\n"

31. f"Page Faults: {misses}\n"

32. f"Hit Ratio: {hits / (hits + misses):.2f}\n"

33. f"Miss Ratio: {misses / (hits + misses):.2f}"

34. )

35.

36. # Plot visualization

37. visualize\_page\_replacement(selected\_algo, page\_refs, num\_frames, history, fault\_positions, hits, misses)

38.

39. except ValueError:

40. messagebox.showerror("Error", "Invalid input! Please enter space-separated integers for reference string.")

41.

1. Purpose of the Function run\_simulation()

* Handles user input and processes the page replacement simulation.
* Retrieves values from GUI input fields (entry\_pages and entry\_frames).
* Calls the appropriate page replacement algorithm based on user selection.
* Displays results in the GUI and visualizes the execution.

2. Steps Involved

Step 1: Input Handling

* Extracts Page References
  + Reads user input from entry\_pages, splits the values, and converts them into integers.
* Extracts Frame Count
  + Retrieves the number of frames from entry\_frames and converts it to an integer.
* Gets Selected Algorithm
  + Fetches the algorithm choice from algo\_var (e.g., FIFO, LRU, Optimal).

Step 2: Input Validation

* Ensures the number of frames is greater than 0.
* Ensures the reference string is not empty.
* Checks for valid algorithm selection.
* Handles invalid input cases using error messages (messagebox.showerror).

Step 3: Running the Page Replacement Algorithm

* Calls the respective function based on the selected algorithm:
  + fifo\_page\_replacement()
  + lru\_page\_replacement()
  + optimal\_page\_replacement()
* These functions return:
  + history → Step-wise memory frame content.
  + hits → Number of page hits.
  + misses → Number of page faults.
  + fault\_positions → Positions where replacements occurred.

Step 4: Displaying Results

* Prepares and sets a summary string (result\_text.set()) including:
  + Selected algorithm.
  + Number of frames.
  + Length and values of the reference string.
  + Total page faults.
  + Hit Ratio = hits / (hits + misses)
  + Miss Ratio = misses / (hits + misses)

Step 5: Visualizing the Page Replacement Process

* Calls visualize\_page\_replacement() to generate a graphical representation.

3. Key Features

✅ GUI-Based Input Handling – Users can enter data interactively.  
✅ Error Handling – Displays messages for invalid input.  
✅ Dynamic Algorithm Selection – Supports FIFO, LRU, and Optimal page replacement.  
✅ Computation & Visualization – Displays metrics and a graphical representation.

This function integrates the user interface with backend logic, ensuring a smooth and interactive experience for simulating page replacement algorithms.

**Main GUI Implementation:**

1. root = tk.Tk()

2. root.title("Page Replacement Algorithm Simulator")

3. root.geometry("500x400")

4. root.configure(bg="lightgray")

5.

6. # Title Label

7. tk.Label(root, text="Page Replacement Algorithm Simulator", font=("Arial", 14, "bold"), bg="lightgray").pack(pady=10)

8.

9. # Algorithm Selection

10. tk.Label(root, text="Select Algorithm:", bg="lightgray").pack()

11. algo\_var = tk.StringVar(value="FIFO")

12. algo\_menu = ttk.Combobox(root, textvariable=algo\_var, values=["FIFO", "LRU", "Optimal"], state="readonly")

13. algo\_menu.pack(pady=5)

14.

15. # Number of Frames

16. tk.Label(root, text="Enter Number of Frames:", bg="lightgray").pack()

17. entry\_frames = tk.Entry(root)

18. entry\_frames.pack(pady=5)

19.

20. # Reference String Input

21. tk.Label(root, text="Enter Reference String (space-separated):", bg="lightgray").pack()

22. entry\_pages = tk.Entry(root)

23. entry\_pages.pack(pady=5)

24.

25. # Run Button

26. tk.Button(root, text="Run Simulation", command=run\_simulation, bg="blue", fg="white").pack(pady=10)

27.

28. # Results Display

29. result\_text = tk.StringVar()

30. result\_label = tk.Label(root, textvariable=result\_text, justify="left", bg="white", font=("Courier", 10), relief="solid", padx=10, pady=5)

31. result\_label.pack(pady=10, fill="both")

32.

33. # Start GUI Loop

34. root.mainloop()

35.

Understanding Main GUI Window Implementation

This script creates a Graphical User Interface (GUI) using Tkinter to allow users to interactively run and visualize a Page Replacement Algorithm Simulator.

1. Creating the Main Window

* tk.Tk() initializes the main application window (root).
* root.title("Page Replacement Algorithm Simulator") → Sets the title of the window.
* root.geometry("500x400") → Defines the window size (500 pixels wide, 400 pixels tall).
* root.configure(bg="lightgray") → Sets a light gray background color.

2. Adding Interface Elements

Title Label

* Displays the title text ("Page Replacement Algorithm Simulator") in bold Arial font.
* tk.Label() creates the label and pack(pady=10) adds vertical spacing.

Algorithm Selection (Dropdown Menu)

* Allows users to choose a page replacement algorithm (FIFO, LRU, Optimal).
* algo\_var = tk.StringVar(value="FIFO") → Holds the selected algorithm (default: FIFO).
* ttk.Combobox(root, textvariable=algo\_var, values=["FIFO", "LRU", "Optimal"], state="readonly")
  + Creates a dropdown menu for algorithm selection.
  + state="readonly" ensures users can only pick from the given options.

Number of Frames Input

* Label prompts the user to enter the number of memory frames.
* entry\_frames = tk.Entry(root) → Creates an input field for numerical entry.

Reference String Input

* Label instructs the user to enter a sequence of space-separated page references.
* entry\_pages = tk.Entry(root) → Creates an input field for entering the page reference string.

Run Simulation Button

* **Executes the page replacement simulation when clicked.**
* tk.Button(root, text="Run Simulation", command=run\_simulation, bg="blue", fg="white")
  + command=run\_simulation → Calls the run\_simulation() function when clicked.
  + Blue background, white text for visibility.

Results Display

* Displays simulation results dynamically.
* result\_text = tk.StringVar() → Holds the output text.
* result\_label = tk.Label(root, textvariable=result\_text, justify="left", bg="white", font=("Courier", 10), relief="solid", padx=10, pady=5)
  + Displays results inside a white box.
  + Uses Courier font (monospaced for better alignment).
  + Relief="solid" → Adds a border for clarity.
  + fill="both" → Ensures the label expands to fit content.

3. Starting the GUI Loop

* root.mainloop() starts the event loop, keeping the GUI active until the user closes the window.

4. Key Features

✅ User-Friendly Input Fields – Allows users to enter required values.  
✅ Dropdown Menu for Algorithm Selection – Simplifies user choices.  
✅ Dynamic Results Display – Shows page faults, hit ratio, and miss ratio.  
✅ Interactive Execution – Runs simulations with a button click.

This GUI provides a clean and intuitive interface for users to experiment with different page replacement algorithms.