

```
import pyautogui import time import cv2 import os # List of required Flowgorithm icons images
= [ "arrow.png", "assign.png", "output.png", "input.png", "for-loop.png", "while-loop.png",
"doloop.png", "call.png", "comment.png", "control-if.png", "declare.png", "file-close.png", "file-
open.png", "file-read.png", "file-write.png", "break-point.png" ] when i show the flowgorithm
tab find the image and click and wait foe 200 milliseconds and double click the image and go to
the text box each 200 milli second by mouse macro understand each flowgorithm input values let
the input be done after i click and show arrow to select each block i scribble on screen and only
if and only if the selected in cmd prompt and when scribbling on screen keyboard select only the
type of image numbered and entered so by just scribbling on screen by animation show
movement of mouse and place only once when selected onscreen create the code
```

ChatGPT said:

Here is a complete `pyautogui`-based Python script that:

1. Waits for you to **manually bring the Flowgorithm window to the front**.
2. **Searches for each Flowgorithm icon** from a list (like `assign.png`, `input.png`, etc.) on the screen.
3. When you **click an icon on the screen**, it:
 - Animates the mouse moving to it.
 - Clicks once.
 - Waits 200 ms.
 - Double-clicks to open the properties box.
4. You **scribble** or write manually in the textbox.
5. Then the script waits for **your keyboard input in the console**, like 1 for "assign", 2 for "output", etc.
6. Only after you type the number and confirm, the next icon is placed and the macro proceeds.

☑ Requirements

1. Install the necessary packages:

```
bash
CopyEdit
pip install pyautogui opencv-python
```

2. Ensure `arrow.png`, `assign.png`, etc., are in the same folder as the script.
-

☐ Python Script: `flow_macro.py`

```
python
CopyEdit
```

```

import pyautogui
import time
import cv2
import os
import keyboard # pip install keyboard

# List of images mapped to numbers for recognition
images = {
    1: "assign.png",
    2: "output.png",
    3: "input.png",
    4: "for-loop.png",
    5: "while-loop.png",
    6: "doloop.png",
    7: "call.png",
    8: "comment.png",
    9: "control-if.png",
    10: "declare.png",
    11: "file-close.png",
    12: "file-open.png",
    13: "file-read.png",
    14: "file-write.png",
    15: "break-point.png",
    16: "arrow.png" # Arrow last so you can "select" after scribbling
}

# Wait to bring Flowgorithm to front
print("⌚ Script starts in 5 seconds... Bring Flowgorithm to front.")
time.sleep(5)

# Main interaction loop
while True:
    print("\n▣ Scribble on screen, then type number (1-16) for icon and press ENTER.")
    print("    [Press 'q' to quit]")

    try:
        user_input = input("Enter block number: ")
        if user_input.strip().lower() == 'q':
            print("▣ Exiting macro.")
            break
        num = int(user_input.strip())

        if num not in images:
            print("▲▣ Invalid number. Try again.")
            continue

        image_file = images[num]
        print(f"▣ Looking for '{image_file}' on screen...")

        # Locate the image on screen
        location = pyautogui.locateCenterOnScreen(image_file, confidence=0.8)
        if location:
            print(f"☑ Found at {location}. Moving mouse.")
            pyautogui.moveTo(location.x, location.y, duration=0.3)
            pyautogui.click() # Single click
            time.sleep(0.2)
    
```

```

        pyautogui.doubleClick() # Double click to open textbox
        print("□□ Double clicked. You can now type in Flowgorithm box.")
    else:
        print("✕ Image not found. Try again or check screen visibility.")

except Exception as e:
    print(f"△□ Error: {e}")

```

□ How to Use It

1. Open Flowgorithm.
 2. Place your script and icons (e.g., `assign.png`, etc.) in one folder.
 3. Run the script.
 4. In the terminal:
 - Scribble or click the desired icon on screen.
 - Enter a number (e.g., 1 for `assign`).
 - Watch mouse auto-click it and open the input dialog.
 5. Input your values manually.
 6. Then go to the next block.
-

☑ To Do for Best Results

- Use **high-quality screenshots** for each icon.
 - Ensure screen scaling is **100%**.
 - Use a **1080p or fixed-size Flowgorithm window** to avoid mismatches.
-

Would you like a version where it shows **animated path trails**, like a dot/line tracing the mouse? I can add that too using visual cues.

You said:

```

import pyautogui import time import cv2 import os import keyboard # pip install keyboard # List
of images mapped to numbers for recognition images = { 1: "assign.png", 2: "output.png", 3:
"input.png", 4: "for-loop.png", 5: "while-loop.png", 6: "doloop.png", 7: "call.png", 8:
"comment.png", 9: "control-if.png", 10: "declare.png", 11: "file-close.png", 12: "file-open.png",
13: "file-read.png", 14: "file-write.png", 15: "break-point.png", 16: "arrow.png" # Arrow last so
you can "select" after scribbling } # Wait to bring Flowgorithm to front print("⌂ Script starts in
5 seconds... Bring Flowgorithm to front.") time.sleep(5) # Main interaction loop while True:
print("\n□ Scribble on screen, then type number (1–16) for icon and press ENTER.") print("
[Press 'q' to quit]") try: user_input = input("Enter block number: ") if user_input.strip().lower()
== 'q': print("□ Exiting macro.") break num = int(user_input.strip()) if num not in images:
print("△□ Invalid number. Try again.") continue image_file = images[num] print(f"□ Looking

```

```

for '{image_file}' on screen...") # Locate the image on screen location =
pyautogui.locateCenterOnScreen(image_file, confidence=0.8) if location: print(f"✅ Found at
{location}. Moving mouse.") pyautogui.moveTo(location.x, location.y, duration=0.3)
pyautogui.click() # Single click time.sleep(0.2) pyautogui.doubleClick() # Double click to open
textbox print("✅✅ Double clicked. You can now type in Flowgorithm box.") else: print("❌
Image not found. Try again or check screen visibility.") except Exception as e: print(f"⚠️ Error:
{e}") give 1 secoond to select after each time

```

ChatGPT said:

To give you **1 second to manually select or observe** after each interaction (especially after the double-click), just add `time.sleep(1)` after the `doubleClick()` line.

Here's your **updated script** with a 1-second pause after each image block interaction:

✅ Modified Script (flow_macro.py)

```

python
CopyEdit
import pyautogui
import time
import cv2
import os
import keyboard # pip install keyboard

# List of images mapped to numbers for recognition
images = {
    1: "assign.png",
    2: "output.png",
    3: "input.png",
    4: "for-loop.png",
    5: "while-loop.png",
    6: "doloop.png",
    7: "call.png",
    8: "comment.png",
    9: "control-if.png",
    10: "declare.png",
    11: "file-close.png",
    12: "file-open.png",
    13: "file-read.png",
    14: "file-write.png",
    15: "break-point.png",
    16: "arrow.png" # Arrow for selection
}

print("⚠️ Script starts in 5 seconds... Bring Flowgorithm to front.")
time.sleep(5)

while True:
    print("\n✅ Scribble on screen, then type number (1-16) for icon and press
ENTER.")

```

```

print("    [Press 'q' to quit]")

try:
    user_input = input("Enter block number: ")
    if user_input.strip().lower() == 'q':
        print("□ Exiting macro.")
        break
    num = int(user_input.strip())

    if num not in images:
        print("△□ Invalid number. Try again.")
        continue

    image_file = images[num]
    print(f"□ Looking for '{image_file}' on screen...")

    location = pyautogui.locateCenterOnScreen(image_file, confidence=0.8)
    if location:
        print(f"☑ Found at {location}. Moving mouse.")
        pyautogui.moveTo(location.x, location.y, duration=0.3)
        pyautogui.click()          # Single click
        time.sleep(0.2)
        pyautogui.doubleClick()    # Double click
        print("□□ Double clicked. You can now type in Flowgorithm box.")

        time.sleep(1) # □ Give 1 second to observe or select manually

    else:
        print("✗ Image not found. Try again or check screen visibility.")

except Exception as e:
    print(f"△□ Error: {e}")

```

☑ What Changed

- `time.sleep(1)` added after:

```

python
CopyEdit
pyautogui.doubleClick()

```

- This gives a short pause for you to click, type, or adjust before continuing.
-

Would you like to:

- Visually animate the mouse trail path?
- Save all actions to a log file?
- Allow block numbers via **on-screen GUI buttons** instead of typing?

Let me know!

You said:

```
import pyautogui import time import cv2 import os import keyboard # pip install keyboard # List
of images mapped to numbers for recognition images = { 1: "assign.png", 2: "output.png", 3:
"input.png", 4: "for-loop.png", 5: "while-loop.png", 6: "doloop.png", 7: "call.png", 8:
"comment.png", 9: "control-if.png", 10: "declare.png", 11: "file-close.png", 12: "file-open.png",
13: "file-read.png", 14: "file-write.png", 15: "break-point.png", 16: "arrow.png" # Arrow for
selection } print("⌚ Script starts in 5 seconds... Bring Flowgorithm to front.") time.sleep(5)
while True: print("\n▢ Scribble on screen, then type number (1–16) for icon and press ENTER.")
print(" [Press 'q' to quit]") try: user_input = input("Enter block number: ") if
user_input.strip().lower() == 'q': print("▢ Exiting macro.") break num = int(user_input.strip()) if
num not in images: print("△▢ Invalid number. Try again.") continue image_file = images[num]
print(f"▢ Looking for '{image_file}' on screen...") location =
pyautogui.locateCenterOnScreen(image_file, confidence=0.8) if location: print(f"✅ Found at
{location}. Moving mouse.") pyautogui.moveTo(location.x, location.y, duration=0.3)
pyautogui.click() # Single click time.sleep(0.2) pyautogui.doubleClick() # Double click
print("▢▢ Double clicked. You can now type in Flowgorithm box.") time.sleep(1) # ▢ Give 1
second to observe or select manually else: print("✗ Image not found. Try again or check screen
visibility.") except Exception as e: print(f"△▢ Error: {e}") i said no time just selection just seach
the screen when i scribble the all over match with the selection of image in the cmd prompt and
then ask for next iunpu and repeat the same rewrite all
```

ChatGPT said:

Got it. You're asking for the following behavior:

▢ Key Logic:

- **You scribble (e.g. move cursor over an icon).**
- The script **scans the entire screen** and matches the current image in the command prompt.
- If that image is found **only once** on the screen, it:
 - Moves to that location.
 - Clicks once.
 - Double-clicks.
 - Immediately waits for **next user input**, without any artificial delay.
- Then continues to the **next block**.

✅ Clean, Updated Script (No Delay, Auto-Match Based on Scribble):

```
python
CopyEdit
import pyautogui
```

```

import time
import os

# Image dictionary: number -> image filename
images = {
    1: "assign.png",
    2: "output.png",
    3: "input.png",
    4: "for-loop.png",
    5: "while-loop.png",
    6: "doloop.png",
    7: "call.png",
    8: "comment.png",
    9: "control-if.png",
    10: "declare.png",
    11: "file-close.png",
    12: "file-open.png",
    13: "file-read.png",
    14: "file-write.png",
    15: "break-point.png",
    16: "arrow.png"
}

print("□ Flowgorithm Macro Started.")
print("□ Bring Flowgorithm to front and get ready.")
time.sleep(3)

while True:
    print("\n□ Scribble on screen, then enter the block number (1-16). [q to quit]")
    user_input = input("Enter block number: ").strip()

    if user_input.lower() == 'q':
        print("□ Macro ended.")
        break

    try:
        num = int(user_input)
        if num not in images:
            print("✗ Invalid block number.")
            continue

        image_file = images[num]
        print(f"□ Scanning screen for: {image_file}...")

        match = pyautogui.locateCenterOnScreen(image_file, confidence=0.8)

        if match:
            print(f"☑ Match found at {match}. Executing macro.")
            pyautogui.moveTo(match.x, match.y, duration=0.2)
            pyautogui.click()
            pyautogui.doubleClick()
        else:
            print("✗ Image not found. Try again by scribbling near the icon.")

    except ValueError:

```

```
print("\n Please enter a valid number between 1-16 or 'q' to quit.")
except Exception as e:
    print(f"\n Error occurred: {e}")
```

□ How to Use:

1. Launch Flowgorithm and **keep it open on screen**.
 2. Place this script and all icons (assign.png, etc.) in the same folder.
 3. Run the script.
 4. **Manually scribble** or hover your mouse over an icon.
 5. Type the block number matching that image.
 6. The script will:
 - Match the image on screen.
 - Auto-move and double-click.
 7. Repeat for next block.
-

Optional Upgrades:

- Highlight matched area (with a red box using OpenCV).
- Log each interaction to a file.
- Allow image-to-number matching using icon selection (GUI version).

Would you like any of these advanced features next?

You said:

i will go through the screen with mouse after selection in cmd prompt after that match with the image and click and place it

```
import pyautogui
import time
import os

# Image dictionary: number
-> image filename
images = { 1: "assign.png", 2: "output.png", 3: "input.png", 4: "for-loop.png",
5: "while-loop.png", 6: "doloop.png", 7: "call.png", 8: "comment.png", 9: "control-if.png", 10:
"declare.png", 11: "file-close.png", 12: "file-open.png", 13: "file-read.png", 14: "file-write.png",
15: "break-point.png", 16: "arrow.png" }

print("\n Flowgorithm Macro Started.")
print("\n Bring Flowgorithm to front and get ready.")
time.sleep(3)

while True:
    print("\n Scribble on screen, then enter the block number (1-16). [q to quit]")
    user_input = input("Enter block number: ").strip()
    if user_input.lower() == 'q':
        print("\n Macro ended.")
        break
    try:
        num = int(user_input)
        if num not in images:
            print("\n ❌ Invalid block number.")
            continue
        image_file = images[num]
        print(f"\n Scanning screen for: {image_file}...")
        match = pyautogui.locateCenterOnScreen(image_file, confidence=0.8)
        if match:
            print(f"\n ✅ Match found at {match}. Executing macro.")
            pyautogui.moveTo(match.x, match.y, duration=0.2)
            pyautogui.click()
            pyautogui.doubleClick()
        else:
            print("\n ❌ Image not found. Try again by scribbling near the icon.")
    except ValueError:
        print("\n Please enter a valid number between 1-16 or 'q' to quit.")
    except Exception as e:
        print(f"\n Error occurred: {e}")
```

ChatGPT said:

Here's the updated script that:

1. Lets you move (or “scribble”) the mouse over the Flowgorithm palette.
2. Reads your block number in the console.
3. Grabs your **current mouse coordinates** and defines a small search region around it.
4. Looks for the icon **only within that region**, then moves, clicks and double-clicks it.

This should dramatically cut down false matches elsewhere on screen.

```
python
CopyEdit
import pyautogui
import time
import os

# Image dictionary: number -> image filename
images = {
    1: "assign.png",
    2: "output.png",
    3: "input.png",
    4: "for-loop.png",
    5: "while-loop.png",
    6: "doloop.png",
    7: "call.png",
    8: "comment.png",
    9: "control-if.png",
    10: "declare.png",
    11: "file-close.png",
    12: "file-open.png",
    13: "file-read.png",
    14: "file-write.png",
    15: "break-point.png",
    16: "arrow.png"
}

def clamp(x, minimum, maximum):
    return max(minimum, min(x, maximum))

print("\n Flowgorithm Macro Started.")
print("\n Bring Flowgorithm to front and get ready.")
time.sleep(3)

screen_w, screen_h = pyautogui.size()

while True:
    print("\n Move your mouse over the desired icon,")
    print(" then enter its block number (1-16), or 'q' to quit.")
    inp = input("Enter block number: ").strip().lower()

    if inp == 'q':
        print("\n Macro ended.")
        break

    try:
        num = int(inp)
```

```

if num not in images:
    print("✗ Invalid block number. Try again.")
    continue

image_file = images[num]
# get current mouse position as center of our search region
mx, my = pyautogui.position()

# define a 300×300px region around cursor (you can adjust size)
region_size = 300
left = clamp(mx - region_size//2, 0, screen_w)
top = clamp(my - region_size//2, 0, screen_h)
width = clamp(region_size, 0, screen_w - left)
height = clamp(region_size, 0, screen_h - top)

print(f"🔍 Searching for '{image_file}' near ({mx},{my})...")
match = pyautogui.locateCenterOnScreen(
    image_file,
    region=(left, top, width, height),
    confidence=0.8
)

if match:
    print(f"✅ Found at {match}. Moving mouse and activating.")
    pyautogui.moveTo(match.x, match.y, duration=0.2)
    pyautogui.click()
    pyautogui.doubleClick()
else:
    print("✗ Didn't find it in that region. Try scribbling closer to
the icon.")

except ValueError:
    print("⚠️ Please enter a number (1-16) or 'q' to quit.")
except Exception as e:
    print(f"⚠️ Error occurred: {e}")

```

Changes in a nutshell:

- After you scribble, we read the **current mouse position**.
- We set a **300×300px bounding box** around it.
- We only run `locateCenterOnScreen(...)` inside that box.
- If found, we move, click, double-click.
- If not, you get a prompt to scribble closer.

Feel free to tweak `region_size` (e.g. 200×200 or 400×400) or the confidence threshold to suit your screen resolution and icon quality.

You said:

```

import pyautogui import time import os # Image dictionary: number -> image filename images =
{ 1: "assign.png", 2: "output.png", 3: "input.png", 4: "for-loop.png", 5: "while-loop.png", 6:
"doloop.png", 7: "call.png", 8: "comment.png", 9: "control-if.png", 10: "declare.png", 11: "file-
close.png", 12: "file-open.png", 13: "file-read.png", 14: "file-write.png", 15: "break-point.png",

```

```

16: "arrow.png" } def clamp(x, minimum, maximum): return max(minimum, min(x, maximum))
print("□ Flowgorithm Macro Started.") print("□ Bring Flowgorithm to front and get ready.")
time.sleep(3) screen_w, screen_h = pyautogui.size() while True: print("\n□ Move your mouse
over the desired icon,") print(" then enter its block number (1–16), or 'q' to quit.") inp =
input("Enter block number: ").strip().lower() if inp == 'q': print("□ Macro ended.") break try:
num = int(inp) if num not in images: print(" ✕Invalid block number. Try again.") continue
image_file = images[num] # get current mouse position as center of our search region mx, my =
pyautogui.position() # define a 300×300px region around cursor (you can adjust size)
region_size = 300 left = clamp(mx - region_size//2, 0, screen_w) top = clamp(my -
region_size//2, 0, screen_h) width = clamp(region_size, 0, screen_w - left) height =
clamp(region_size, 0, screen_h - top) print(f"□ Searching for '{image_file}' near
({mx},{my})...") match = pyautogui.locateCenterOnScreen( image_file, region=(left, top, width,
height), confidence=0.8 ) if match: print(f"☑Found at {match}. Moving mouse and activating.")
pyautogui.moveTo(match.x, match.y, duration=0.2) pyautogui.click() pyautogui.doubleClick()
else: print(" ✕Didn't find it in that region. Try scribbling closer to the icon.") except ValueError:
print("△□ Please enter a number (1–16) or 'q' to quit.") except Exception as e: print(f"△□ Error
occurred: {e}") wait till scribble onscreen with mouse match with the input image selection in
cmd prompt understand correctly its made for special children whow does not have normal
reaction rate externally after rewrite fully

```

ChatGPT said:

Here's a fully rewritten version that's built for learners who may need more time. After you enter the block number, the script will:

1. **Let you move your mouse** (“scribble”) anywhere over the icon—no rush.
2. **Continuously scan** a small region around your cursor until it finds the matching image.
3. As soon as it's found, it will **move, click, and double-click** to open the text box.
4. Then you can take your time typing in Flowgorithm before moving on.

```

python
CopyEdit
import pyautogui
import time
import os

# Map numbers to your Flowgorithm icon filenames
IMAGES = {
    1: "assign.png",
    2: "output.png",
    3: "input.png",
    4: "for-loop.png",
    5: "while-loop.png",
    6: "doloop.png",
    7: "call.png",
    8: "comment.png",
    9: "control-if.png",
    10: "declare.png",
    11: "file-close.png",
    12: "file-open.png",
    13: "file-read.png",

```

```

14: "file-write.png",
15: "break-point.png",
16: "arrow.png"
}

def clamp(val, low, high):
    return max(low, min(val, high))

def wait_for_image_near_cursor(image_path, region_size=300, confidence=0.8):
    """
    Continuously search within a square region around the current mouse
    cursor until the target image is found. Returns its center point.
    """
    screen_w, screen_h = pyautogui.size()

    while True:
        mx, my = pyautogui.position()
        left = clamp(mx - region_size//2, 0, screen_w)
        top = clamp(my - region_size//2, 0, screen_h)
        width = clamp(region_size, 0, screen_w - left)
        height = clamp(region_size, 0, screen_h - top)

        match = pyautogui.locateCenterOnScreen(
            image_path,
            region=(left, top, width, height),
            confidence=confidence
        )
        if match:
            return match
        # Provide gentle feedback so the user knows it's still running
        print(f"    ...still waiting for '{os.path.basename(image_path)}' near ({mx},{my})")
        time.sleep(0.5) # half-second pause before retrying

def main():
    print("\n Flowgorithm Macro (slow-reaction mode) started.")
    print("\n Bring Flowgorithm to front. You have all the time you need.")
    time.sleep(3)

    while True:
        print("\n Move your mouse over the icon you want,")
        print("    then enter its block number (1-16), or 'q' to quit.")
        choice = input("Block number: ").strip().lower()

        if choice == 'q':
            print("\n Ending macro. Goodbye!")
            break

        try:
            num = int(choice)
            if num not in IMAGES:
                print("\n X That number isn't in the list. Please try 1-16.")
                continue

            image_file = IMAGES[num]
            print(f"\n Now waiting to find '{image_file}'. Move your mouse over it...")

```

```

# Block until the image is found near the current cursor position
location = wait_for_image_near_cursor(image_file)

print(f"🔍 Found at {location}! Clicking now.")
pyautogui.moveTo(location.x, location.y, duration=0.3)
pyautogui.click()
time.sleep(0.2)
pyautogui.doubleClick()
print("👉 Double-clicked. You can now type in Flowgorithm.")

# Pause here until the user confirms they're ready to continue
input("Press ENTER when you're ready for the next block...")

except ValueError:
    print("⚠️ Please type a number (1-16) or 'q'.")
except Exception as e:
    print(f"⚠️ Unexpected error: {e}")

if __name__ == "__main__":
    main()

```

What's different?

- **wait_for_image_near_cursor** loops indefinitely—no timeout—until the icon appears under your scribble.
- A gentle “**still waiting...**” message printed every 0.5 s.
- After double-clicking, the script **pauses** (`input(...)`) so you can finish typing without feeling rushed.
- All timings and region sizes can be tweaked to suit your screen or pace.

Let me know if you'd like any more adjustments!

You said:

☐ Flowgorithm Macro (slow-reaction mode) started. ☐ Bring Flowgorithm to front. You have all the time you need. ☐ Move your mouse over the icon you want, then enter its block number (1–16), or 'q' to quit. Block number: 1 ☐ Now waiting to find 'assign.png'. Move your mouse over it... ⚠️ Unexpected error: understand and create just search the screen according to input in cmd prompt [t match with the image pixel after click ask the next block]

```

import pyautogui
import time
import os

# Map numbers to your Flowgorithm icon filenames
IMAGES = { 1: "assign.png", 2: "output.png", 3: "input.png", 4: "for-loop.png", 5: "while-loop.png", 6: "doloop.png", 7: "call.png", 8: "comment.png", 9: "control-if.png", 10: "declare.png", 11: "file-close.png", 12: "file-open.png", 13: "file-read.png", 14: "file-write.png", 15: "break-point.png", 16: "arrow.png" }

def clamp(val, low, high):
    return max(low, min(val, high))

def wait_for_image_near_cursor(image_path, region_size=300, confidence=0.8):
    """ Continuously search within a square region around the current mouse cursor until the target image is found. Returns its center point. """
    screen_w, screen_h = pyautogui.size()
    while True:
        mx, my = pyautogui.position()
        left = clamp(mx - region_size//2, 0, screen_w)
        top = clamp(my - region_size//2, 0, screen_h)
        width = clamp(region_size, 0, screen_w - left)
        height =
  
```

```

clamp(region_size, 0, screen_h - top) match = pyautogui.locateCenterOnScreen( image_path,
region=(left, top, width, height), confidence=confidence ) if match: return match # Provide
gentle feedback so the user knows it's still running print(f" ...still waiting for
'{os.path.basename(image_path)}' near ({mx},{my})") time.sleep(0.5) # half-second pause
before retrying def main(): print("□ Flowgorithm Macro (slow-reaction mode) started.")
print("□ Bring Flowgorithm to front. You have all the time you need.") time.sleep(3) while
True: print("\n□ Move your mouse over the icon you want,") print(" then enter its block number
(1–16), or 'q' to quit.") choice = input("Block number: ").strip().lower() if choice == 'q': print("□
Ending macro. Goodbye!") break try: num = int(choice) if num not in IMAGES: print(" ✕ That
number isn't in the list. Please try 1–16.") continue image_file = IMAGES[num] print(f"□ Now
waiting to find '{image_file}'. Move your mouse over it...") # Block until the image is found
near the current cursor position location = wait_for_image_near_cursor(image_file) print(f" ✓
Found at {location}! Clicking now.") pyautogui.moveTo(location.x, location.y, duration=0.3)
pyautogui.click() time.sleep(0.2) pyautogui.doubleClick() print("□ □ Double-clicked. You can
now type in Flowgorithm.") # Pause here until the user confirms they're ready to continue
input("Press ENTER when you're ready for the next block...") except ValueError: print("△ □
Please type a number (1–16) or 'q'.") except Exception as e: print(f"△ □ Unexpected error: {e}")
if __name__ == "__main__": main() " use open cv for image recogonition with py autogui
rewrite fully

```

ChatGPT said:

A network error occurred. Please check your connection and try again. If this issue persists please contact us through our help center at help.openai.com.

Here's a from-scratch rewrite that uses OpenCV for template-matching instead of locateCenterOnScreen. It:

1. Takes a screenshot with `pyautogui` and converts it to a NumPy array.
2. Uses `cv2.matchTemplate` to find the block icon in a region around your current mouse position.
3. Blocks (loops) until a good match is found.
4. Moves, clicks, double-clicks, then prompts you for the next block.

```

python
CopyEdit
import pyautogui
import cv2
import numpy as np
import time
import os

# Map numbers to your Flowgorithm icon filenames
IMAGES = {
    1: "assign.png",
    2: "output.png",
    3: "input.png",
    4: "for-loop.png",
    5: "while-loop.png",
    6: "doloop.png",

```

```

7: "call.png",
8: "comment.png",
9: "control-if.png",
10: "declare.png",
11: "file-close.png",
12: "file-open.png",
13: "file-read.png",
14: "file-write.png",
15: "break-point.png",
16: "arrow.png"
}

def clamp(val, minimum, maximum):
    return max(minimum, min(val, maximum))

def find_template_in_region(template_path, region, threshold=0.8):
    """
    region: (left, top, width, height)
    Returns center (x,y) if match ≥ threshold, else None.
    """
    screenshot = pyautogui.screenshot(region=region)
    img_rgb = cv2.cvtColor(np.array(screenshot), cv2.COLOR_BGR2RGB)
    template = cv2.imread(template_path, cv2.IMREAD_UNCHANGED)
    if template is None:
        raise FileNotFoundError(f"Template not found: {template_path}")
    w, h = template.shape[1], template.shape[0]

    res = cv2.matchTemplate(img_rgb, template, cv2.TM_CCOEFF_NORMED)
    min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(res)
    if max_val >= threshold:
        # max_loc is top-left of match in the region
        match_x = region[0] + max_loc[0] + w // 2
        match_y = region[1] + max_loc[1] + h // 2
        return (match_x, match_y)
    return None

def wait_for_block(num, region_size=300, threshold=0.8):
    """
    Blocks until the requested image is found near cursor.
    Returns its center coords.
    """
    screen_w, screen_h = pyautogui.size()
    image_path = IMAGES[num]
    print(f"\n🔍 Searching for '{os.path.basename(image_path)}'...")

    while True:
        mx, my = pyautogui.position()
        left = clamp(mx - region_size//2, 0, screen_w)
        top = clamp(my - region_size//2, 0, screen_h)
        width = clamp(region_size, 0, screen_w - left)
        height = clamp(region_size, 0, screen_h - top)

        coords = find_template_in_region(
            image_path,
            region=(left, top, width, height),
            threshold=threshold
        )

```

```

        if coords:
            print(f"📍 Found at {coords}!")
            return coords

        print(f"    ...not yet at ({mx},{my}); moving cursor closer or adjust
window.")
        time.sleep(0.5)

def main():
    print("□ Flowgorithm Macro (OpenCV mode) started.")
    print("□ Bring Flowgorithm to front. Take your time.\n")
    time.sleep(2)

    while True:
        choice = input("Enter block number (1-16), or 'q' to quit:
").strip().lower()
        if choice == 'q':
            print("□ Macro terminated.")
            break

        try:
            num = int(choice)
            if num not in IMAGES:
                print("✗ Invalid number—please choose 1-16.")
                continue

            # Wait until template is found near current mouse cursor
            x, y = wait_for_block(num)

            # Click and double-click
            pyautogui.moveTo(x, y, duration=0.2)
            pyautogui.click()
            time.sleep(0.2)
            pyautogui.doubleClick()
            print("□□ Block placed. Type your text in Flowgorithm now.")

        except ValueError:
            print("△□ Please input a number or 'q'.")
        except Exception as e:
            print(f"△□ Error: {e}")

if __name__ == "__main__":
    main()

```

How this works

- **Screenshot region:** every half-second it grabs a small box around your cursor.
- **Template match:** `cv2.matchTemplate` finds the icon within that box.
- **Threshold:** only accepts matches ≥ 0.8 confidence; you can lower it if needed.
- **Loop:** continues searching until it finds the icon.
- **Clicks:** moves, clicks once, waits 0.2 s, then double-clicks to open the block's text box.
- **Next block:** returns you immediately to the prompt for the next number.

Feel free to adjust `region_size` and `threshold` to suit your display and icon clarity.

A network error occurred. Please check your connection and try again. If this issue persists please contact us through our help center at help.openai.com.

You said:

```
.py □ Flowgorithm Macro (OpenCV mode) started. □ Bring Flowgorithm to front. Take your
time. Enter block number (1–16), or 'q' to quit: 1 □ Searching for 'assign.png'... △□ Error:
OpenCV(4.11.0) D:\a\opencv-python\opencv-
python\opencv\modules\imgproc\src\templmatch.cpp:1164: error: (-215:Assertion failed) (depth
== CV_8U || depth == CV_32F) && type == _templ.type() && _img.dims() <= 2 in function
'cv::matchTemplate' Enter block number (1–16), or 'q' to quit:
```

ChatGPT said:

The OpenCV error means your screenshot/array and the template image have different channel layouts (e.g. one has an alpha channel). Let's fix that by:

1. **Always loading the template as a 3-channel BGR image** (no alpha).
2. **Converting the screenshot** from RGB to BGR before matching.

Here's the corrected version of the key functions—and the full script below:

```
python
CopyEdit
import pyautogui
import cv2
import numpy as np
import time
import os

# Map numbers to your Flowgorithm icon filenames
IMAGES = {
    1: "assign.png",
    2: "output.png",
    3: "input.png",
    4: "for-loop.png",
    5: "while-loop.png",
    6: "doloop.png",
    7: "call.png",
    8: "comment.png",
    9: "control-if.png",
    10: "declare.png",
    11: "file-close.png",
    12: "file-open.png",
    13: "file-read.png",
    14: "file-write.png",
    15: "break-point.png",
    16: "arrow.png"
}

def clamp(val, low, high):
    return max(low, min(val, high))
```

```

def find_template_in_region(template_path, region, threshold=0.8):
    """
    region: (left, top, width, height)
    Returns center (x,y) if match ≥ threshold, else None.
    """
    # 1) screenshot (PIL Image), convert to BGR ndarray
    screenshot = pyautogui.screenshot(region=region)
    img_bgr = cv2.cvtColor(np.array(screenshot), cv2.COLOR_RGB2BGR)

    # 2) load template as BGR (ignore alpha)
    template = cv2.imread(template_path, cv2.IMREAD_COLOR)
    if template is None:
        raise FileNotFoundError(f"Template not found: {template_path}")
    h, w = template.shape[:2]

    # 3) template match
    res = cv2.matchTemplate(img_bgr, template, cv2.TM_CCOEFF_NORMED)
    _, max_val, _, max_loc = cv2.minMaxLoc(res)

    if max_val >= threshold:
        match_x = region[0] + max_loc[0] + w // 2
        match_y = region[1] + max_loc[1] + h // 2
        return (match_x, match_y)
    return None

def wait_for_block(num, region_size=300, threshold=0.8):
    screen_w, screen_h = pyautogui.size()
    image_path = IMAGES[num]
    print(f"\n🔍 Searching for '{os.path.basename(image_path)}'...")

    while True:
        mx, my = pyautogui.position()
        left = clamp(mx - region_size//2, 0, screen_w)
        top = clamp(my - region_size//2, 0, screen_h)
        width = clamp(region_size, 0, screen_w - left)
        height = clamp(region_size, 0, screen_h - top)

        coords = find_template_in_region(
            image_path,
            region=(left, top, width, height),
            threshold=threshold
        )
        if coords:
            print(f"✅ Found at {coords}!")
            return coords

        print(f"❌ ...not yet at ({mx},{my}); move pointer closer.")
        time.sleep(0.5)

def main():
    print("🔧 Flowgorithm Macro (OpenCV mode) started.")
    print("👉 Bring Flowgorithm to front. Take your time.\n")
    time.sleep(2)

    while True:

```

```

        choice = input("Enter block number (1-16), or 'q' to quit:
").strip().lower()
        if choice == 'q':
            print("□ Macro terminated.")
            break

    try:
        num = int(choice)
        if num not in IMAGES:
            print("✗ Invalid number—please choose 1-16.")
            continue

        x, y = wait_for_block(num)
        pyautogui.moveTo(x, y, duration=0.2)
        pyautogui.click()
        time.sleep(0.2)
        pyautogui.doubleClick()
        print("□□ Block placed. Type your text in Flowgorithm now.\n")

    except ValueError:
        print("△□ Please input a number or 'q'.")
    except Exception as e:
        print(f"△□ Error: {e}")

if __name__ == "__main__":
    main()

```

What changed?

- `cv2.imread(..., IMREAD_COLOR)` forces templates to be 3-channel BGR.
- `cv2.cvtColor(..., COLOR_RGB2BGR)` converts screenshots into the same format.

Now `cv2.matchTemplate` will see matching data types and channel counts. Let me know if you hit any other snags!

You said:

```

import pyautogui import cv2 import numpy as np import time import os # Map numbers to your
Flowgorithm icon filenames IMAGES = { 1: "assign.png", 2: "output.png", 3: "input.png", 4:
"for-loop.png", 5: "while-loop.png", 6: "doloop.png", 7: "call.png", 8: "comment.png", 9:
"control-if.png", 10: "declare.png", 11: "file-close.png", 12: "file-open.png", 13: "file-read.png",
14: "file-write.png", 15: "break-point.png", 16: "arrow.png" } def clamp(val, low, high): return
max(low, min(val, high)) def find_template_in_region(template_path, region, threshold=0.8): """
region: (left, top, width, height) Returns center (x,y) if match ≥ threshold, else None. """ # 1)
screenshot (PIL Image), convert to BGR ndarray screenshot =
pyautogui.screenshot(region=region) img_bgr = cv2.cvtColor(np.array(screenshot),
cv2.COLOR_RGB2BGR) # 2) load template as BGR (ignore alpha) template =
cv2.imread(template_path, cv2.IMREAD_COLOR) if template is None: raise
FileNotFoundError(f"Template not found: {template_path}") h, w = template.shape[:2] # 3)
template match res = cv2.matchTemplate(img_bgr, template, cv2.TM_CCOEFF_NORMED) _,
max_val, _, max_loc = cv2.minMaxLoc(res) if max_val >= threshold: match_x = region[0] +

```

```

max_loc[0] + w // 2 match_y = region[1] + max_loc[1] + h // 2 return (match_x, match_y) return
None def wait_for_block(num, region_size=300, threshold=0.8): screen_w, screen_h =
pyautogui.size() image_path = IMAGES[num] print(f"\n□ Searching for
'{os.path.basename(image_path)}'...") while True: mx, my = pyautogui.position() left =
clamp(mx - region_size//2, 0, screen_w) top = clamp(my - region_size//2, 0, screen_h) width =
clamp(region_size, 0, screen_w - left) height = clamp(region_size, 0, screen_h - top) coords =
find_template_in_region( image_path, region=(left, top, width, height), threshold=threshold ) if
coords: print(f"✅ Found at {coords}!") return coords print(f" ...not yet at ({mx},{my}); move
pointer closer.") time.sleep(0.5) def main(): print("□ Flowgorithm Macro (OpenCV mode)
started.") print("□ Bring Flowgorithm to front. Take your time.\n") time.sleep(2) while True:
choice = input("Enter block number (1–16), or 'q' to quit: ").strip().lower() if choice == 'q':
print("□ Macro terminated.") break try: num = int(choice) if num not in IMAGES: print("✗
Invalid number—please choose 1–16.") continue x, y = wait_for_block(num)
pyautogui.moveTo(x, y, duration=0.2) pyautogui.click() time.sleep(0.2) pyautogui.doubleClick()
print("□□ Block placed. Type your text in Flowgorithm now.\n") except ValueError: print("△□
Please input a number or 'q'.") except Exception as e: print(f"△□ Error: {e}") if __name__ ==
"__main__": main() i told i will scribble onscreen you must find the template image and match
and click it according to user in put inn cmd prompt understand for right cause rewrite fully

```

ChatGPT said:

```

python
CopyEdit
import pyautogui
import cv2
import numpy as np
import time
import os

# Map numbers to Flowgorithm icon filenames
IMAGES = {
    1: "assign.png",
    2: "output.png",
    3: "input.png",
    4: "for-loop.png",
    5: "while-loop.png",
    6: "doloop.png",
    7: "call.png",
    8: "comment.png",
    9: "control-if.png",
    10: "declare.png",
    11: "file-close.png",
    12: "file-open.png",
    13: "file-read.png",
    14: "file-write.png",
    15: "break-point.png",
    16: "arrow.png"
}

def clamp(v, lo, hi):
    return max(lo, min(v, hi))

def screenshot_region(region):

```

```

        """Take a screenshot of the given region and return as BGR numpy
        array."""
        img = pyautogui.screenshot(region=region)
        return cv2.cvtColor(np.array(img), cv2.COLOR_RGB2BGR)

def load_template(path):
    """Load the template as a 3-channel BGR image."""
    tpl = cv2.imread(path, cv2.IMREAD_COLOR)
    if tpl is None:
        raise FileNotFoundError(f"Template not found: {path}")
    return tpl

def find_in_region(template, region, threshold=0.8):
    """
    Template-match within region.
    Returns center-coords of best match if ≥ threshold, else None.
    """
    img = screenshot_region(region)
    res = cv2.matchTemplate(img, template, cv2.TM_CCOEFF_NORMED)
    _, max_val, _, max_loc = cv2.minMaxLoc(res)
    if max_val >= threshold:
        x0, y0 = max_loc
        h, w = template.shape[:2]
        # convert back to screen coords
        return (region[0] + x0 + w//2, region[1] + y0 + h//2)
    return None

def wait_and_click(template, region_size=300, threshold=0.8):
    """
    Loop until template is found near cursor, then click & double-click it.
    """
    sw, sh = pyautogui.size()
    while True:
        mx, my = pyautogui.position()
        left = clamp(mx - region_size//2, 0, sw)
        top = clamp(my - region_size//2, 0, sh)
        width = clamp(region_size, 0, sw - left)
        height = clamp(region_size, 0, sh - top)
        region = (left, top, width, height)

        loc = find_in_region(template, region, threshold)
        if loc:
            pyautogui.moveTo(*loc, duration=0.2)
            pyautogui.click()
            time.sleep(0.2)
            pyautogui.doubleClick()
            return
        # waiting feedback
        print(f"...searching near ({mx},{my})")
        time.sleep(0.5)

def main():
    print("□ Flowgorithm Macro Started. Bring Flowgorithm to front.")
    time.sleep(2)

    # pre-load all templates

```

```

templates = {k: load_template(v) for k, v in IMAGES.items()}

while True:
    choice = input("Enter block number (1-16) or 'q' to quit:
").strip().lower()
    if choice == 'q':
        break
    if not choice.isdigit() or int(choice) not in templates:
        print("Invalid choice. Please enter 1-16.")
        continue

    num = int(choice)
    print(f"Move mouse over your scribble for '{IMAGES[num]}', waiting
for match...")
    wait_and_click(templates[num])
    print("👍 Block placed. Type your text in Flowgorithm.\n")

if __name__ == "__main__":
    main()

```

How it works:

1. **You type** the block number in the console.
2. **You “scribble”** or hover your mouse roughly over that icon in Flowgorithm.
3. The script continuously grabs a small box around your cursor, runs OpenCV template matching, and as soon as it finds your chosen icon it **moves, clicks once**, then **double-clicks** to open the properties.
4. You type your values in Flowgorithm at your own pace.

Feel free to adjust `region_size` or the `match threshold` to match your screen resolution and image quality.

You said:

C:\Users\RITS\Desktop\flowchart-ai>python python-for-special.py ☐ Flowgorithm Macro Started. Bring Flowgorithm to front. [WARN:0@2.340] global loadsave.cpp:268 cv::findDecoder imread_('arrow.png'): can't open/read file: check file path/integrity Traceback (most recent call last): File "C:\Users\RITS\Desktop\flowchart-ai\python-for-special.py", line 102, in <module> main() ~~~^^ File "C:\Users\RITS\Desktop\flowchart-ai\python-for-special.py", line 86, in main templates = {k: load_template(v) for k, v in IMAGES.items()} ~~~~~^^ File "C:\Users\RITS\Desktop\flowchart-ai\python-for-special.py", line 39, in load_template raise FileNotFoundError(f"Template not found: {path}") FileNotFoundError: Template not found: arrow.png i told let me scribbble after you clock the mouse and click rewrite it correctly

ChatGPT said:

Thought for a couple of seconds

Here's a version that:

1. **Checks** which icon files actually exist up front and removes any missing ones (so you won't get a crash on `arrow.png` if it's not there).
2. Lets you “**scribble**” (move your mouse) **after** each click—just hit ENTER in the console when you've positioned your cursor over the next icon.
3. **Loops** until you explicitly quit.

```
python
CopyEdit
import pyautogui
import cv2
import numpy as np
import time
import os

# Map numbers to Flowgorithm icon filenames
IMAGES = {
    1: "assign.png",
    2: "output.png",
    3: "input.png",
    4: "for-loop.png",
    5: "while-loop.png",
    6: "doloop.png",
    7: "call.png",
    8: "comment.png",
    9: "control-if.png",
    10: "declare.png",
    11: "file-close.png",
    12: "file-open.png",
    13: "file-read.png",
    14: "file-write.png",
    15: "break-point.png",
    16: "arrow.png"
}

def clamp(v, lo, hi):
    return max(lo, min(v, hi))

def screenshot_region(region):
    """Take a screenshot of the given region and return as BGR numpy
    array."""
    img = pyautogui.screenshot(region=region)
    return cv2.cvtColor(np.array(img), cv2.COLOR_RGB2BGR)

def load_template(path):
    """Load the template as a 3-channel BGR image."""
    tpl = cv2.imread(path, cv2.IMREAD_COLOR)
    return tpl # may be None if file not found

def find_in_region(template, region, threshold=0.8):
    """Template-match within region. Return center-coords if ≥ threshold, else
    None."""
    img = screenshot_region(region)
    res = cv2.matchTemplate(img, template, cv2.TM_CCOEFF_NORMED)
    _, max_val, _, max_loc = cv2.minMaxLoc(res)
    if max_val >= threshold:
```

```

        x0, y0 = max_loc
        h, w = template.shape[:2]
        return (region[0] + x0 + w//2, region[1] + y0 + h//2)
    return None

def wait_and_click(template, region_size=300, threshold=0.8):
    """Loop until template is found near cursor, then click & double-click
    it."""
    sw, sh = pyautogui.size()
    print("☐ Move your mouse ("scribble") over the icon now...")
    input(" Press ENTER when ready to search→")
    while True:
        mx, my = pyautogui.position()
        left = clamp(mx - region_size//2, 0, sw)
        top = clamp(my - region_size//2, 0, sh)
        width = clamp(region_size, 0, sw - left)
        height = clamp(region_size, 0, sh - top)
        match = find_in_region(template, (left, top, width, height),
threshold)
        if match:
            pyautogui.moveTo(*match, duration=0.2)
            pyautogui.click()
            time.sleep(0.2)
            pyautogui.doubleClick()
            print(f"☑ Clicked at {match}")
            return
        print(f" ...still searching near ({mx},{my})")
        time.sleep(0.5)

def main():
    print("☐ Flowgorithm Macro Started – special-needs mode.")
    print("☐ Bring Flowgorithm to front and have your icons ready.\n")
    time.sleep(2)

    # Preload templates, drop any missing files
    templates = {}
    for num, fname in IMAGES.items():
        tpl = load_template(fname)
        if tpl is None:
            print(f"⚠ Warning: '{fname}' not found—skipping block #{num}.")
        else:
            templates[num] = tpl

    if not templates:
        print("✗ No templates loaded. Please place your PNGs in this folder
and retry.")
        return

    while True:
        choice = input("\nEnter block number (available: "
            + ", ".join(map(str, templates.keys()))
            + "), or 'q' to quit: ").strip().lower()
        if choice == 'q':
            print("☐ Macro terminated. Goodbye!")
            break
        if not choice.isdigit() or int(choice) not in templates:

```



```

        print("✗ Invalid choice—try again.")
        continue

    num = int(choice)
    print(f"\n□ Now searching for '{IMAGES[num]}'...")
    wait_and_click(templates[num])

if __name__ == "__main__":
    main()

```

How this addresses your points

- **Missing files** (like your `arrow.png`) are simply skipped with a warning—no crash.
- **After each block**, you get to reposition (“scribble”) the cursor over the next icon and hit ENTER to start matching.
- The script then **continuously scans** the area around your mouse until it finds the correct template, clicks and double-clicks it, and returns you to the prompt.

Let me know if you need any further tweaks!

You said:

Press ENTER when ready to search→ quickly match template with good accuracy

```

import pyautogui
import cv2
import numpy as np
import time
import os

# Map numbers to Flowgorithm icon filenames
IMAGES = { 1: "assign.png", 2: "output.png", 3: "input.png", 4: "for-loop.png",
5: "while-loop.png", 6: "doloop.png", 7: "call.png", 8: "comment.png", 9: "control-if.png", 10:
"declare.png", 11: "file-close.png", 12: "file-open.png", 13: "file-read.png", 14: "file-write.png",
15: "break-point.png", 16: "arrow.png" }

def clamp(v, lo, hi):
    return max(lo, min(v, hi))

def screenshot_region(region):
    """Take a screenshot of the given region and return as BGR numpy array."""
    img = pyautogui.screenshot(region=region)
    return cv2.cvtColor(np.array(img), cv2.COLOR_RGB2BGR)

def load_template(path):
    """Load the template as a 3-channel BGR image."""
    tpl = cv2.imread(path, cv2.IMREAD_COLOR)
    return tpl # may be None if file not found

def find_in_region(template, region, threshold=0.8):
    """Template-match within region. Return center-coords if ≥ threshold, else None."""
    img = screenshot_region(region)
    res = cv2.matchTemplate(img, template, cv2.TM_CCOEFF_NORMED)
    _, max_val, _, max_loc = cv2.minMaxLoc(res)
    if max_val >= threshold:
        x0, y0 = max_loc
        h, w = template.shape[:2]
        return (region[0] + x0 + w//2, region[1] + y0 + h//2)
    return None

def wait_and_click(template, region_size=300, threshold=0.8):
    """Loop until template is found near cursor, then click & double-click it."""
    sw, sh = pyautogui.size()
    print("□ Move your mouse (“scribble”) over the icon now...")
    input("Press ENTER when ready to search→")
    while True:
        mx, my = pyautogui.position()
        left = clamp(mx - region_size//2, 0, sw)
        top = clamp(my - region_size//2, 0, sh)
        width = clamp(region_size, 0, sw - left)
        height = clamp(region_size, 0, sh - top)
        match = find_in_region(template, (left, top, width, height), threshold)
        if match:
            pyautogui.moveTo(*match, duration=0.2)
            pyautogui.click()
            time.sleep(0.2)
            pyautogui.doubleClick()
            print(f"☑ Clicked at {match}")
            return print(f"...still searching near ({mx}, {my})")
            time.sleep(0.5)

def main():
    print("□ Flowgorithm Macro Started — special-needs mode.")
    print("□ Bring Flowgorithm to front and have your icons ready.\n")
    time.sleep(2) # Preload templates, drop any missing files
    templates = {}
    for num, fname in

```

```

IMAGES.items(): tpl = load_template(fname) if tpl is None: print(f'⚠ Warning: '{fname}' not
found—skipping block #{num}.') else: templates[num] = tpl if not templates: print(" ✖No
templates loaded. Please place your PNGs in this folder and retry.") return while True: choice =
input("\nEnter block number (available: " + ", ".join(map(str, templates.keys())) + "), or 'q' to
quit: ").strip().lower() if choice == 'q': print("☐ Macro terminated. Goodbye!") break if not
choice.isdigit() or int(choice) not in templates: print(" ✖Invalid choice—try again.") continue
num = int(choice) print(f"\n☐ Now searching for '{IMAGES[num]}''...")
wait_and_click(templates[num]) if __name__ == "__main__": main() "

```

ChatGPT said:

A network error occurred. Please check your connection and try again. If this issue persists please contact us through our help center at help.openai.com.

Retry

Here's a fully-rewritten, high-accuracy & high-speed version:

- **No “Press ENTER”** step—after you type the block number, it immediately begins searching.
- Converts both screenshot and template to **grayscale** for faster, more robust matching.
- Uses a **higher threshold (0.9)** by default for fewer false positives.
- You still “scribble” (hover) over the icon—matching begins right away.
- Prints live feedback every 0.3 s, so you know it's working.

```

python
CopyEdit
import pyautogui
import cv2
import numpy as np
import time
import os

# Map numbers to your Flowgorithm icon filenames
IMAGES = {
    1: "assign.png",
    2: "output.png",
    3: "input.png",
    4: "for-loop.png",
    5: "while-loop.png",
    6: "doloop.png",
    7: "call.png",
    8: "comment.png",
    9: "control-if.png",
    10: "declare.png",
    11: "file-close.png",
    12: "file-open.png",
    13: "file-read.png",
    14: "file-write.png",
    15: "break-point.png",
    16: "arrow.png"
}

```

```

def clamp(v, lo, hi):
    return max(lo, min(v, hi))

def load_gray_template(path):
    tpl = cv2.imread(path, cv2.IMREAD_GRAYSCALE)
    if tpl is None:
        raise FileNotFoundError(f"Template not found: {path}")
    return tpl

def grab_gray_region(region):
    """Screenshot region, return grayscale numpy array."""
    img = pyautogui.screenshot(region=region)
    arr = np.array(img)
    # Convert from RGB to grayscale
    return cv2.cvtColor(arr, cv2.COLOR_RGB2GRAY)

def find_template(template, region, threshold):
    """
    Grayscale template-match in region.
    Returns center (x,y) if match ≥ threshold.
    """
    hay = grab_gray_region(region)
    res = cv2.matchTemplate(hay, template, cv2.TM_CCOEFF_NORMED)
    _, max_val, _, max_loc = cv2.minMaxLoc(res)
    if max_val >= threshold:
        ty, tx = max_loc
        h, w = template.shape
        return (region[0] + tx + w//2, region[1] + ty + h//2)
    return None

def wait_and_click(template, threshold=0.9, region_size=200):
    sw, sh = pyautogui.size()
    print("☐ Hover over the icon now...")
    start = time.time()
    while True:
        mx, my = pyautogui.position()
        l = clamp(mx - region_size//2, 0, sw)
        t = clamp(my - region_size//2, 0, sh)
        w = clamp(region_size, 0, sw - l)
        h = clamp(region_size, 0, sh - t)
        region = (l, t, w, h)

        match = find_template(template, region, threshold)
        if match:
            pyautogui.moveTo(*match, duration=0.1)
            pyautogui.click()
            time.sleep(0.1)
            pyautogui.doubleClick()
            print(f"☑ Matched & clicked at {match} (search took {time.time()-start:.2f}s)")
            return
        # Feedback every 0.3s
        if time.time() - start > 0 and int((time.time()-start)*10) % 3 == 0:
            print(f"...searching near ({mx},{my})")
            time.sleep(0.05)

```

```

def main():
    print("□ Flowgorithm Macro - fast, grayscale template-match mode.")
    print("□ Bring Flowgorithm to front. Type a block number when ready.\n")
    time.sleep(1)

    # Preload grayscale templates
    templates = {}
    for num, fname in IMAGES.items():
        if os.path.isfile(fname):
            templates[num] = load_gray_template(fname)
        else:
            print(f"△□ Missing '{fname}', skipping #{num}")

    if not templates:
        print("✕ No templates loaded. Place your PNGs in this folder first.")
        return

    while True:
        choice = input("Enter block number (1-16), or 'q' to quit:
").strip().lower()
        if choice == 'q':
            break
        if not choice.isdigit() or int(choice) not in templates:
            print("✕ Invalid. Choose from:", ", ".join(map(str,
templates.keys())))
            continue

        num = int(choice)
        print(f"\n□ Searching for '{IMAGES[num]}' (threshold=0.9)...")
        wait_and_click(templates[num])

    print("□ Macro ended. Goodbye!")

if __name__ == "__main__":
    main()

```

Highlights:

- **Grayscale matching** is faster and more consistent across small color variations.
- **Threshold 0.9** reduces false hits—bump it down to 0.85 if you miss valid matches.
- **Region size 200×200 px** around your mouse—adjust smaller for speed or larger for reliability.
- **Immediate start** of matching when you hover, no extra ENTER step.
- Live feedback so you know it's actively looking.

Give it a spin and let me know if you need any other tweaks!