

## lab2a

```
lab2a.py > ...
1  # Add comments before you do anything else.
2
3  #!/usr/bin/env python3
4  # Author: sbastani1
5  # Date: 2025-09-25
6  # Purpose: Create a variable, check its type and use a condition to check the value of the variable.
7  # Usage: ./lab2a.py
8
9  # Step 1: Get input from user
10 x = input("Please enter a number: ")
11
12 # Step 2: Check the type of x (it will be string)
13 print("The type of x is:", type(x))
14
15 # Step 3: Convert x to integer
16 x = int(x)
17
18 # Step 4: First if statement - check if x >= 6
19 if x >= 6:
20     print("x is greater than or equal to 6!")
21
22 # Step 5: Second if statement using both relational and boolean operators
23 if x >= 4 and x < 12:
24     print("x is between 4 and 11 inclusive!")
25

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS 1

/home/codespace/.python/current/bin/python3 /workspaces/lab-2-soroush-bastani/lab2a.py
● @Soroush-Bastani →/workspaces/lab-2-soroush-bastani (main) $ /home/codespace/.python/current/bin/python3 /workspaces/lab-2-soroush-bastani/lab2a.py
Please enter a number: 12
The type of x is: <class 'str'>
x is greater than or equal to 6!
● @Soroush-Bastani →/workspaces/lab-2-soroush-bastani (main) $ python ./lab2a.py
Please enter a number: 5
The type of x is: <class 'str'>
x is between 4 and 11 inclusive!
○ @Soroush-Bastani →/workspaces/lab-2-soroush-bastani (main) $ 
```

## lab2b

```
lab2b.py > ...
1  # Add comments before you do anything else.
2
3  #!/usr/bin/env python3
4  # Author: sbastani1
5  # Date: 2025-09-25
6  # Purpose: Practice using if and else statements.
7  # Usage: ./lab2b.py
8
9  # TO DO 1:
10 # Follow the instructions given in the README.md file.
11
12 # Get a 4-digit number from user
13 num = input("Enter a 4-digit number: ")
14
15 # Check if it equals 1984
16 if num == "1984":
17     print("George Orwell")
18 else:
19     print("Not quite right!")

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS 1

● @Soroush-Bastani →/workspaces/lab-2-soroush-bastani (main) $ python ./lab2b.py
Enter a 4-digit number: 1234
Not quite right!
● @Soroush-Bastani →/workspaces/lab-2-soroush-bastani (main) $ python ./lab2b.py
Enter a 4-digit number: 1984
George Orwell
○ @Soroush-Bastani →/workspaces/lab-2-soroush-bastani (main) $ 
```

## lab2c

The screenshot shows a code editor with a Python script named `lab2c.py`. The script prompts the user for two sentences, compares their lengths using `len()`, and prints the result. The terminal below shows the execution of the script and its output.

```
lab2c.py > ...
1 # Add comments before you do anything else.
2
3 #!/usr/bin/env python3
4 # Author: sbastanil
5 # Date: 2025-09-25
6 # Purpose: Practice using if, elif, and else statements.
7 # Usage: ./lab2c.py
8
9 # Prompt the user to enter a sentence, save it in the variable str1
10 str1 = input("Enter a sentence: ")
11
12 # Prompt the user to enter another sentence, save it in the variable str2
13 str2 = input("Enter another sentence: ")
14
15 # Use if, elif, and else statements with the len() function to check which of the 2 is longer
16 if len(str1) > len(str2):
17     print("str1 is longer than str2!")
18 elif len(str2) > len(str1):
19     print("str2 is longer than str1!")
20 else:
21     print("str1 and str2 are of equal length!")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS 1

```
@Soroush-Bastani → /workspaces/lab-2-soroush-bastani (main) $ python ./lab2c.py
Enter a sentence: hey love
Enter another sentence: hey love
str1 and str2 are of equal length!
@Soroush-Bastani → /workspaces/lab-2-soroush-bastani (main) $
```

## lab2d

The screenshot shows a code editor with a Python script named `lab2d.py`. The script explores the `sys` module, printing various command-line arguments and system information. The terminal shows the execution of the script and its output.

```
lab2d.py > ...
1 # Add comments before you do anything else.
2
3 #!/usr/bin/env python3
4 # Author: sbastanil
5 # Date: Learn how to use command line arguments.
6 # Purpose: .
7 # Usage: ./lab2d.py
8
9 import sys
10
11 # TO DO 1: Basic sys module exploration
12 print(sys.version) # prints the version of python currently in use
13 print(sys.platform) # prints the name of operating system
14 print(sys.argv) # prints the list of all arguments given at command line
15 print(len(sys.argv)) # displays the number of command-line arguments
16
17 # TO DO 2: Accessing individual arguments (will only work with arguments!)
18 print(sys.argv[0]) # prints the first argument (always script name)
19 print(sys.argv[1]) # prints the second argument
20 print(sys.argv[2]) # prints the third argument
21 print(len(sys.argv)) # tells us the number of command line arguments
```

TERMINAL

```
@Soroush-Bastani → /workspaces/lab-2-soroush-bastani (main) $ python ./lab2d.py
3.12.1 (main, Jul 10 2025, 11:57:50) [GCC 13.3.0]
linux
['./lab2d.py']
1
./lab2d.py
Traceback (most recent call last):
File "/workspaces/lab-2-soroush-bastani./lab2d.py", line 19, in <module>
    print(sys.argv[2]) # prints the second argument
IndexError: list index out of range
@Soroush-Bastani → /workspaces/lab-2-soroush-bastani (main) $ python ./lab2d.py Maija Maija
3.12.1 (main, Jul 10 2025, 11:57:50) [GCC 13.3.0]
linux
 ['./lab2d.py', 'Maija', 'Maija']
3
./lab2d.py
Maija
Maija
3
@Soroush-Bastani → /workspaces/lab-2-soroush-bastani (main) $
```

## lab2e

The screenshot shows a terminal window with several entries:

- Hi Maija, you are 20 years old and the script received exactly 2 arguments!
- Hi Maija, you are 20 years old and the script received 3 arguments.
- Hi Maija, you are 20 years old and the script received 4 arguments.
- The script requires at least 2 arguments.

```
[Preview] README.md | lab2a.py M | lab2b.py M | lab2c.py M | lab2d.py M | lab2e.py M | lab2f.py M | D v T ...  
lab2e.py > ...  
1 # Add comments before you do anything else.  
2  
3 #!/usr/bin/env python3  
4 # Author: sbastani1  
5 # Date: 2025-09-25  
6 # Purpose: Learn how to use command line arguments.  
7 # Usage: ./lab2e.py  
8  
9 # TO DO 1: Follow the instructions given in README.md file  
10  
11 import sys  
12  
13 # Check if we have at least 2 arguments (plus script name = 3 total)  
14 if len(sys.argv) < 3:  
15     print("The script requires atleast 2 arguments.")  
16 else:  
17     # Create variables for name and age from command line arguments  
18     name = sys.argv[1]  
19     age = sys.argv[2]  
20  
21     # Calculate number of actual arguments (excluding script name)  
22     num_arguments = len(sys.argv) - 1  
23  
24     # Use if-elif structure to handle different cases  
25     if len(sys.argv) == 3:  
26         # Exactly 2 arguments provided  
27         print(f"Hi {name}, you are {age} years old and the script received exactly 2 arguments!")  
28     else:  
29         # More than 2 arguments provided  
30         print(f"Hi {name}, you are {age} years old and the script received {num_arguments} arguments.")
```

## lab2f

The screenshot shows a terminal window with the following interaction:

- Enter your income: 99000
- Enter your marital status (single or married): single
- Tax calculation: \$3,200 + (\$99,000.00 - \$32,000) x 25%
- Tax = \$3,200 + \$67,000.00 x 25% = \$19,950.00
- Your total tax owed: \$19,950.00

```
[Preview] README.md | lab2a.py M | lab2b.py M | lab2c.py M | lab2d.py M | lab2e.py M | lab2f.py M | D v T ...  
lab2f.py > ...  
1 # Add comments before you do anything else.  
2  
3 #!/usr/bin/env python3  
4 # Author: sbastani1  
5 # Date: 2025-09-25  
6 # Purpose: Learn how and practice using nested if, elif, and else statements..  
7 # Usage: ./lab2f.py  
8  
9 # TO DO 1: create the required variables and get thier values form users, convert the variables to int.  
10 income = float(input("Enter your income: "))  
11 status = input("Enter your marital status (single or married): ").lower()  
12 # TO DO 2: write nested conditional statements to calcualte tax  
13 if status == "single":  
14     # First level: Check if status is single  
15     if income < 32000:  
16         # Second level: Low income for single person  
17         tax = income * 0.10  
18         print(f"Tax calculation: ${income:.2f} x 10% = ${tax:.2f}")  
19     else:  
20         # Second level: High income for single person  
21         base_tax = 3200  
22         excess = income - 32000  
23         additional_tax = excess * 0.25  
24         tax = base_tax + additional_tax  
25         print(f"Tax calculation: ${3200 + (income - 32000) * 0.25:.2f}")  
26         print(f"Tax = ${3200 + (excess * 0.25)} = ${tax:.2f}")  
27 elif status == "married":  
28     # First level: Check if status is married  
29     if income < 64000:  
30         # Second level: Low income for married person  
31         tax = income * 0.10  
32         print(f"Tax calculation: ${income:.2f} x 10% = ${tax:.2f}")  
33     else:  
34         # Second level: High income for married person  
35         base_tax = 6400  
36         excess = income - 64000  
37         additional_tax = excess * 0.25  
38         tax = base_tax + additional_tax  
39         print(f"Tax calculation: ${6400 + (income - 64000) * 0.25:.2f}")  
40         print(f"Tax = ${6400 + (excess * 0.25)} = ${tax:.2f}")  
41 else:  
42     # Handle invalid status input  
43     print("Error: Please enter 'single' or 'married' for marital status.")  
44     tax = 0  
45 # Display final result  
46 if tax > 0:  
47     print(f"\nYour total tax owed: ${tax:.2f}")
```

## lab2g

The screenshot shows a terminal window with several tabs at the top, all labeled 'M'. The active tab is 'lab2g.py M'. The code in the editor is as follows:

```
50     print(count)
51     count = count + 1
52 print("Loop has ended")
53 print()
54
55 # Event-driven loop example (guessing)
56 print("== Event-Driven Loop Example ==")
57 guess = 5
58 number = int(input("Guess what number less than 10 I am thinking of? "))
59 while number != guess: # loop condition
60     print("incorrect guess, try again...")
61     number = int(input("Guess what number less than 10 I am thinking of? ")) # keep taking input from user
62
63 print("You got it right!") # this statement will be executed when loop has terminated
64
65 # Observations and learning
66 print("\n==== What I Learned ===")
67 print("1. Initial value of loop variable matters")
68 print("2. Condition operator (!=, <, >) changes how many times loop runs")
69 print("3. Off-by-one errors happen when loop runs wrong number of times")
70 print("4. Counter-driven: We know how many iterations")
71 print("5. Event-driven: We don't know - depends on user input")
```

The terminal output shows three experiments:

- Experiment 1: count = 1
- Experiment 2: count < 5
- Experiment 3: count <= 5

Each experiment prints numbers 0 through 4, followed by the message "Loop has ended".

## lab2h

The screenshot shows a terminal window with several tabs at the top, all labeled 'M'. The active tab is 'lab2h.py M'. The code in the editor is as follows:

```
17 # Add a while loop to create program that wont end until the user enters 1234.
18 # Follow the specific instructions given in the README.md file.
19
20 # Keep asking until user enters correct PIN
21 while pin != "1234":
22     # Check if PIN is exactly 4 digits and all characters are numeric
23     if len(pin) != 4 or not pin.isdigit():
24         print("Incorrect...enter a 4 digit number")
25     else:
26         # PIN is 4 digits, but wrong PIN
27         print("Incorrect...try again")
28
29     # Add blank line for formatting
30     print()
31
32     # Ask for PIN again
33     pin = input("Please type in your PIN: ")
34
35 # When loop ends, user entered correct PIN
36 print("Correct PIN, You can enter!")
```

The terminal output shows two interactions:

- The first interaction asks for a PIN. The user types "9999" and receives the message "Incorrect...try again".
- The second interaction asks for a PIN. The user types "1234" and receives the message "Correct PIN, You can enter!".

## lab2i

The screenshot shows a Jupyter Notebook interface with several tabs at the top: lab2c.py M, lab2d.py M, lab2epy M, lab2f.py M, lab2g.py M, lab2h.py M, lab2ipy M, and lab2j.py M. The lab2i.py tab is active, displaying Python code. The code includes comments explaining the purpose of the script, imports, and a while loop that handles user input for calculating square roots and exiting the loop.

**TERMINAL**

```
@Soroush-Bastani ➔ /workspaces/lab-2-soroush-bastani (main) $ python ./lab2i.py
Please type in a number: 2121
46.054315786782
Please type in a number: 985
31.484512552093
Please type in a number: 69
8.306623826218075
Please type in a number: 96
9.70958971132712
Please type in a number: 1234
35.12833614050059
Please type in a number: -886
Invalid number.
Please type in a number: 0
Exiting ...
@Soroush-Bastani ➔ /workspaces/lab-2-soroush-bastani (main) $
```

## lab2j

The screenshot shows a Jupyter Notebook interface with several tabs at the top: lab2d.py M, lab2epy M, lab2f.py M, lab2g.py M, lab2h.py M, lab2ipy M, lab2j.py M, and lab2j.ipynb M. The lab2j.py tab is active, displaying Python code. The code follows instructions from a README.md file to calculate the sum of even numbers from 1 to 100.

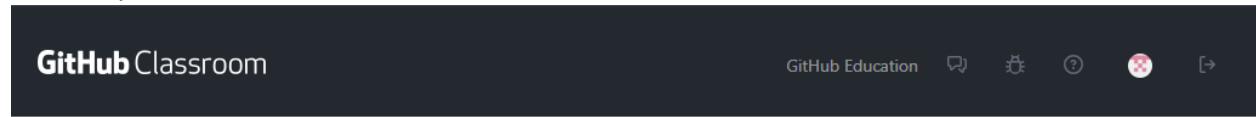
**TERMINAL**

```
@Soroush-Bastani ➔ /workspaces/lab-2-soroush-bastani (main) $ python ./lab2j.py
Sum of even numbers from 1 to 100: 2550
@Soroush-Bastani ➔ /workspaces/lab-2-soroush-bastani (main) $
```

## Note:

When I log in with my personal Github Soroush-Bastani, I'm able to accept this lab.

Hence my username is Soroush-Bastani



The screenshot shows the GitHub Classroom header. It features the GitHub Classroom logo on the left, followed by "GitHub Education" and several icons: a gear, a question mark, a user profile, and a link icon.



You're ready to go —  
Soroush-Bastani

You accepted the assignment, **Lab 2**.

Your team's assignment repository has been created:

 <https://github.com/TR-Seneca-Python/lab-2-soroush-bastani>

We've configured the repository associated with this assignment.



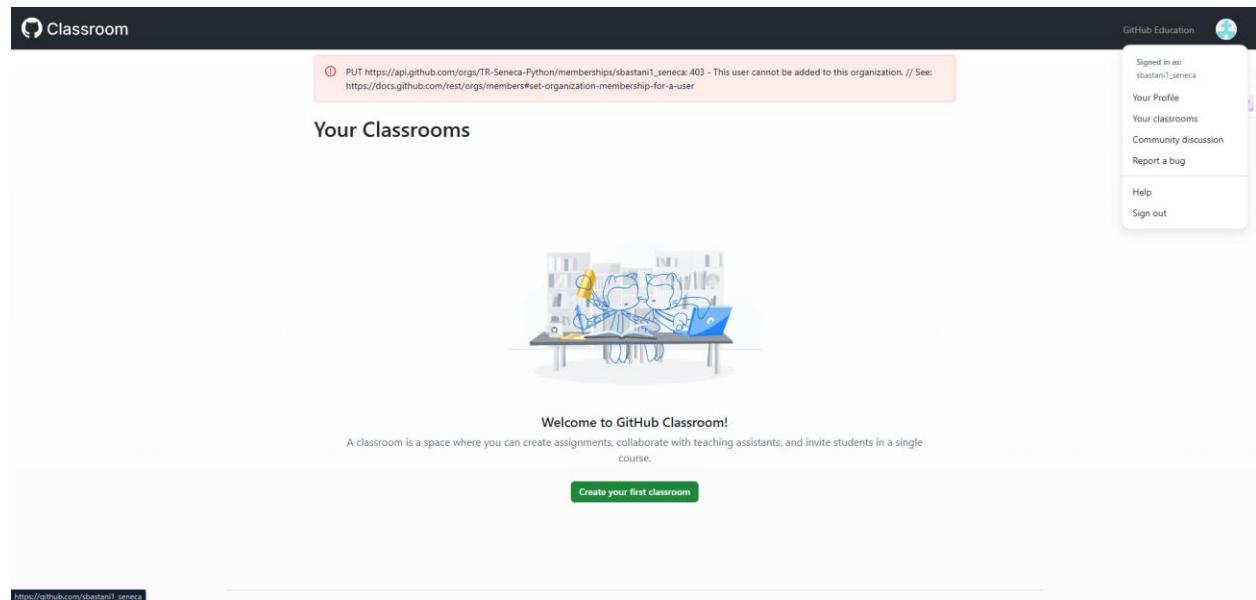
#### Join the GitHub Student Developer Pack

Verified students receive free GitHub Pro plus thousands of dollars worth of the best real-world tools and training from GitHub Education partners — for free. For more information, visit [GitHub Student Developer Pack](#).

**Apply**

Note: You may receive an email invitation to join [TR-Seneca-Python](#) on your behalf. No further action is necessary.

But when I use the Seneca SSO (by going to <https://github.com/senecapolytechnic>) and try to accept the lab, this happens:



The screenshot shows the GitHub Classroom interface after logging in via SSO. A red error message box at the top indicates a 403 error: "PUT https://api.github.com/orgs/TR-Seneca-Python/memberships/sbastani1\_seneca: 403 - This user cannot be added to this organization. // See: https://docs.github.com/rest/orgs/members#set-organization-membership-for-a-user". On the right, a sidebar shows the user is signed in as "sbastani1\_seneca" and provides links to "Your Profile", "Your classrooms", "Community discussion", "Report a bug", "Help", and "Sign out". The main area displays a cartoon illustration of two people working at a desk. Below it, a welcome message reads: "Welcome to GitHub Classroom! A classroom is a space where you can create assignments, collaborate with teaching assistants, and invite students in a single course." A green "Create your first classroom" button is visible.

My Seneca ID is sbastani1

Thanks for reading