

## **Data Science & AI Fellowship**

**Instructor/Mentor: Sir Noorullah**

**Assignment # 02**

**Submitted by: Nadia Mustafa**

---

### **Problem # 01 – Debugging**

#### **1. Error Version (Code)**

```
print(Day 1 - String Manipulation)  
print("String Concatenation is done with the "+" sign.")  
print('e.g. print("Hello " + world)')
```

#### **2. Debugging Explanation**

- Line 1 Error:**

The text Day 1 - String Manipulation is **not inside quotation marks**. Python thinks it's code, not a string.

- Line 2 Error:**

The **string is broken** because of the "+" sign. part. The quotes are not placed correctly.

- Line 3 Error:**

The word world is **missing quotation marks**. Python assumes it is a variable, but it should be a string.

#### **3. Corrected Code**

```
print("Day 1 - String Manipulation ")  
print('String Concatenation is done with the "+" sign. ')  
print('e.g. print("Hello " + "world")')
```

The screenshot shows the Thonny IDE interface. The code editor window displays the following Python script:

```
224
225
226
227
228
229
230 print("Day 1 - String Manipulation ")
231 print('String Concatenation is done with the "+" sign.')
232 print('e.g. print("Hello " + "world")')
233
234
235
236
237
238
239
```

The shell window shows the output of running the script:

```
>>> %Run Assignment.py
Day 1 - String Manipulation
String Concatenation is done with the "+" sign.
e.g. print("Hello " + "world")
>>>
```

The variables panel shows the following:

Name	Value
my_name	'Hello world'
name_length	13

The assistant panel provides feedback:

The code in [Assignment.py](#) looks good.  
If it is not working as it should, then consider using some general [debugging techniques](#).

[Was it helpful or confusing?](#)

---

## Problem # 02 - Inputs

```
my_name = input("Please Enter your name: ")
name_length = len(my_name)
print(name_length)
```

The screenshot shows the Thonny IDE interface. The code editor window displays the following Python script:

```
233
234
235 my_name = input("Please Enter your name: ")
236 name_length = len(my_name)
237 print(name_length)
238
239
240
241
242
243
244
245
246
247
248
```

The shell window shows the output of running the script:

```
>>> %Run Assignment.py
Please Enter your name: Nadia Mustafa
13
>>> |
```

The variables panel shows the following:

Name	Value
my_name	'Nadia Mustafa'
name_length	13

The assistant panel provides feedback:

The code in [Assignment.py](#) looks good.  
If it is not working as it should, then consider using some general [debugging techniques](#).

[Was it helpful or confusing?](#)

### Problem # 03 - Variables

```
a = input("enter any value: ")  
b = input("What is temperature outside: ")
```

```
temp = a
```

```
a = b
```

```
b = temp
```

```
print("a: " + a)
```

```
print("b: " + b)
```

### Example Dry Run

A **dry run** means we trace the program line by line and write down what's inside each variable (**a**, **b**, **temp**) at every step.

👉 Suppose the user types:

- a = 10
- b = 25

Step	Code Line	a	b	temp	What happens
1	a = input("enter any value")	10	-	-	User enters 10, stored in a
2	b = input("What is temperature outside")	10	25	-	User enters 25, stored in b
3	temp = a	10	25	10	Save a's value into temp
4	a = b	25	25	10	Put b's value into a
5	b = temp	25	10	10	Put temp (old a) into b
6	Print results	25	10	10	Shows swapped values

## Final Output

a: 25

b: 10

---

✓ That's the **swap in action**: the **two values exchanged places**.

---

## Printing the swapped values

```
print("a: " + a)
```

```
print("b: " + b)
```

- "a:" and "b:" are **string labels**.
- The **+** operator **concatenates** (joins) these labels with the **values stored in a and b**.
- Since both **input values** are **stored as strings**, they join correctly.
- Example: "a:" + "5" → "a: 5"

## Example # 02 – with string:

The screenshot shows the Thonny IDE interface. The code in `Assignment.py` is as follows:

```
1 # Problem no. 02
2
3 # my_name = len(input("Please Enter your name: "))
4 #name_length = len(my_name)
5 #print(my_name)
6
7 a = input("enter your name: ")
8 b = input("how is your health?: ")
9
10 # swap using a temporary variable
11 temp = a
12 a = b
13 b = temp
14
15 print("a: " + a)
16 print("b: " + b)
17
18
```

The Variables panel shows:

Name	Value
a	'fever'
b	'nadia'
temp	'nadia'

The Shell panel shows the output of running the script:

```
>>> %Debug Assignment.py
enter your name: nadia
how is your health?: fever
a: fever
b: nadia
>>>
```

## Problem # 04 - Printing to the Console

program in main.py

```
print("Day 1 - Python Print Function")
print("The function is declared like this:")
print("print('what to print')")
```

The screenshot shows the Thonny Python IDE interface. The top menu bar includes File, Edit, View, Run, Tools, and Help. The main window has tabs for 'Assignment.py' and 'Shell'. The 'Assignment.py' tab displays the following code:

```
14 #
15 # print("a: " + a)
16 # print("b: " + b)
17
18
19 print("Day 1 - Python Print Function")
20 print("The function is declared like this:")
21 print("print('what to print')")
22
23
24
25
26
```

The 'Shell' tab shows the output of running the script:

```
>>> %Debug Assignment.py
Day 1 - Python Print Function
The function is declared like this:
print('what to print')
>>>
```

On the right side of the interface, there are several panes: 'Variables' (empty), 'Assistant' (with a message about the code being good), and 'Local Python' (empty).

---

## Problem # 05 – Data Types

code on lines 1–3 = Input() & Variable

`input("Type a two-digit number: ")` -> string value/data type

`two_digit_number = input("Type a two-digit number: ")` -> Variable

`print(type(two_digit_number))` -> Type function

`<class 'str'>`

## What is Subscripting (substring)?

Subscripting = **picking out characters from a string using square brackets []**.

Example:

```
word = "hello"  
  
print(word[0])      # h  
  
print(word[1])      # e  
  
print(word[4])      # o
```

### ☞ Indexing in Python starts from 0:

- word[0] = **first character**
- word[1] = **second character**
- and so on.

So "39" → is just like "hello" a **sequence of characters**.

- "39"[0] = "3"
- "39"[1] = "9"

```
first_digit = int(two_digit_number[0])      # "3" → 3  → int() Function  
  
second_digit = int(two_digit_number[1])    # "9" → 9  → int() Function  
  
print(first_digit + second_digit)           # 3 + 9 = 12
```

### ✓ Key Takeaway

- "3" + "9" → "39" (**string concatenation**).
- int("3") + int("9") → **12 (integer addition)**.

That's why we **must use int()** when we want **digits to be added**.

```
two_digit_number = input("Type a two-digit number: ")  
print(type(two_digit_number))  
  
print(two_digit_number[0])  
print(two_digit_number[1])  
  
first_digit = int(two_digit_number[0])  
second_digit = int(two_digit_number[1])  
  
print(first_digit + second_digit)
```

Variables

Name	Value
first_digit	3
second_digit	9
two_digit_number	'39'

Assistant

The code in Assignment.py looks good.  
If it is not working as it should, then consider using some general debugging techniques.

Was it helpful or confusing?

```
>>> %Debug Assignment.py  
Type a two-digit number: 39  
<class 'str'>  
3  
9  
12  
>>>
```

---

## Problem # 06 - Your Life in Weeks

### Your Life in Weeks Program

#### # Step 1: Take input

```
user_age = input("What is your current age?: ")  
age = int(user_age)
```

#### # Step 2: Define lifespan

```
max_age = 90
```

#### # Step 3: Calculate remaining years

```
years_left = max_age - age
```

#### # Step 4: Convert into days, weeks, and months

```
days = years_left * 365
```

```
weeks = years_left * 52
```

```
months = years_left * 12
```

#### # Step 5: Store the message in a variable using f-string

```
message = f"You have {days} days, {weeks} weeks, and {months} months left."
```

## # Step 6: Print the message

```
print(message)
```

The screenshot shows the Thonny Python IDE interface. The code editor contains a script named 'Assignment.py' with the following content:

```
34 # Your Life in Weeks Program
35 user_age = input("What is your current age?: ")
36 # Step 1: Take input
37 age = int(user_age)
38
39 # Step 2: Define lifespan
40 max_age = 90
41
42 # Step 3: Calculate remaining years
43 years_left = max_age - age
44
45 # Step 4: Convert into days, weeks, and months
46 days = years_left * 365
47 weeks = years_left * 52
48 months = years_left * 12
49
50 # Step 5: Store the message in a variable using f-string
51 message = f"You have {days} days, {weeks} weeks, and {months} months left."
52
53 # Step 6: Print the message
54 print(message)
55
56
```

The shell window shows the output of running the script:

```
>>> %Debug Assignment.py
What is your current age?:25
You have 23725 days, 3380 weeks, and 780 months left.
>>>
```

A sidebar on the right displays the variables in the current scope:

Name	Value
age	25
days	23725
max_age	90
message	'You have 23725 da'
months	780
user_age	'25'
weeks	3380
years_left	65

## 1. What is an **f-string**?

- f stands for **formatted string**.
- An **f-string** is a string with the letter **f (or F)** before the quotation marks.
- It allows us to insert **variables, numbers, or calculations** inside a string easily

## 2. How f-strings work

- **Inside the f-string**, we use **curly brackets {}**.
- Whatever we **put inside {}** will be **evaluated** (calculated or replaced by its value).
- Anything **outside {}** is treated as **normal text (letters, words, symbols)**.

---

## 3. Variables inside {}

```
age = 25
```

```
print(f"I am {age} years old.")
```

👉 Output:

I am 25 years old.

- `{age}` is replaced by the **value 25**.
- 

#### 4. Calculations/Expressions inside {}

```
x = 10
```

```
y = 5
```

```
print(f"{x} + {y} = {x + y}")
```

👉 Output:

```
10 + 5 = 15
```

- `{x}` → 10
  - `{y}` → 5
  - `{x + y}` → 15 → **Expression**
  - `+` and `=` outside {} remain as **plain text**.
- 

#### 5. Strings inside {}

```
first = "Hello"
```

```
second = "World"
```

```
print(f"{first + second}")
```

👉 Output:

```
HelloWorld
```

If you want a **space**:

```
print(f"{first} {second}")
```

👉 Output:

```
Hello World
```

- ✓ Inside {} → + joins strings (concatenation).
- ✓ With numbers → + adds values.

---

## 6. F-strings are not only for print()

We can **store them in a variable, use them in files**, etc.

```
name = "Nadia"
```

```
marks = 95
```

```
message = f"Student {name} scored {marks} marks."
```

```
print(message)
```

👉 Output:

Student Nadia scored 95 marks.

### Why f-strings are useful

- Automatically convert numbers into strings (**no need for str()**).
  - **Cleaner and easier** than using + for concatenation.
  - Work with **variables, numbers, calculations, and even functions**.
  - Can be used anywhere a string is needed (printing, storing, writing to files).
- 

### ✅ Final Takeaway:

An f-string in Python is a powerful way to mix text with variables or calculations. **Curly brackets { }** act as placeholders where Python inserts the actual values. **Words and symbols** are written normally outside the brackets, while numbers, variables, or expressions go inside.

---

### ❗ Why f-strings useful here

- Normally, **Python doesn't let you directly join text (string) with numbers (int)**.

Example (✗ wrong):

```
age = 25
```

```
print("I am " + age + " years old.") # This gives an error
```

- You would need to convert the **number manually** → str(age).

- But with **f-strings**, Python does the conversion **automatically** inside { }.

**F-strings** allow us to combine letters (strings) and digits (numbers) in one sentence easily. The **number inside { }** is automatically converted into text, so we do not need to use **str()** or the **+** operator.

---

### Problem # 07 - Odd or Even

```
number = int(input("Enter a number: "))
```

If user types 43, then:

- **input()** → "43" (string)
- **int("43")** → 43 (integer)
- Now we can do math (43 % 2).

**if number % 2 == 0:**

```
    print("This is an even number.")
```

**else:**

```
    print("This is an odd number.")
```

**Examples:**

6 % 2            # result = 0 → no remainder → even

5 % 2            # result = 1 → remainder → odd

14 % 4          # result = 2 → remainder is 2

```

Thonny - C:\Users\Use\Desktop\Assignment.py @ 65 : 36
File Edit View Run Tools Help
Program arguments: 
Assignment.py
52 # message = f"You have {days} days, {weeks} weeks, and {months} months left."
53 #
54 # # Step 6: Print the message
55 # print(message)
56
57
58
59
60 number = int(input("Enter a number: "))
61
62 if number % 2 == 0:
63     print("This is an even number.")
64 else:
65     print("This is an odd number.")
66
67
Shell >
>>> %Debug Assignment.py
Enter a number: 23
This is an odd number.
>>>

```

### Problem # 08 - BMI Calculator 2.0

```
weight = float(input("Enter your weight in kg: "))
```

```
height = float(input ("Enter your height in m: "))
```

```
BMI = round(weight / height**2)           # round() -> to get int
```

```
if BMI <= 18.5:                      # Under 18.5
```

```
    print(f"Your BMI is {BMI}, you are \033[1munderweight\033[0m.")
```

```
elif BMI > 18.5 and BMI < 25:      # Over 18.5 but below 25
```

```
    print(f"Your BMI is {BMI}, you have a \033[1mnormal weight\033[0m.")
```

```
elif BMI > 25 and BMI < 30:        # Over 25 but below 30
```

```
    print(f"Your BMI is {BMI}, you are \033[1mslightly overweight\033[0m.")
```

```
elif BMI > 30 and BMI < 35:        # Over 30 but below 35
```

```
    print(f"Your BMI is {BMI}, you are \033[1mobese\033[0m.")
```

```
elif BMI > 35: # Above 35
```

```
    print(f"Your BMI is {BMI}, you are \033[1mclinically obese\033[0m.")
```

## Meaning of \033[1m... \033[0m

This is called an **ANSI escape sequence**. It's a code that tells the terminal **how to format text**.

### Breaking it down:

- \033 → This is the **escape character** (written in Python as \033).
- [1m → Means **turn bold ON**.
- ... → Your actual text goes here.
- \033[0m → Means **reset to normal text** (turn bold OFF).

## Without Bold Interpretation:

The screenshot shows the Thonny IDE interface with a Python script named 'Assignment.py' open. The script calculates BMI based on user input for weight and height, then prints a message indicating the result. The 'Variables' panel on the right shows the values for BMI (23), height (1.75), and weight (70.0). The 'Shell' panel at the bottom shows the execution of the script and the resulting output: 'Your BMI is 23, you have a normal weight.' The output is displayed in a standard black font, indicating no bold interpretation.

```
Thonny - C:\Users\Use\Desktop\Assignment.py @ 79 : 12
File Edit View Run Tools Help
Assignment.py * 
73 weight = float(input("Enter your weight in kg: "))
74 height = float(input ("Enter your height in m: "))
75
76 BMI = round(weight / height**2) # round() -> to get int
77
78 if BMI <= 18.5:
79     print(f'Your BMI is {BMI}, you are underweight.')
80 elif BMI > 18.5 and BMI < 25:
81     print(f'Your BMI is {BMI}, you have a normal weight.')
82 elif BMI > 25 and BMI < 30:
83     print(f'Your BMI is {BMI}, you are slightly overweight')
84 elif BMI > 30 and BMI < 35:
85     print(f'Your BMI is {BMI}, you are obese.')
86 elif BMI > 35:
87     print(f'Your BMI is {BMI}, you are clinically obese.')

>>> %Debug Assignment.py
Enter your weight in kg: 70
Enter your height in m: 1.75
Your BMI is 23, you have a normal weight.
>>>
```

## With Bold Interpretation:

```

73 weight = float(input("Enter your weight in kg: "))
74 height = float(input ("Enter your height in m: "))
75
76 BMI = round(weight / height**2) # round() -> to get int
77
78 if BMI <= 18.5:
79     print(f"Your BMI is {BMI}, you are \033[1munderweight\033[0m.")
80 elif BMI > 18.5 and BMI < 25:
81     print(f"Your BMI is {BMI}, you have a \033[1mnormal weight\033[0m.")
82 elif BMI > 25 and BMI < 30:
83     print(f"Your BMI is {BMI}, you are \033[1mslightly overweight\033[0m.")
84 elif BMI > 30 and BMI < 35:
85     print(f"Your BMI is {BMI}, you are \033[1mobese\033[0m.")
86 elif BMI > 35:
87     print(f"Your BMI is {BMI}, you are \033[1mclinically obese\033[0m.")
88

```

**Variables**

Name	Value
BMI	125
height	2.0
weight	500.0

**Shell**

```

>>> %Debug Assignment.py
Enter your weight in kg: 500
Enter your height in m: 2
Your BMI is 125, you are clinically obese.

>>>

```

## Problem # 09 - Leap Year

### Leap Year Program – Deduced code structure – From Flowchart

**year = 2015**

**if year % 4 == 0:** → Is it divisible by 4?

**if year % 100 == 0:** Yes → check divisibility by 100.

**if year % 400 == 0:** Yes → check divisibility by 400.

**print('Leap year')** Yes → leap.

**else:**

**print('Not leap year')** No → leap.

**else:**

**print('Leap year')** Yes → leap.

**else:**

**print('Not leap year')** No → not leap.

The screenshot shows the Thonny Python IDE interface. The code editor contains the following Python script:

```

Assignment.py practice.py
1
27     print("Leap year.")
28 # else:
29 #     print("Not leap year.")
30
31 year = int(input('Enter year to check leap year: '))
32 if year % 4 == 0:
33     if year % 100 == 0:
34         if year % 400 == 0:
35             print('Leap year')
36         else:
37             print('Not leap year')
38     else:
39         print('Leap year')
40 else:
41     print('Not leap year.')
42

```

The shell window shows the execution of the script and its output:

```

>>> %Debug practice.py
Enter year to check leap year: 1989
Not leap year.

>>>

```

Variables	
Name	Value
year	1989

## 2<sup>nd</sup> code Structure - Leap Year Program

```

year = int(input('Enter year to check leap year: '))

if year % 4 == 0 and (year % 100 != 0 or year % 400 == 0):

    print("Leap year.")

else:

    print("Not leap year.")

```

The screenshot shows the Thonny Python IDE interface. The code editor contains the following Python script:

```

Assignment.py practice.py
40
41 #     print('Not leap year.')
42
43
44 year = int(input('Enter year to check leap year: '))
45 if year % 4 == 0 and (year % 100 != 0 or year % 400 == 0):
46     print("Leap year.")
47 else:
48     print("Not leap year.")

49
50
51
52
53
54
55

```

The shell window shows the execution of the script and its output:

```

>>> %Debug practice.py
Enter year to check leap year: 2000
Leap year.

>>>

```

Variables	
Name	Value
year	2000

## Problem # 10 - Pizza Order Program

- Ask the user to choose a pizza size

.upper() ensures input is always in uppercase (S, M, or L)

```
size = input("Choose a size (S, M, or L): ").upper()
```

- Ask if the user wants pepperoni (Y for Yes, N for No)

```
add_pepperoni = input("Do you want pepperoni? (Y or N): ").upper()
```

- Ask if the user wants extra cheese (Y for Yes, N for No)

```
extra_cheese = input("Do you want extra cheese? (Y or N): ").upper()
```

**Step 1:** Start bill at 0

```
bill = 0 # this variable will store the total price
```

**Step 2:** Add price based on pizza size

```
if size == "S": # if user chose Small  
    bill += 15 # add 15 dollars to bill  
  
elif size == "M": # if user chose Medium  
    bill += 20 # add 20 dollars to bill  
  
elif size == "L": # if user chose Large  
    bill += 25 # add 25 dollars to bill  
  
else:  
    print("Invalid size chosen!") # safety check if user typed wrong  
    bill = 0 # reset bill in case of invalid input
```

**Step 3:** Add cost for pepperoni

```
if add_pepperoni == "Y": # check if user wants pepperoni  
    if size == "S": # pepperoni price depends on size  
        bill += 2 # add 2 for small pizza  
  
    else:  
        bill += 3 # add 3 for medium or large pizza
```

#### Step 4: Add cost for extra cheese

```
if extra_cheese == "Y":          # check if user wants extra cheese  
    bill += 1                  # always add 1 regardless of size
```

#### Step 5: Show the final bill

```
print(f"Your final bill is: ${bill}.")
```

The screenshot shows the Thonny Python IDE interface. The code in the editor window is:

```
105 size = input("Choose a size (S, M, L): ").strip().upper()  
106 add_pepperoni = input("Add pepperoni? (Y/N): ").strip().upper()  
107 extra_cheese = input("Extra cheese? (Y/N): ").strip().upper()  
108  
109 bill = 0  
110 if size == "S":  
111     bill += 15  
112 elif size == "M":  
113     bill += 20  
114 elif size == "L":  
115     bill += 25  
116  
117 if add_pepperoni == "Y":  
118     if size == "S":  
119         bill += 2  
120     else:  
121         bill += 3  
122  
123 if extra_cheese == "Y":  
124     bill += 1  
125  
126 print(f"Your final bill is: ${bill}.")  
127
```

The shell window shows the execution of the script:

```
>>> %Debug practice.py  
Choose a size (S, M, L): M  
Add pepperoni? (Y/N): Y  
Extra cheese? (Y/N): Y  
Your final bill is: $24.  
>>>
```

The variables pane shows the current values:

Name	Value
add_pepperoni	'Y'
bill	24
extra_cheese	'Y'
size	'M'

The status bar indicates: "The code in [practice.py](#) looks good." and "If it is not working as it should, then consider using some general debugging techniques."

```
size = input("Choose a size (S, M, L): ").strip().upper()
```

```
add_pepperoni = input("Add pepperoni? (Y/N): ").strip().upper()
```

```
extra_cheese = input("Extra cheese? (Y/N): ").strip().upper()
```

The screenshot shows the Thonny IDE interface. The top menu bar includes File, Edit, View, Run, Tools, and Help. The main window displays a Python script named practice.py:

```

105 size = input("Choose a size (S, M, L): ").strip().upper()
106 add_pepperoni = input("Add pepperoni? (Y/N): ").strip().upper()
107 extra_cheese = input("Extra cheese? (Y/N): ").strip().upper()
108
109 bill = 0
110 if size == "S":
111     bill += 15
112 elif size == "M":
113     bill += 20
114 elif size == "L":
115     bill += 25
116
117 if add_pepperoni == "Y":
118     if size == "S":
119         bill += 2
120     else:
121         bill += 3
122
123 if extra_cheese == "Y":
124     bill += 1
125
126 print(f"Your final bill is: ${bill}.")
127

```

The right side of the interface features a sidebar with tabs for Variables, Assistant, and Help. The Variables tab shows the current state of variables:

Name	Value
add_pepperoni	'Y'
bill	24
extra_cheese	'Y'
size	'M'

The Assistant tab contains a message from the code in practice.py: "The code in practice.py looks good. If it is not working as it should, then consider using some general debugging techniques." A link "Was it helpful or confusing?" is also present.

**.strip()** is a **string method** that removes **unwanted spaces (and invisible characters)** from the beginning and end of a string.

👉 It removes the extra spaces before and after the word, but not the ones **inside**.

### Complete dictionary-based solution:

```

size = input("Choose a size (S, M, L): ").strip().upper()

add_pepperoni = input("Add pepperoni? (Y/N): ").strip().upper()

extra_cheese = input("Extra cheese? (Y/N): ").strip().upper()

```

### Dictionaries for prices

```
base_prices = {"S": 15, "M": 20, "L": 25}
```

```
pepperoni_prices = {"S": 2, "M": 3, "L": 3}
```

```
bill = base_prices[size] # Start with base price
```

```
if add_pepperoni == "Y": # Add pepperoni if needed
```

```
    bill += pepperoni_prices[size]
```

```

if extra_cheese == "Y":          # Add cheese if needed

    bill += 1

print(f"Your final bill is: ${bill}.")      # Final result

```

```

117 size = input("Choose a size (S, M, L): ").strip().upper()
118 add_pepperoni = input("Add pepperoni? (Y/N): ").strip().upper()
119 extra_cheese = input("Extra cheese? (Y/N): ").strip().upper()
120
121
122 # Dictionaries for prices
123 base_prices = {"S": 15, "M": 20, "L": 25}
124 pepperoni_prices = {"S": 2, "M": 3, "L": 3}
125
126 # Start with base price
127 bill = base_prices[size]
128
129 # Add pepperoni if needed
130 if add_pepperoni == "Y":
131     bill += pepperoni_prices[size]
132
133 # Add cheese if needed
134 if extra_cheese == "Y":
135     bill += 1
136
137
138 # Final result
139 print(f"Your final bill is: ${bill}.")
140

```

Shell

```

>>> #Debug Assignment.py
Choose a size (S, M, L): M
Add pepperoni? (Y/N): Y
Extra cheese? (Y/N): N
Your final bill is: $23.

>>>

```

Variables

Name	Value
add_pepperoni	"Y"
base_prices	{'S': 15, 'M': 20, 'L': 25}
bill	23
extra_cheese	"N"
pepperoni_prices	{'S': 2, 'M': 3, 'L': 3}
size	'M'

Assistant

The code in [Assignment.py](#) looks good.  
If it is not working as it should, then consider using some general debugging techniques.  
[Was it helpful or confusing?](#)

The **square brackets []** in Python are called the **subscript operator**. They are used to **pick or access data** from **collections like lists, strings, or dictionaries**.

In this program, we use **[] with a dictionary** to get the correct base price of the pizza:

```

base_prices = {"S": 15, "M": 20, "L": 25}

bill = base_prices[size]

```

If the **user chooses "L"**, then `base_prices[size]` becomes `base_prices["L"]` and gives the value 25.

👉 Therefore, in this program we use `[]` to **look up the pizza price by its size (key)** instead of writing multiple if statements.

## Problem # 11 - Love Calculator

```
name1 = input("Enter the first name: ")  
name2 = input("Enter the second name: ")  
combined_names = (name1 + name2).lower()
```

### **Count letters for the word "TRUE"**

```
t = combined_names.count("t")  
r = combined_names.count("r")  
u = combined_names.count("u")  
e = combined_names.count("e")  
true_score = t + r + u + e
```

### **Count letters for the word "LOVE"**

```
l = combined_names.count("l")  
o = combined_names.count("o")  
v = combined_names.count("v")  
e = combined_names.count("e")  
love_score = l + o + v + e
```

### **Combine the two numbers (not add, but join as text, then convert back to number)**

```
final_score = int(str(true_score) + str(love_score))
```

### **Apply conditions to print result**

```
if final_score < 10 or final_score > 90:  
    print(f"Your score is {final_score}, you go together like coke and mentos.")  
elif 40 <= final_score <= 50:  
    print(f"Your score is {final_score}, you are alright together.")  
else:  
    print(f"Your score is {final_score}.")
```

The screenshot shows the Thonny Python IDE interface. The code editor displays `Assignment.py` with logic to calculate a combined score from two names based on letter counts. The variables pane shows the state of variables after running the script with inputs "Nadia" and "Salma Mustafa". The shell pane shows the execution of the script and its output.

```
Assignment.py *
148 t = combined_names.count("t")
149 r = combined_names.count("r")
150 u = combined_names.count("u")
151 e = combined_names.count("e")
152 true_score = t + r + u + e
153 # Count letters for the word "LOVE"
154 l = combined_names.count("l")
155 o = combined_names.count("o")
156 v = combined_names.count("v")
157 e = combined_names.count("e")
158 love_score = l + o + v + e
159 # Combine the two numbers (not add, but join as text, then convert back to number)
160 final_score = int(str(true_score) + str(love_score))
161 # Apply conditions to print result
162 if final_score < 10 or final_score > 90:
163     print(f"Your score is {final_score}, you go together like coke and mentos.")
164 elif 40 <= final_score <= 50:
165     print(f"Your score is {final_score}, you are alright together.")
166 else:
167     print(f"Your score is {final_score}.")
```

Variables:

Name	Value
combined_name	'nadiasalma mustafa'
e	0
final_score	21
l	1
love_score	1
name1	'Nadia'
name2	'Salma Mustafa'
o	0
r	0
t	1
true_score	2
u	1
v	0

Shell:

```
>>> %Debug Assignment.py
Enter the first name: Nadia
Enter the second name: Salma Mustafa
Your score is 21.
>>>
```

Assistant:

The code in [Assignment.py](#) looks good.  
If it is not working as it should, then consider using some general debugging techniques.  
[Was it helpful or confusing?](#)

The **count() function** tells us **how many times a certain letter** appears in a string. We use it because instead of checking letters manually one by one, count() does the work for us **automatically**.

## Why Concatenate Instead of Add?

👉 “Combine” here means **putting digits side by side**, not adding them.

### Example to See the Difference

**Take names:** "Kim alish" + "Jack Bauer"

- TRUE score = **3**
- LOVE score = **2**

**If we add:**

$$3 + 2 = 5$$

Love Score = 5 ✗ (just **one digit**, doesn't match rules)

**If we concatenate:**

$$"3" + "2" = "32"$$

Convert back to number → 32 ✓ (a proper **two-digit score**)

## How Concatenation Works

```
final_score = int(str(true_score) + str(love_score))
```

- true\_score = 3 → str(true\_score) = "3"
- love\_score = 2 → str(love\_score) = "2"
- "3" + "2" = "32" (string concatenation)
- int("32") = 32 (final number)

 **That's why concatenation is required:** it creates a “two-digit love score” instead of just a small sum.

---

## Problem # 12 - Treasure Island

```
print("Welcome to Treasure Island.")
```

```
print("Your mission is to find the treasure.")
```

**# Introduction**

### # Step 1: Crossroad

```
choice1 = input("You're at a crossroad. Where do you want to go? Type 'left' or 'right'.\n").lower()
```

```
if choice1 == "left":
```

### # Step 2: Lake

```
choice2 = input("You've come to a lake. Type 'wait' to wait for a boat. Type 'swim' to swim across.\n").lower()
```

```
if choice2 == "wait":
```

### # Step 3: Doors

```
choice3 = input("You arrive at the island unharmed. There is a house with 3 doors: red, yellow, and blue. Which colour do you choose?\n").lower()
```

```
if choice3 == "yellow":
```

```
    print("You found the treasure! 🎉 You Win!")
```

```
elif choice3 == "red":
```

```
    print("It's a room full of fire 🔥 . Game Over.")
```

```

elif choice3 == "blue":
    print("You enter a room of beasts 🦖. Game Over.")

else:
    print("You chose a door that doesn't exist. 🚪 Game Over.")

else:
    print("You get attacked by an angry trout 🐟. Game Over.")

else:
    print("You fell into a hole 🕸️. Game Over.")

```

The screenshot shows the Thonny IDE interface. The top menu bar includes File, Edit, View, Run, Tools, and Help. The main window displays the Python script `Assignment.py`. The code itself is as follows:

```

171 print('Welcome to Treasure Island')
172 print('Your mission is to find the treasure.')
173
174 choice1 = input("You're at a crossroad. Where do you want to go? Type 'left' or 'right'.\n").lower()
175
176 if choice1 == "left":
177     choice2 = input("You've come to a lake. Type 'wait' to wait for a boat. Type 'swim' to swim across.\n").lower()
178     if choice2 == "wait":
179         choice3 = input("You arrive at the island unharmed. There is a house with 3 doors. One red, one yellow, and one blue. Which colour do you choose?\n")
180         if choice3 == "yellow":
181             print("You found the treasure! You Win!")
182         elif choice3 == "red":
183             print("It's a room full of fire. Game Over.")
184         elif choice3 == "blue":
185             print("You enter a room of beasts. Game Over.")
186         else:
187             print("You chose a door that doesn't exist. Game Over.")
188     else:
189         print("You get attacked by an angry trout. Game Over.")
190 else:
191     print("You fell into a hole. Game Over.")
192

```

The middle pane shows the variables: choice1 ('left'), choice2 ('wait'), and choice3 ('yellow'). The bottom left pane shows the shell output of running the script, which includes the game logic and outcomes. The bottom right pane shows the 'Warnings' section with four entries about bad indentation.

## What \n Means

- `\n` is called a **newline character** in Python (and many programming languages).
- It tells Python: “**move the cursor to the next line**” when printing text for better readability.

- In this game, it makes the prompts clear, so the **player's answer doesn't mix with the question.**