

# AI-ASSISTED CODING

NAME:SANIYA

H.T NO: 2403A510E7

BATCH NO: 05

## Task #1: Email Validator

### Prompt

Write a Python program to validate email addresses using regular expressions. The program should define a function `is_valid_email()` that returns True if the email is valid, and False otherwise. It should then test multiple email samples and display whether each one is valid or not.

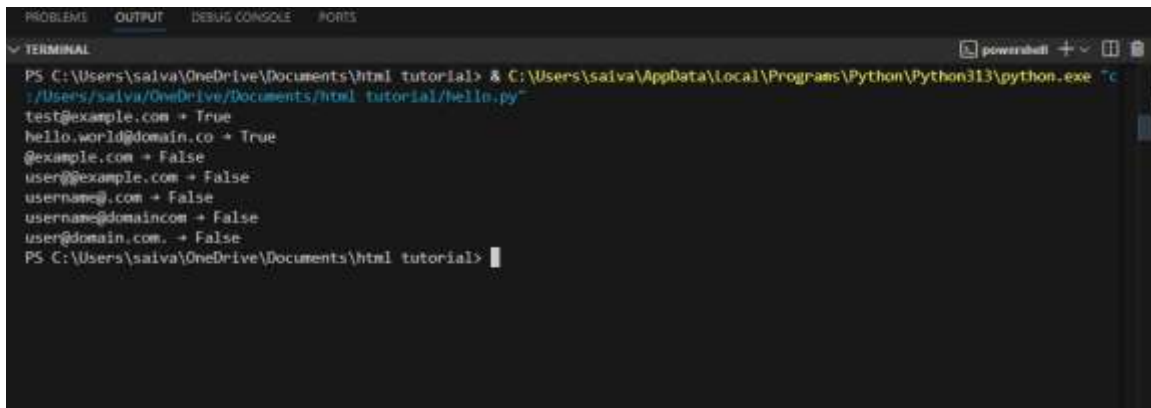
### Python Code:

```
email_validator.py •
email_validator.py > ...
1  import re
2
3  def is_valid_email(email: str) -> bool:
4      # Regex to validate email format
5      pattern = r'^[A-Za-z0-9]+[A-Za-z0-9._%+-]*@[A-Za-z0-9.-]+\.[A-Za-z]{2,}$'
6      return re.fullmatch(pattern, email) is not None
7
8
9  # Test Cases
10 test_emails = [
11     "test@example.com",      # ✓ valid
12     "hello.world@domain.co", # ✓ valid
13     "@example.com",         # ✗ starts with @
14     "user@@example.com",    # ✗ multiple @
15     "username@.com",         # ✗ domain error
16     "username@domaincom",    # ✗ no dot
17     "user@domain.com."       # ✗ ends with dot
18 ]
19
20 for email in test_emails:
21     print(f"{email} -> {is_valid_email(email)}")
22
```

### Output

# AI-ASSISTED CODING

---



The image shows a screenshot of a Visual Studio Code terminal window. The terminal is titled 'TERMINAL' and shows the execution of a Python script. The command prompt is 'PS C:\Users\saiiva\OneDrive\Documents\html tutorial>'. The command executed is 'C:\Users\saiiva\AppData\Local\Programs\Python\Python313\python.exe .\Users\saiiva\OneDrive\Documents\html tutorial\hello.py'. The output of the script is as follows:

```
test@example.com + True
hello.world@domain.co + True
@example.com + False
user@example.com + False
username@.com + False
username@domaincom + False
user@domain.com. + False
PS C:\Users\saiiva\OneDrive\Documents\html tutorial>
```

# AI-ASSISTED CODING

## Observation

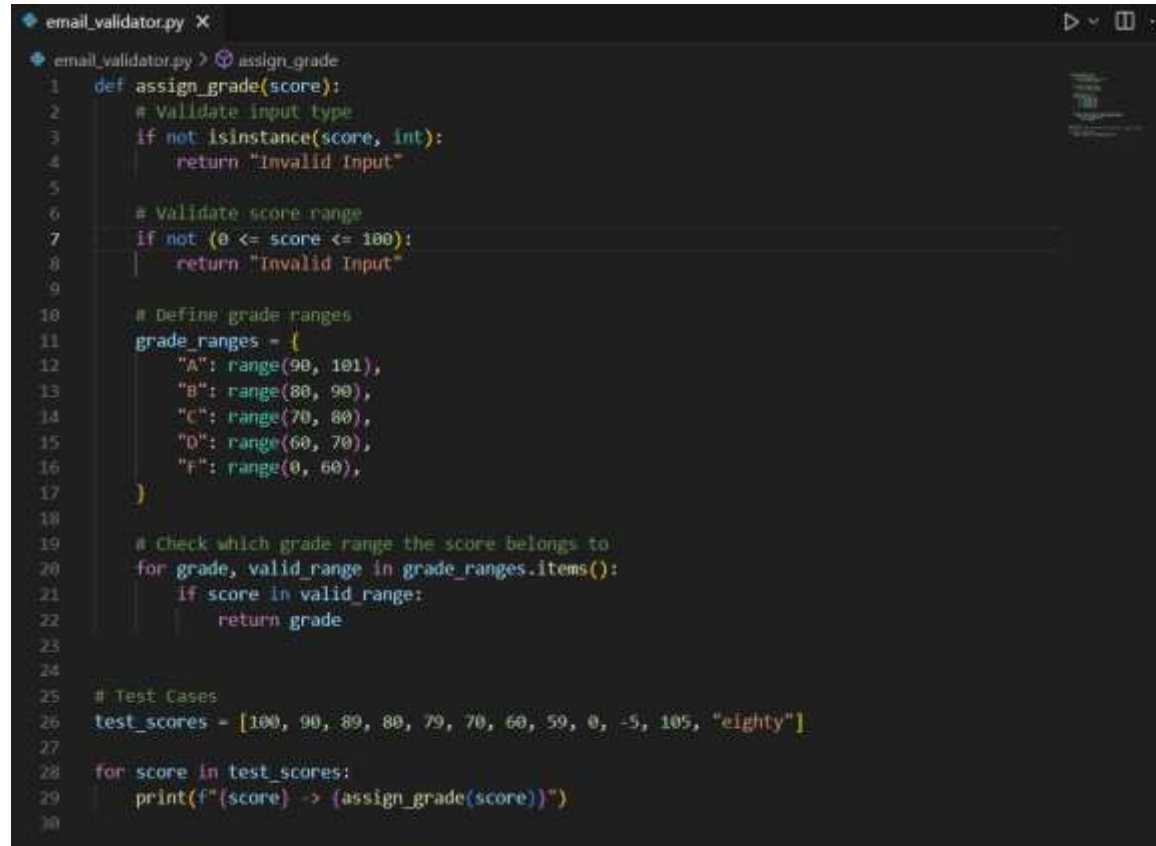
when the program is executed, it checks each test email against the regex pattern. Valid emails return True and invalid emails return False.

## Task #2: Grade Assignment

### Prompt

The program is designed to assign grades based on student scores. It uses a function `assign_grade()` that returns "A" for scores 90–100, "B" for 80–89, "C" for 70–79, "D" for 60–69, and "F" for below 60. It also checks for invalid inputs such as negative numbers, values above 100, or non-numeric entries.

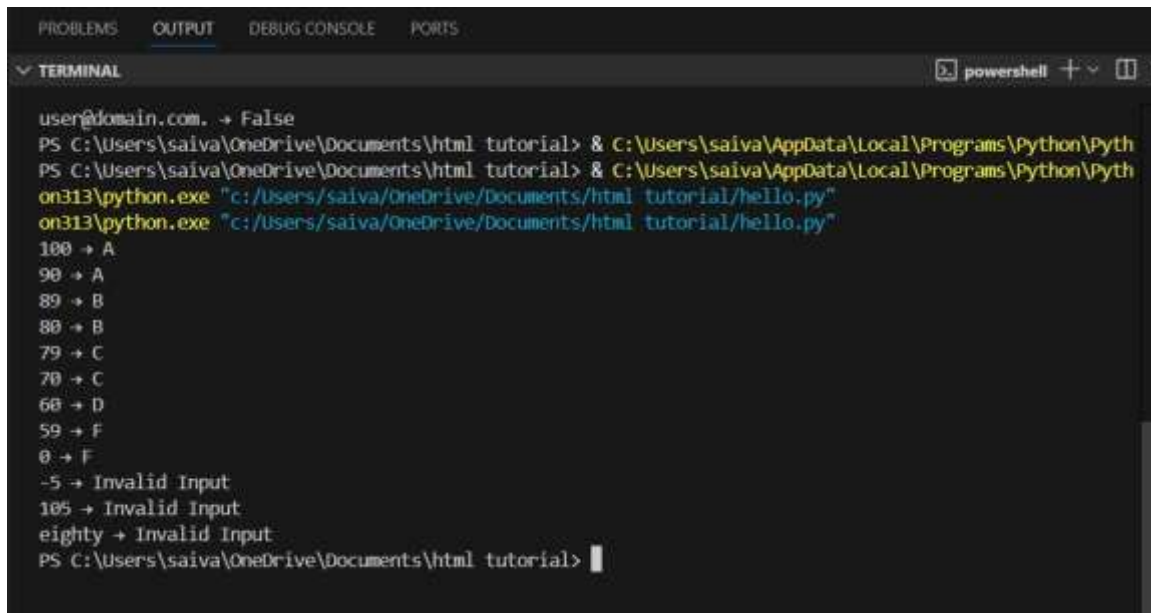
### Python Code

A screenshot of a code editor window titled 'email\_validator.py'. The code defines a function 'assign\_grade(score)' that validates input and assigns grades based on score ranges. It includes test cases for various scores and non-numeric input.

```
email_validator.py X
email_validator.py > assign_grade
1 def assign_grade(score):
2     # Validate input type
3     if not isinstance(score, int):
4         return "Invalid Input"
5
6     # Validate score range
7     if not (0 <= score <= 100):
8         return "Invalid Input"
9
10    # Define grade ranges
11    grade_ranges = {
12        "A": range(90, 101),
13        "B": range(80, 90),
14        "C": range(70, 80),
15        "D": range(60, 70),
16        "F": range(0, 60),
17    }
18
19    # Check which grade range the score belongs to
20    for grade, valid_range in grade_ranges.items():
21        if score in valid_range:
22            return grade
23
24
25    # Test Cases
26    test_scores = [100, 90, 89, 80, 79, 70, 60, 59, 0, -5, 105, "eighty"]
27
28    for score in test_scores:
29        print(f'{score} -> {assign_grade(score)}')
30
```

### Output

## AI-ASSISTED CODING



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  PORTS
TERMINAL
powershell + - [ ]
user@domain.com. → False
PS C:\Users\saiva\OneDrive\Documents\html tutorial> & C:\Users\saiva\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/saiva/OneDrive/Documents/html tutorial/hello.py"
on313\python.exe "c:/Users/saiva/OneDrive/Documents/html tutorial/hello.py"
100 → A
90 → A
89 → B
80 → B
79 → C
70 → C
60 → D
59 → F
0 → F
-5 → Invalid Input
105 → Invalid Input
eighty → Invalid Input
PS C:\Users\saiva\OneDrive\Documents\html tutorial> |
```

### Observation

When executed, the program correctly classified valid scores into their respective grades and returned "Invalid Input" for values like -5, 105, and "eighty". This shows that the program works correctly and handles errors effectively.

### Task #3: Sentence Palindrome

#### Prompt

Give test cases for `is_sentence_palindrome(sentence)` that ignores spaces, punctuation, and case by using ai.

#### Python Code

## AI-ASSISTED CODING

```
hello.py x
hello.py > ...
C:\Users\saiva\OneDrive\Documents\html tutorial\hello.py
2
3 def is_sentence_palindrome(sentence: str) -> bool:
4     cleaned = re.sub(r'^A-Za-z0-9', '', sentence).lower()
5     return cleaned == cleaned[::-1]
6
7 # Test Cases
8 test_sentences = [
9     "A man a plan a canal Panama", # ✓ True
10    "No lemon, no melon", # ✓ True
11    "Was it a car or a cat I saw?", # ✓ True
12    "Hello World", # ✗ False
13    "Racecar", # ✓ True
14    "Python coding" # ✗ False
15 ]
16
17 for s in test_sentences:
18     print(f"{s} -> {is_sentence_palindrome(s)}")
19
```

### Output

```
PROBLEMS OUTPUT DEBUG CONSOLE PORTS
TERMINAL powershell + -
PS C:\Users\saiva\OneDrive\Documents\html tutorial> & C:\Users\saiva\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/saiva/OneDrive/Documents/html tutorial/hello.py"
'A man a plan a canal Panama' -> True
'No lemon, no melon' -> True
'Was it a car or a cat I saw?' -> True
'Hello World' -> False
'Racecar' -> True
'Python coding' -> False
PS C:\Users\saiva\OneDrive\Documents\html tutorial> |
```

### Observation

The code correctly checks palindromes by ignoring case, spaces, and punctuation, returning **True** for valid palindromes and **False** otherwise.

# AI-ASSISTED CODING

## Task #4: Shopping Cart

### Prompt

Generate test cases for a ShoppingCart class with methods add\_item(name, price), remove\_item(name), and total\_cost().

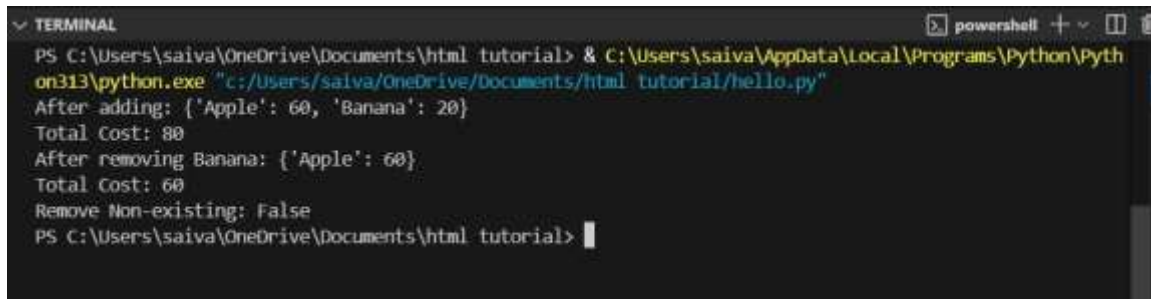
### Python Code

```
hello.py X
hello.py > ...
1 class ShoppingCart:
2     def __init__(self):
3         self.items = {}
4
5     def add_item(self, name, price):
6         if price < 0:
7             return "Invalid Price"
8         self.items[name] = self.items.get(name, 0) + price
9
10    def remove_item(self, name):
11        if name in self.items:
12            del self.items[name]
13            return True
14        return False
15
16    def total_cost(self) (variable) items: dict
17        return sum(self.items.values())
18
19 # Test Cases
20 cart = ShoppingCart()
21 cart.add_item("Apple", 30)
22 cart.add_item("Banana", 20)
23 cart.add_item("Apple", 30) # Add again
24 print("After adding:", cart.items)
25 print("Total Cost:", cart.total_cost())
26
27 cart.remove_item("Banana")
28 print("After removing Banana:", cart.items)
29 print("Total Cost:", cart.total_cost())
30
31 print("Remove Non-existing:", cart.remove_item("Orange"))
32
```

# AI-ASSISTED CODING

---

## Output



```
PS C:\Users\saiva\OneDrive\Documents\html tutorial> & C:\Users\saiva\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/saiva/OneDrive/Documents/html tutorial/hello.py"
After adding: {'Apple': 60, 'Banana': 20}
Total Cost: 80
After removing Banana: {'Apple': 60}
Total Cost: 60
Remove Non-existing: False
PS C:\Users\saiva\OneDrive\Documents\html tutorial>
```

## Observation:

The cart works fine: adding updates totals, duplicates add up, removing items lowers cost, and removing something not in the cart just returns **False**.

## Task #5: Date Format Converter

### Prompt:

write test cases for `convert_date_format(date_str)` to switch from "YYYY-MM-DD" to "DD-MM-YYYY".

### Python Code



## AI-ASSISTED CODING

```
hello.py X
hello.py > ...
1 def convert_date_format(date_str: str) -> str:
2     try:
3         year, month, day = date_str.split("-")
4         return f"{day}-{month}-{year}"
5     except:
6         return "Invalid Date Format"
7
8 # Test Cases
9 test_dates = [
10     "2023-10-15", # ✓ valid
11     "1999-01-01", # ✓ valid
12     "2025-12-31", # ✓ valid
13     "2023/10/15", # ✗ invalid
14     "15-10-2023"  # ✗ invalid
15 ]
16
17 for d in test_dates:
18     print(f"{d} → {convert_date_format(d)}")
19
```

### Output

```
PROBLEMS OUTPUT DEBUG CONSOLE PORTS
TERMINAL powershell + -
PS C:\Users\saiva\OneDrive\Documents\html tutorial> & C:\Users\saiva\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/saiva/OneDrive/Documents/html tutorial/hello.py"
2023-10-15 → 15-10-2023
1999-01-01 → 01-01-1999
2025-12-31 → 31-12-2025
2023/10/15 → Invalid Date Format
15-10-2023 → 2023-10-15
PS C:\Users\saiva\OneDrive\Documents\html tutorial>
```



---

## AI-Generated Test Cases and Implementations

### Observation

The code converts dates from YYYY-MM-DD to DD-MM-YYYY. Valid formats work fine, invalid ones like 2023/10/15 are caught, but 15-10-2023 is wrongly treated as valid since it still splits into 3 parts.