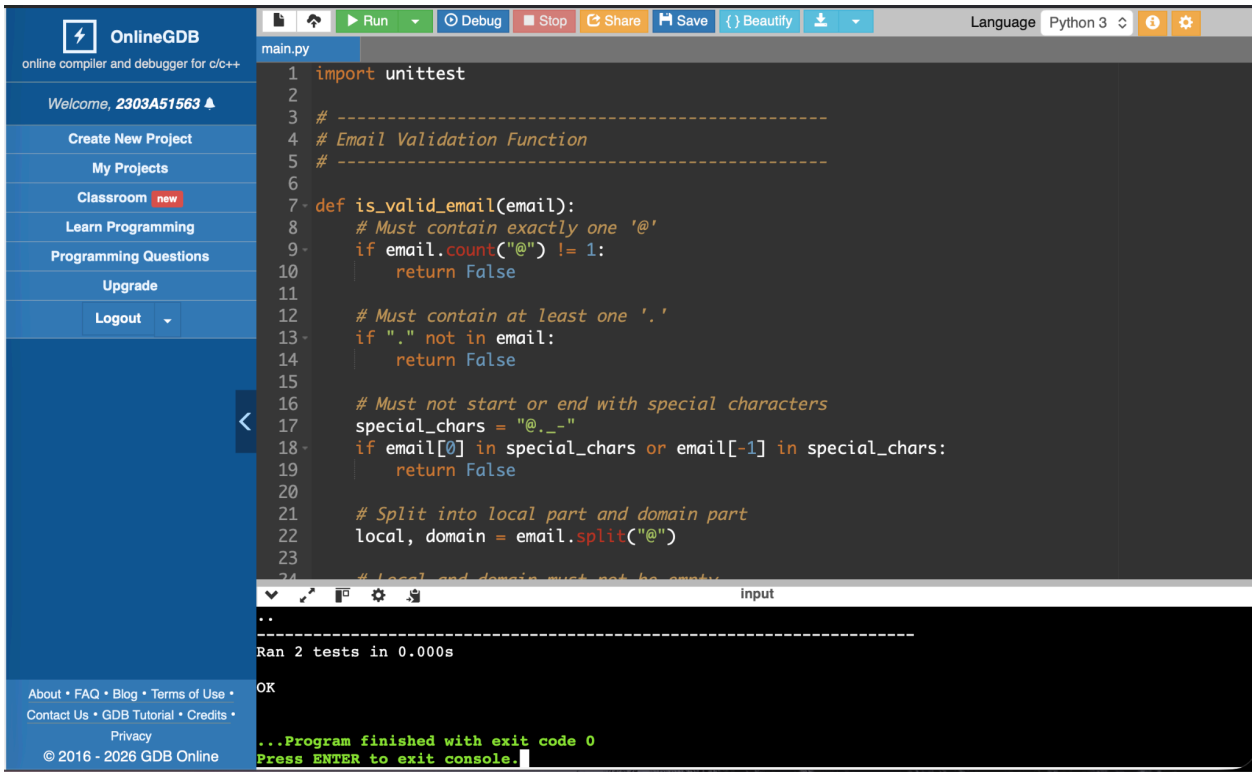
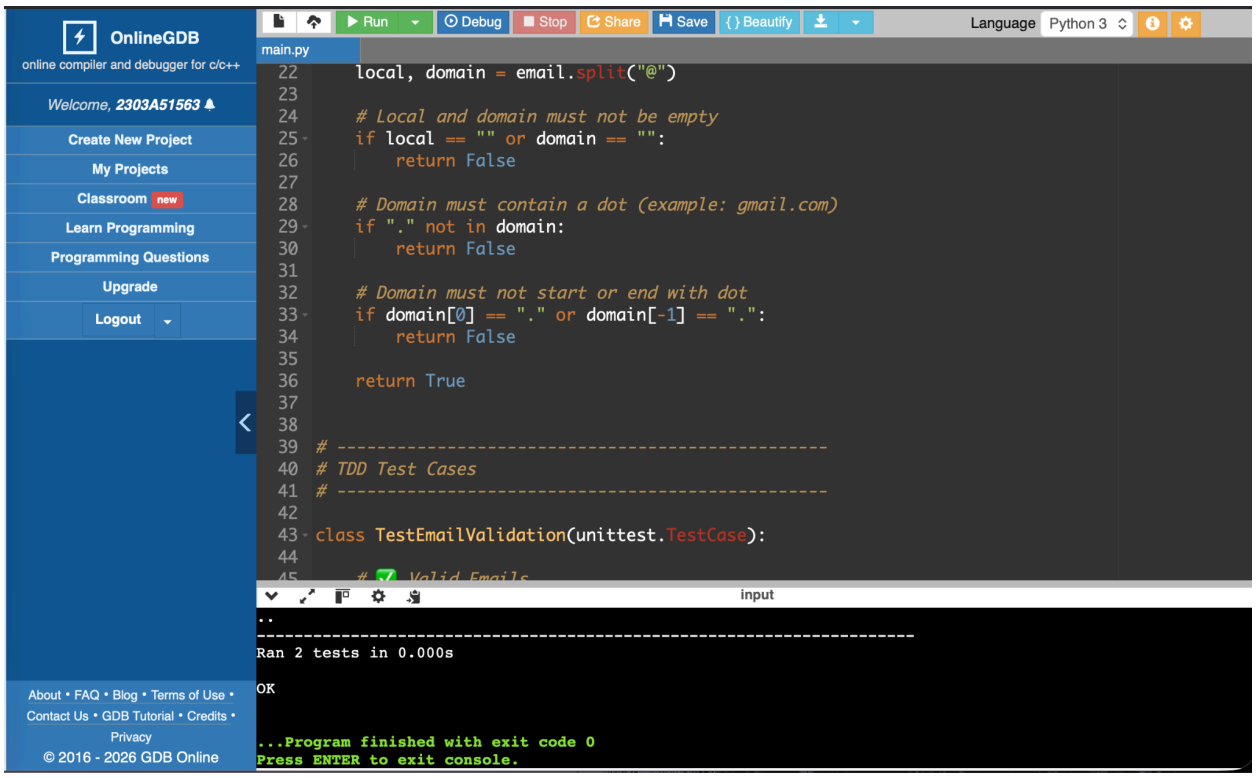
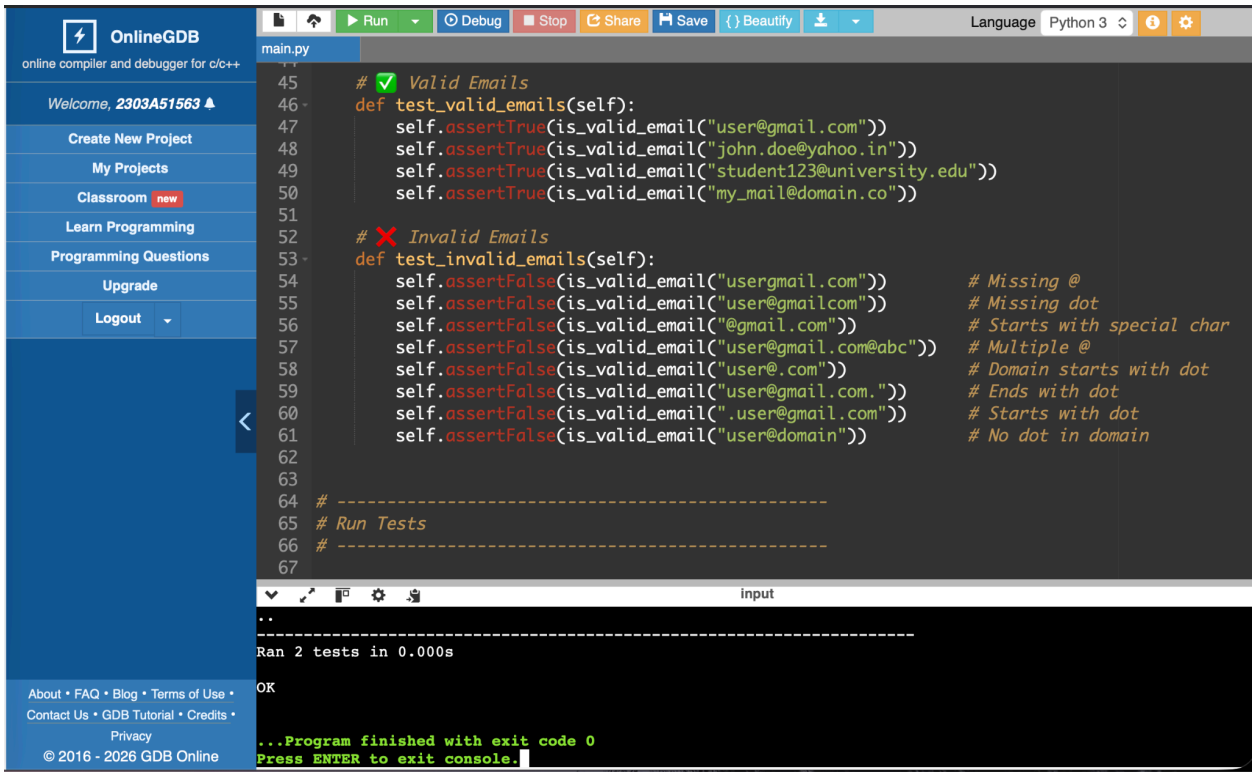
2303A51563
batch=10
Task 1: Email Validation using TDD
Scenario
You are developing a user registration system that requires reliable email input validation.
Requirements
• Must contain @ and . characters
• Must not start or end with special characters
• Should not allow multiple @ symbols
• AI should generate test cases covering valid and invalid email formats
• Implement is_valid_email(email) to pass all AI-generated test cases
Expected Output
• Python function for email validation
• All AI-generated test cases pass successfully
• Invalid email formats are correctly rejected
• Valid email formats return True
Code

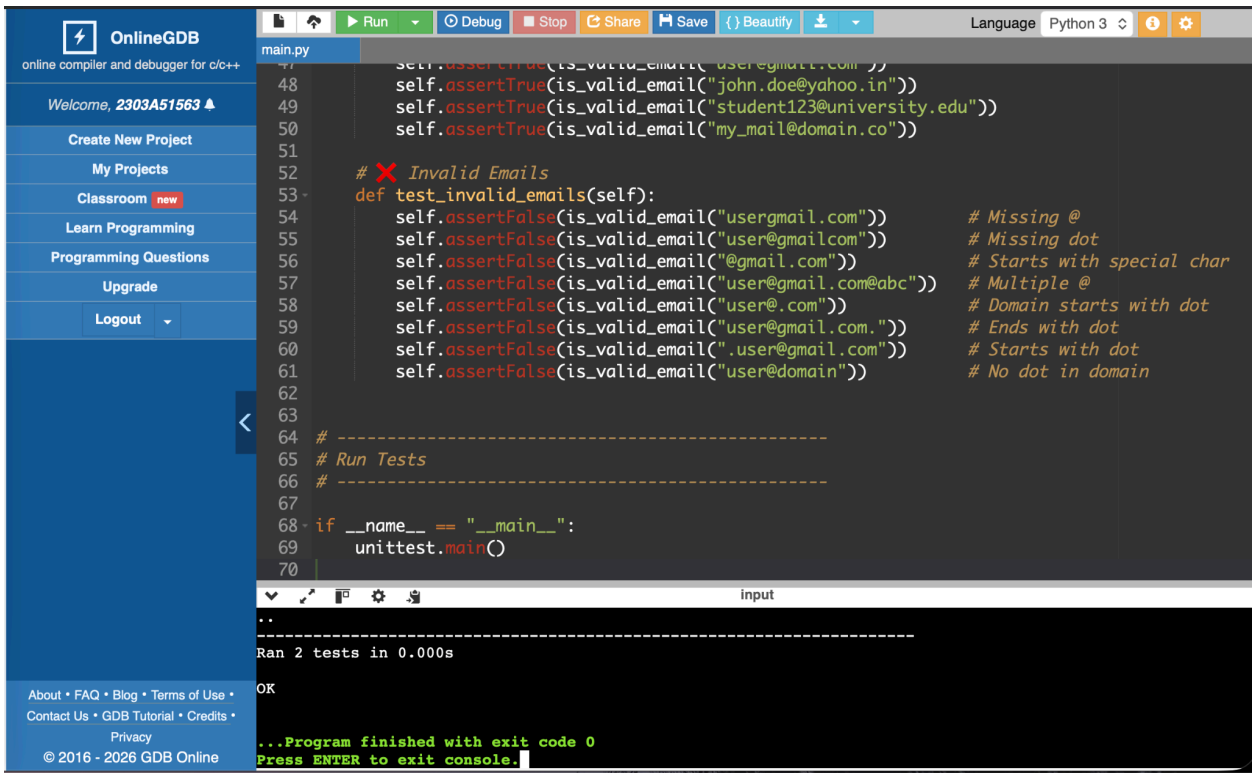
```
import unittest

# ------------------------------------------------
# Email Validation Function
# ------------------------------------------------

def is_valid_email(email):
    # Must contain exactly one '@'
    if email.count("@") != 1:
        return False

    # Must contain at least one '.'
    if "." not in email:
        return False

    # Must not start or end with special characters
    special_chars = "@._-"
    if email[0] in special_chars or email[-1] in special_chars:
        return False

    # Split into local part and domain part
    local, domain = email.split("@")

    # Local and domain must not be empty
    if local == "" or domain == "":
        return False
```

```
    # Domain must contain a dot (example: gmail.com)
    if "." not in domain:
        return False

    # Domain must not start or end with dot
    if domain[0] == "." or domain[-1] == ".":
        return False

    return True


# ----------------------------------------------
# TDD Test Cases
# ----------------------------------------------

class TestEmailValidation(unittest.TestCase):

    # ✅ Valid Emails
    def test_valid_emails(self):
        self.assertTrue(is_valid_email("user@gmail.com"))
        self.assertTrue(is_valid_email("john.doe@yahoo.in"))
        self.assertTrue(is_valid_email("student123@university.edu"))
        self.assertTrue(is_valid_email("my_mail@domain.co"))

    # ❌ Invalid Emails
    def test_invalid_emails(self):
        self.assertFalse(is_valid_email("usergmail.com"))       # Missing @
        self.assertFalse(is_valid_email("user@gmailcom"))       # Missing dot
        self.assertFalse(is_valid_email("@gmail.com"))          # Starts with special char
        self.assertFalse(is_valid_email("user@gmail.com@abc"))  # Multiple @
        self.assertFalse(is_valid_email("user@.com"))           # Domain starts with dot
        self.assertFalse(is_valid_email("user@gmail.com."))     # Ends with dot
        self.assertFalse(is_valid_email(".user@gmail.com"))     # Starts with dot
        self.assertFalse(is_valid_email("user@domain"))         # No dot in domain


# ----------------------------------------------
# Run Tests
# ----------------------------------------------

if __name__ == "__main__":
    unittest.main()
```


OnlineGDB
online compiler and debugger for c/c++
Welcome, 2303A51563
Create New Project
My Projects
Classroom new
Learn Programming
Programming Questions
Upgrade
Logout
Run
Debug
Stop
Share
Save
{ } Beautify
Language
Python 3
main.py
import unittest
# --
# Email Validation Function
# --
def is_valid_email(email):
# Must contain exactly one '@'
if email.count("@") != 1:
return False
# Must contain at least one '.'
if "." not in email:
return False
# Must not start or end with special characters
special_chars = "@._-"
if email[0] in special_chars or email[-1] in special_chars:
return False
Split into local part and domain part
local, domain = email.split("@")
input
..
Ran 2 tests in 0.000s
OK
...Program finished with exit code 0
Press ENTER to exit console.
About • FAQ • Blog • Terms of Use • Contact Us • GDB Tutorial • Credits • Privacy
© 2016 - 2026 GDB Online


OnlineGDB
online compiler and debugger for c/c++
Welcome, 2303A51563
Create New Project
My Projects
Classroom new
Learn Programming
Programming Questions
Upgrade
Logout
Run
Debug
Stop
Share
Save
{ } Beautify
Language
Python 3
main.py
local, domain = email.split("@")
# Local and domain must not be empty
if local == "" or domain == "":
return False
# Domain must contain a dot (example: gmail.com)
if "." not in domain:
return False
# Domain must not start or end with dot
if domain[0] == "." or domain[-1] == ".":
return False
return True
# --
# TDD Test Cases
# --
class TestEmailValidation(unittest.TestCase):
input
..
Ran 2 tests in 0.000s
OK
...Program finished with exit code 0
Press ENTER to exit console.
About • FAQ • Blog • Terms of Use • Contact Us • GDB Tutorial • Credits • Privacy
© 2016 - 2026 GDB Online





Task 2: Grade Assignment using Loops

Scenario

You are building an automated grading system for an online examination platform.

Requirements

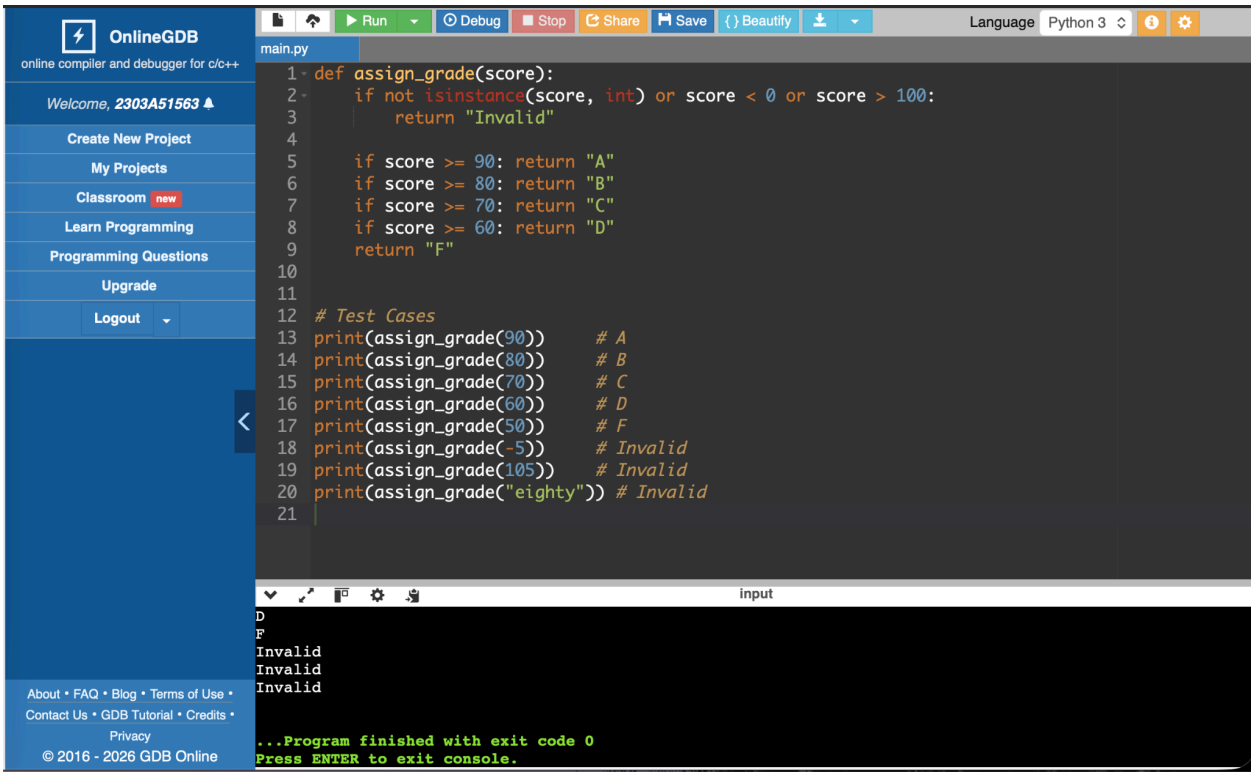
• AI should generate test cases for assign_grade(score) where:
– 90–100 → A
– 80–89 → B
– 70–79 → C
– 60–69 → D
– Below 60 → F
• Include boundary values (60, 70, 80, 90)
• Include invalid inputs such as -5, 105, "eighty"
• Implement the function using a test-driven approach
Expected Output
• Grade assignment function implemented in Python
• Boundary values handled correctly
• Invalid inputs handled gracefully
• All AI-generated test cases pass
Code

```
def assign_grade(score):
    if not isinstance(score, int) or score < 0 or score > 100:
        return "Invalid"

    if score >= 90: return "A"
    if score >= 80: return "B"
    if score >= 70: return "C"
    if score >= 60: return "D"
    return "F"


# Test Cases
print(assign_grade(90))     # A
print(assign_grade(80))     # B
print(assign_grade(70))     # C
print(assign_grade(60))     # D
print(assign_grade(50))     # F
print(assign_grade(-5))     # Invalid
print(assign_grade(105))    # Invalid
print(assign_grade("eighty")) # Invalid
```

Task 2: Grade Assignment using Loops
Scenario
You are building an automated grading system for an online examination platform.
Requirements
• AI should generate test cases for assign_grade(score) where:
– 90–100 → A
– 80–89 → B
– 70–79 → C

– 60–69 → D
– Below 60 → F
• Include boundary values (60, 70, 80, 90)
• Include invalid inputs such as -5, 105, "eighty"
• Implement the function using a test-driven approach
Expected Output
• Grade assignment function implemented in Python
• Boundary values handled correctly
• Invalid inputs handled gracefully
• All AI-generated test cases pass
Code

```
def assign_grade(score):
    if not isinstance(score, int) or score < 0 or score > 100:
        return "Invalid"

    if score >= 90: return "A"
    if score >= 80: return "B"
    if score >= 70: return "C"
    if score >= 60: return "D"
    return "F"


# Test Cases
print(assign_grade(90))     # A
print(assign_grade(80))     # B
print(assign_grade(70))     # C
print(assign_grade(60))     # D
print(assign_grade(50))     # F
print(assign_grade(-5))     # Invalid
print(assign_grade(105))    # Invalid
print(assign_grade("eighty")) # Invalid
```



Task 3: Sentence Palindrome Checker

Scenario

You are developing a text-processing utility to analyze sentences.

Requirements

• AI should generate test cases for is_sentence_palindrome(sentence)

• Ignore case, spaces, and punctuation

• Test both palindromic and non-palindromic sentences

• Example:

– "A man a plan a canal Panama" → True

Expected Output

• Function correctly identifies sentence palindromes

• Case and punctuation are ignored

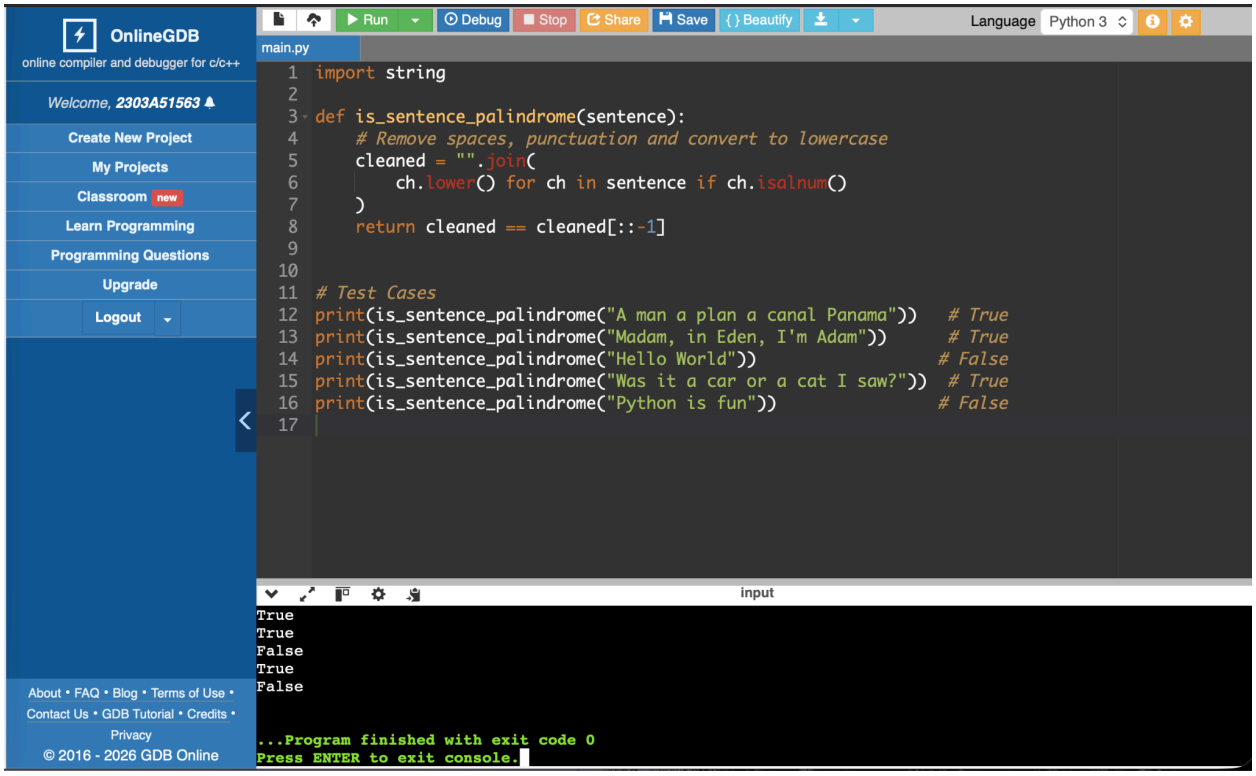
• Returns True or False accurately

• All AI-generated test cases pass

Code

```
import string

def is_sentence_palindrome(sentence):
    # Remove spaces, punctuation and convert to lowercase
    cleaned = "".join(
        ch.lower() for ch in sentence if ch.isalnum()
    )
    return cleaned == cleaned[::-1]
```

```
# Test Cases
print(is_sentence_palindrome("A man a plan a canal Panama"))   # True
print(is_sentence_palindrome("Madam, in Eden, I'm Adam"))      # True
print(is_sentence_palindrome("Hello World"))                   # False
print(is_sentence_palindrome("Was it a car or a cat I saw?"))  # True
print(is_sentence_palindrome("Python is fun"))                 # False
```



Task 4: ShoppingCart Class

Scenario

You are designing a basic shopping cart module for an e-commerce application.

Requirements

• AI should generate test cases for the ShoppingCart class

• Class must include the following methods:

– add_item(name, price)

– remove_item(name)

– total_cost()

• Validate correct addition, removal, and cost calculation

• Handle empty cart scenarios

Expected Output

• Fully implemented ShoppingCart class

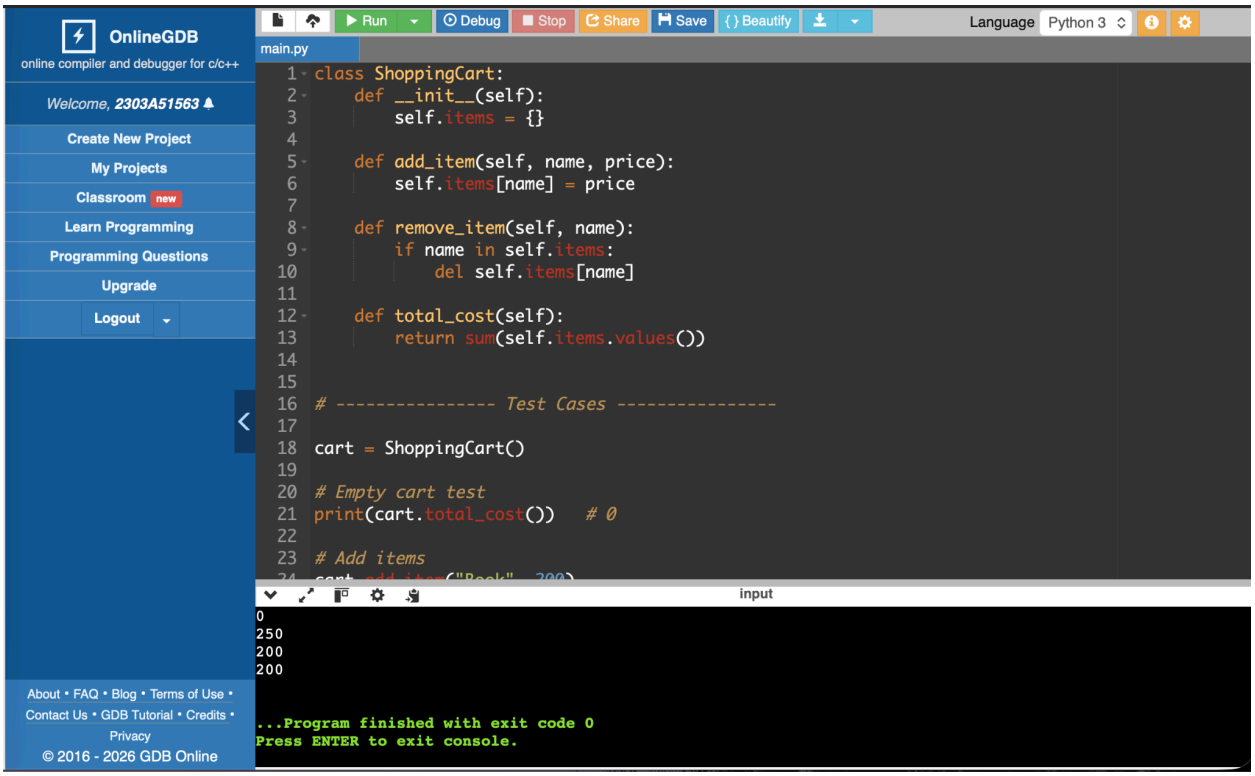
• All methods pass AI-generated test cases

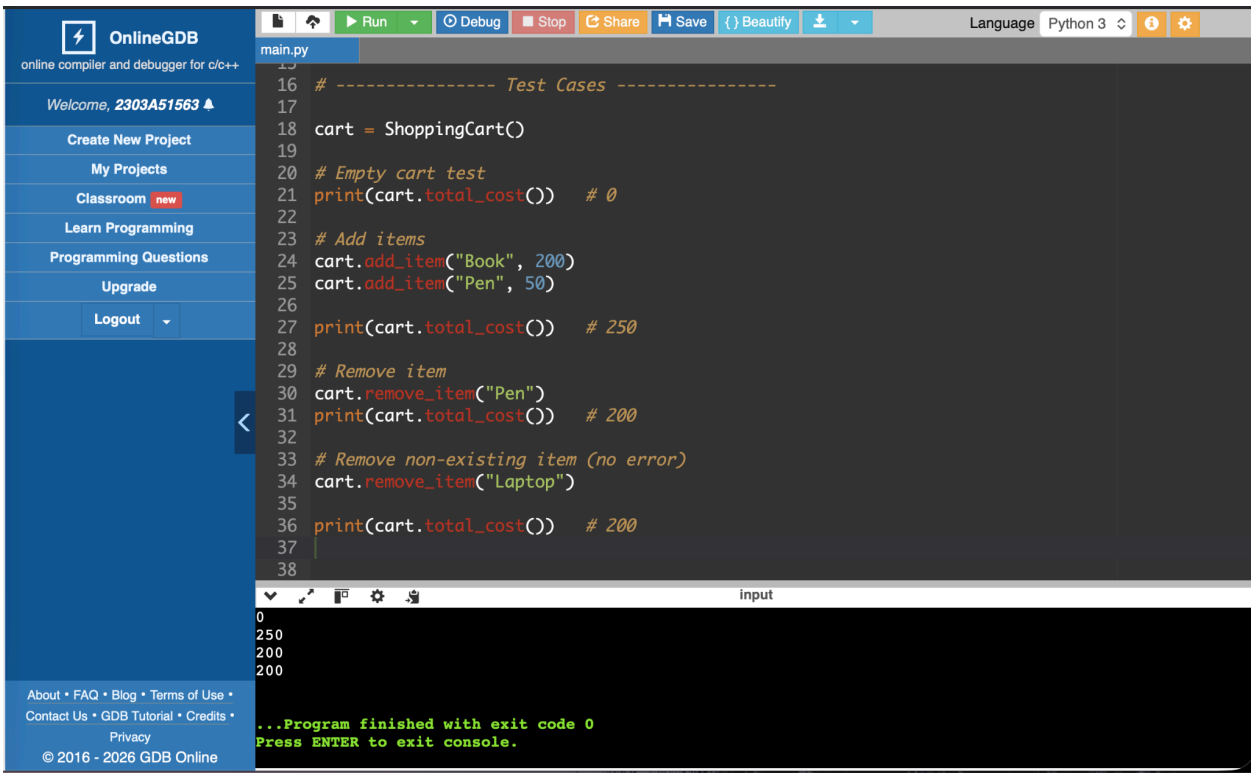
• Total cost is calculated accurately

• Items are added and removed correctly

Code

```
class ShoppingCart:
    def __init__(self):
        self.items = {}

    def add_item(self, name, price):
        self.items[name] = price

    def remove_item(self, name):
        if name in self.items:
            del self.items[name]

    def total_cost(self):
        return sum(self.items.values())


# ---------- Test Cases ----------

cart = ShoppingCart()

# Empty cart test
print(cart.total_cost())   # 0

# Add items
cart.add_item("Book", 200)
cart.add_item("Pen", 50)

print(cart.total_cost())   # 250

# Remove item
cart.remove_item("Pen")
print(cart.total_cost())   # 200

# Remove non-existing item (no error)
cart.remove_item("Laptop")

print(cart.total_cost())   # 200
```





Task 5: Date Format Conversion

Scenario

You are creating a utility function to convert date formats for reports.

Requirements

• AI should generate test cases for convert_date_format(date_str)
• Input format must be "YYYY-MM-DD"
• Output format must be "DD-MM-YYYY"
• Example:
– "2023-10-15" → "15-10-2023"
Expected Output
• Date conversion function implemented in Python
• Correct format conversion for all valid inputs
• All AI-generated test cases pass successfully
Code

```
def convert_date_format(date_str):
    # Split YYYY-MM-DD into parts
    y, m, d = date_str.split("-")
    return f"{d}-{m}-{y}"


# -------- Test Cases --------
print(convert_date_format("2023-10-15"))  # 15-10-2023
print(convert_date_format("2024-01-01"))  # 01-01-2024
print(convert_date_format("1999-12-31"))  # 31-12-1999
```