



manikandan26052004 / MODULE-8



Code

Pull requests

Actions

Projects

Wiki

Security

Insights

MODULE-8 / README.md



manikandan26052004 Create README.md

0808d9a · 8 minutes ago



223 lines (162 loc) · 6.74 KB

Hackerrank:# Student Topper Finder

This Python program helps determine the **top-performing student** based on the total marks across five subjects. It uses a dictionary to store each student's marks and identifies the topper using simple calculations and built-in functions.

Aim

To maintain a dictionary of students with their marks in five subjects, calculate their **total marks**, store them in a new dictionary, and identify the **student with the highest total (topper)**.

Algorithm

1. Start the program.
2. Create a dictionary `student_marks` :
 - o Keys → Student names.
 - o Values → List of marks in five subjects.
3. Initialize an empty dictionary `total_marks` .
4. Loop through `student_marks` :
 - o Calculate the total marks using `sum()` .
 - o Store the result in `total_marks` .
5. Use `max()` on `total_marks` to find the student with the highest total.

6. Print:

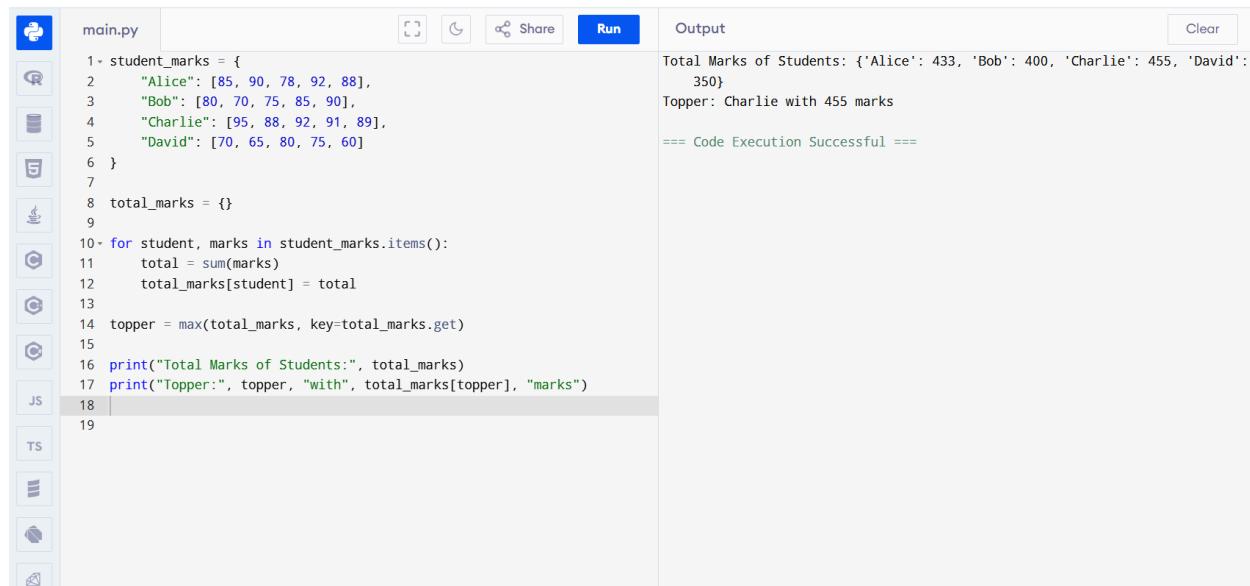
- o The `total_marks` dictionary.
- o The topper's name and score.



PROGRAM:

```
student_marks = {  
    "Alice": [85, 90, 78, 92, 88],  
    "Bob": [80, 70, 75, 85, 90],  
    "Charlie": [95, 88, 92, 91, 89],  
    "David": [70, 65, 80, 75, 60]  
}  
  
total_marks = {}  
  
for student, marks in student_marks.items():  
    total = sum(marks)  
    total_marks[student] = total  
  
topper = max(total_marks, key=total_marks.get)  
  
print("Total Marks of Students:", total_marks)  
print("Topper:", topper, "with", total_marks[topper], "marks")
```

OUTPUT



The screenshot shows a code editor interface with a Python file named `main.py`. The code calculates the total marks for each student and identifies the student with the highest marks as the topper. The output window displays the results of the execution.

```
main.py  
1- student_marks = {  
2     "Alice": [85, 90, 78, 92, 88],  
3     "Bob": [80, 70, 75, 85, 90],  
4     "Charlie": [95, 88, 92, 91, 89],  
5     "David": [70, 65, 80, 75, 60]  
6 }  
7  
8 total_marks = {}  
9  
10 for student, marks in student_marks.items():  
11     total = sum(marks)  
12     total_marks[student] = total  
13  
14 topper = max(total_marks, key=total_marks.get)  
15  
16 print("Total Marks of Students:", total_marks)  
17 print("Topper:", topper, "with", total_marks[topper], "marks")  
18  
19
```

Output:
Total Marks of Students: {'Alice': 433, 'Bob': 400, 'Charlie': 455, 'David': 350}
Topper: Charlie with 455 marks
== Code Execution Successful ==

RESULT

Thus, the program is executed successfully



Preview

Code

Blame

Raw



ensuring each line does not exceed a specified width.



Aim

To write a Python function that takes a long string and a specified width, and returns the string formatted with line breaks such that each line has **at most the given width**.



Algorithm

1. Start the program.
2. Define a function `wrap(string, max_width)` :
 - o Create an empty list `wrapped_lines` to store parts of the string.
 - o Loop through the string using steps of `max_width` .
 - o In each iteration, extract a substring of length `max_width` .
 - o Append this substring to the list.
3. Join the list with `\n` to create the final string.
4. Return the result.
5. End the program.



Program

```
def wrap(string, max_width):  
    wrapped_lines = []  
    for i in range(0, len(string), max_width):  
        wrapped_lines.append(string[i:i+max_width])  
    return '\n'.join(wrapped_lines)  
  
text = "This is a sample string that needs to be wrapped after a certain  
width."  
width = 10  
result = wrap(text, width)  
print(result)
```

Sample Output

The screenshot shows a code editor interface with a sidebar containing file icons for various languages: Python (selected), C, C++, C#, Java, JavaScript, and TypeScript. The main area displays Python code named 'main.py'.

```
1 def wrap(string, max_width):
2     wrapped_lines = []
3     for i in range(0, len(string), max_width):
4         wrapped_lines.append(string[i:i+max_width])
5     return '\n'.join(wrapped_lines)
6
7 text = "This is a sample string that needs to be wrapped after a
8         certain width."
9 width = 10
10 result = wrap(text, width)
11 print(result)
```

The code defines a function 'wrap' that takes a string and a maximum width as arguments. It iterates over the string in steps of 'max_width', appends each segment to a list, and then joins them with a newline character. The main part of the script creates a sample string, sets a width of 10, calls the 'wrap' function, and prints the result.

At the top right, there are buttons for copy, share, and run. The 'Run' button is highlighted in blue. To the right of the code area is a 'Output' panel. The output shows the wrapped string and a success message:

This is a
sample str
ing that n
eeds to be
wrapped a
fter a cer
tain width
. . .
== Code Execution Successful ==

Result

Thus, the program is executed successfully

Hackerrank:Python Program to Find Students with the Second Lowest Grade

This program reads student names and their corresponding grades, identifies the **second lowest grade**, and prints the names of all students who have that grade in alphabetical order.

Aim

To write a Python program to:

- Read a list of students and their grades.
- Identify the second lowest grade.
- Print the names of students who have that grade, sorted alphabetically.



Algorithm

1. Read an integer `n` representing the number of students.
 2. Read each student's name and grade, and store them as a sublist inside a list.
 3. Extract all the grades and sort them.
 4. Identify the second lowest grade from the sorted grade list.
 5. Collect names of all students whose grade matches the second lowest grade.
 6. Sort the names alphabetically.
 7. Print each name on a new line.
-



Program

```
n = int(input())
students = []

for _ in range(n):
    name = input()
    grade = float(input())
    students.append([name, grade])

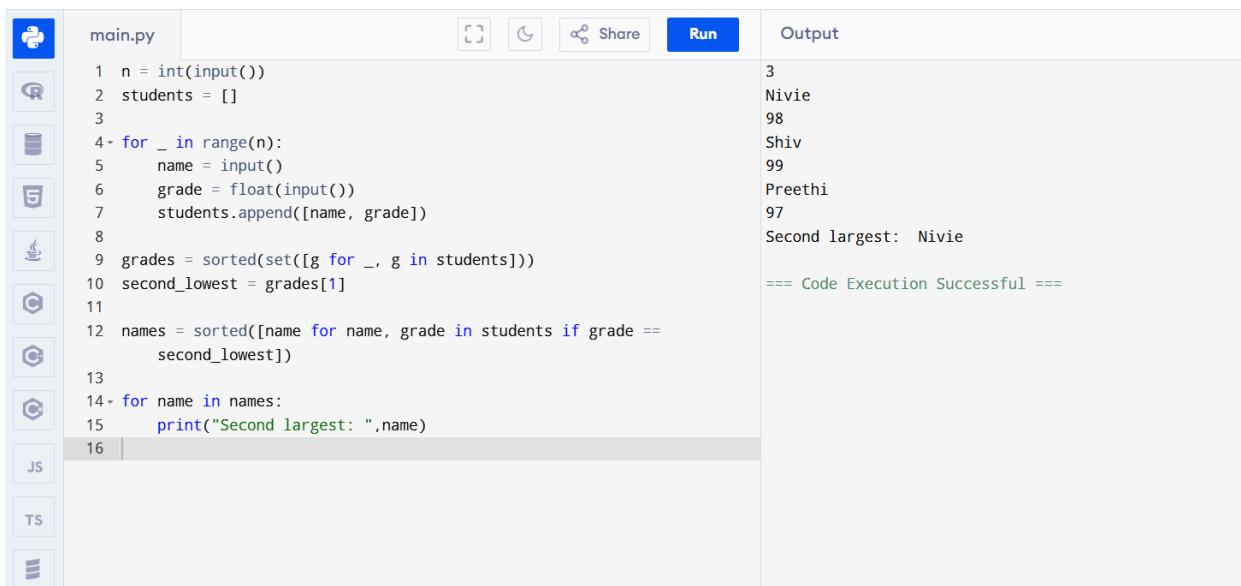
grades = sorted(set([g for _, g in students]))
second_lowest = grades[1]

names = sorted([name for name, grade in students if grade ==
second_lowest])

for name in names:
    print(name)
```



Output



```
main.py
1 n = int(input())
2 students = []
3
4 for _ in range(n):
5     name = input()
6     grade = float(input())
7     students.append([name, grade])
8
9 grades = sorted(set([g for _, g in students]))
10 second_lowest = grades[1]
11
12 names = sorted([name for name, grade in students if grade == second_lowest])
13
14 for name in names:
15     print("Second largest: ",name)
16
```

Output

```
3
Nivie
98
Shiv
99
Preethi
97
Second largest: Nivie
== Code Execution Successful ==
```

Result

Thus, the program is executed successfully

Hackerrank:Runner-Up Score Finder in Python

AIM:

To write a Python program that takes a list of scores from participants and finds the **runner-up score** (i.e., the second-highest score), eliminating any duplicates.

ALGORITHM:

1. Start
2. Create a variable `n` and get its value from the user (number of participants)
3. Read the list of `n` scores from the user using `input().split()` and convert them to integers
4. Store the scores in a list
5. Use `set()` to remove any duplicate scores
6. Convert the set back to a list and sort it in ascending order
7. Print the second-last element of the sorted list (i.e., the runner-up score)

8. Stop



PROGRAM:

```
n = int(input())
scores = list(map(int, input().split()))
unique_scores = list(set(scores))
unique_scores.sort()
print(unique_scores[-2])
```



OUTPUT

	main.py			Output
	1 n = int(input()) 2 scores = list(map(int, input().split())) 3 unique_scores = list(set(scores)) 4 unique_scores.sort() 5 print(unique_scores[-2])			5 2 3 6 7 4 6 ==== Code Execution Successful ===

RESULT

Thus, the program is executed successfully

Hackerrank:Python Program to Check if a String Ends with a Numeric Digit

This Python program checks whether the last character of a given input string is a numeric digit (0–9).

Aim

To write a Python program that checks if a given string ends with a number using Python's built-in string methods.

Algorithm

1. Start the program.
2. Input a string from the user.
3. Access the last character using indexing (`string[-1]`).
4. Check if the last character is a digit using the `.isdigit()` method.
5. If true, print that the string ends with a number.
6. Else, print that the string does not end with a number.
7. End the program.

Program

```
user_input = input("Enter a string: ")  
  
if len(user_input) > 0 and user_input[-1].isdigit():  
    print("The string ends with a number.")  
else:  
    print("The string does not end with a number.")
```



Output

The screenshot shows a code editor interface with a sidebar on the left containing icons for various file types: Python (selected), SQL, JSON, CSS, JS, and TS. The main area has tabs for 'main.py' and 'Output'. The code in 'main.py' is:

```
1 user_input = input("Enter a string: ")
2
3 if len(user_input) > 0 and user_input[-1].isdigit():
4     print("The string ends with a number.")
5 else:
6     print("The string does not end with a number.")
7
8
```

The 'Run' button is highlighted in blue. The 'Output' tab shows the results of running the code:

```
Enter a string: star6
The string ends with a number.

==== Code Execution Successful ===
```

Result

Thus, the program is executed successfully