

The screenshot shows a code editor interface with a Python script named `Nguyen_Albert_and_Nguyen_Thomas_Lab4.py`. The code is divided into two sections: `# Lab 4a` and `# Lab 4b`.

```
# Due to PEP8, 2 blank lines at start and end of functions in labs below, also variable in functions in camelCase
# Lab 4a
def print_multiples_of_5_10(n): # parameter1 is passing argument from variable n in outer scope
    numberToTest = 1 # numberToTest starts at 1 to exclude negative numbers and 0
    print("Printing all multiples of 5 or 10 less than %d:" % n)
    while numberToTest < n:
        if numberToTest % 5 == 0 or numberToTest % 10 == 0: # check if numberToTest is multiple of 5 or 10
            print(numberToTest)
        numberToTest = numberToTest + 1 # or numberToTest += 1, go to next array: numberToTest[0]==>[1]==>[2]==>[n-1]<n
    return numberToTest # return value won't be printed

n = 20 # variable n or given number is the upper limit which loop stops at once while loop increment finishes
print_multiples_of_5_10(n) # With n = 20
print_multiples_of_5_10(30) # With n = 30

# Lab 4b
def vowel_count(string): # to keep vowelCounter and stringArrayNum local, default para is string
    vowelCounter = 0 # start at count 0
    stringArrayNum = [0, 0, 0, 0, 0] # start at 0, 0, 0, 0, 0
```

The code editor has various toolbars and panels visible, including a Project panel, a Git panel, and a Database panel. The status bar at the bottom provides information about the file, including its path, encoding, and version control details.

```
C:\Python311\python.exe "D:\github.com\anguyen798\cs131-47853-homework\Chapter 4\Nguyen_Albert_and_Nguyen_Thomas_Lab4.py"
Run - Nguyen_Albert_and_Lastname2_FirstName2_Lab4.py
Printing all multiples of 5 or 10 less than 20:
5
10
15
Printing all multiples of 5 or 10 less than 30:
5
10
15
20
25

Process finished with exit code 0
```

The screenshot shows a code editor interface with a Python script named `Nguyen_Albert_and_Nguyen_Thomas_Lab4.py`. The code defines a function `vowel_count` that takes a string as input and returns the count of vowels ('a', 'e', 'i', 'o', 'u'). It uses a while loop to iterate through the string, checking each character against a set of vowels. The code also includes examples of calling the function with user input and using an f-string to print the result.

```
18 # Lab 4b
19 def vowel_count(string): # to keep vowelCounter and stringArrayNum local, default para is string
20     vowelCounter = 0 # start at count 0
21     stringArrayNum = 0 # start at string[0] [Array=0]
22     vowels = "aeiouAEIOU"
23     while stringArrayNum < len(string): # the length of the string, i.e. len("foo") = 3, loop foo[0],foo[1],foo[2]
24         if string[stringArrayNum] in vowels: # i.e. vowelCounter = 0, foo[0] = f, foo[1] = o, vowelCounter += 1
25             vowelCounter = vowelCounter + 1 # or vowelCounter += 1, add to
26             # or stringArrayNum += 1: after executing if statement(vowelCounter), look at next stringArray[0] => [1] => [2]
27             stringArrayNum = stringArrayNum + 1
28             # print statement commented out in function to practice assigning function to return values:
29             # print("The string \"%s\" has %d vowels." % (parameter1, vowelCounter)) # commented out
30     return vowelCounter # return the vowel count to be assigned outside function
31
32
33 # Default, i.e. string = input("Albert-o")
34 string = input("Enter a string: ")
35 vowelCounter = vowel_count(string) # call function on string and assign vowel counter to function output, default para
36 print("The string \"%s\" has %d vowels." % (string, vowelCounter)) # format vowelCounter with integer %d specifier
37
38 # Bypassing the input() function with string = "Thomas"
39 string = "Thomas"
40 vowelCounter = vowel_count(string) # call function, must explicitly pass variable "string" as parameter
41 print(f"The string \"{string}\" has {vowelCounter} vowels.") # using f string, \ to escape the double quotes
```

At the bottom of the screen, there are various toolbars and status indicators, including Git, Run, Python Packages, TODO, Python Console, Problems, Terminal, Evernote, Services, Notes, and GitHub integration. The status bar also shows file statistics like CRLF, UTF-8, 4 spaces, and blame information.



```
Run - Nguyen_Albert_and_Lastname2_FirstName2_Lab4.py
Run: Nguyen_Albert_and_Nguyen_Thomas_Lab4 ×
C:\Python311\python.exe "D:\github.com\anguyen798\cs131-47853-homework\Chapter 4\Nguyen_Albert_and_Nguyen_Thomas_Lab4.py"
Enter a string: Albert-0
The string "Albert-0" has 3 vowels.
The string "Thomas" has 2 vowels.

Process finished with exit code 0
```

4b_Output

4b_Output

The screenshot shows a code editor interface with a Python script named `Nguyen_Albert_and_Nguyen_Thomas_Lab4.py`. The code implements a function `grades_compute()` to calculate statistics from user input grades. It uses a sentinel-controlled loop to read grades until -1 is entered. The program then calculates the average grade, counts passing and failing grades, and prints the results.

```
48 # Lab 4c
49 def grades_compute():
50     # Set all starting values to empty list or 0
51     grades = []
52     grade = 0
53     passing_grades = 0
54     failing_grades = 0
55     sum_of_grades = 0
56
57     while grade != -1:
58         grade = float(input("Enter a grade or -1 to finish: ")) # -1 is sentinel to end while loop
59         if grade != -1:
60             grades.append(grade) # .append method to add grade to the grades list
61             sum_of_grades = sum_of_grades + grade
62             if grade >= 60:
63                 passing_grades = passing_grades + 1
64             else:
65                 failing_grades = failing_grades + 1
66
67     if len(grades) == 0: # if condition to show no grades were entered
68         print("No grades were entered.")
69     else:
70         avg_grade = sum_of_grades / len(grades) # find the average of grades list
71         max_grade = max(grades) # find min value in grades list
72         min_grade = min(grades) # find max value in grades list
73
74         print("The average grade is %.2f" % avg_grade)
75         print("Number of passing grades is %d" % passing_grades)
76         print("Number of failing grades is %d" % failing_grades)
77         print("The maximum grade is %.2f" % max_grade)
78         print("The minimum grade is %.2f" % min_grade)
79
80
81     grades_compute()
```

At the bottom of the screen, there are several status indicators: Git (up-to-date), Python Packages, TODO, Python Console, Problems, Terminal, Evernote, Services, Notes, and a message about externally added files. The status bar also shows file statistics like 109:1 CRLF, UTF-8, 4 spaces, and the current file path: `Nguyen_Albert_and_Nguyen_Thomas_Lab4.py`.

```
C:\Python311\python.exe "D:\github.com\anguyen798\cs131-47853-homework\Chapter 4\Nguyen_Albert_and_Nguyen_Thomas_Lab4.py"
Run - Nguyen_Albert_and_Lastname2_FirstName2_Lab4.py
Run: Nguyen_Albert_and_Nguyen_Thomas_Lab4 ×
Enter a grade or -1 to finish: 70
Enter a grade or -1 to finish: 55
Enter a grade or -1 to finish: 80
Enter a grade or -1 to finish: 90
Enter a grade or -1 to finish: 66
Enter a grade or -1 to finish: 40
Enter a grade or -1 to finish: -1
The average grade is 66.83
Number of passing grades is 4
Number of failing grades is 2
The maximum grade is 90.00
The minimum grade is 40.00
```

Process finished with exit code 0

The screenshot shows a code editor interface with a Python script named `Nguyen_Albert_and_Nguyen_Thomas_Lab4.py`. The code implements a function to calculate compound interest for a given number of years. It uses a while loop to iterate through each year, updating the balance by adding the interest rate to the current balance. The code also includes a for loop alternative and user input for the number of years.

```
83
84 # Lab 4d
85 def compound_interest(numYears): # calculate and print balance for each year
86     INTEREST_RATE = 0.05
87     INITIAL_BALANCE = 10000.00
88     print("For Initial Balance of ${:10.2f} at {:.2f}%, for {} years, the balance of each year is as follows: "
89           .format(INITIAL_BALANCE, INTEREST_RATE * 100, numYears)) # multi-line print statement due to line length
89
90     balance = INITIAL_BALANCE #
91     currentYear = 0
92
93     while currentYear < numYears: #
94         balance = balance + balance * INTEREST_RATE
95         currentYear = currentYear + 1 # start printing at year 1
96         print("For Year: {}d, Balance is ${:10.2f}" .format(currentYear, balance)) # print balance for each year
96         print("*" * 30) # Separator between function calls
97
98     # Using For loop:
99     # for currentYear in range(1, numYears + 1):
100    #     balance = balance + balance * INTEREST_RATE
100    #     print("For Year: {}d, Balance is ${:10.2f}" .format(currentYear, balance))
101
102
103 numYears = int(input("Enter number of years: "))
104 compound_interest(numYears) # Using input(), example: 5 years
105 compound_interest(2) # 2 years
106
107
```

At the bottom of the screen, there are various status icons and toolbars for the code editor, including Git, Python Console, Terminal, and GitHub integration.

Run - Nguyen_Albert_and_Lastname2_FirstName2_Lab4.py

scratch

```
C:\Python311\python.exe C:/Users/user/AppData/Roaming/JetBrains/PyCharm2022.3/scratches/scratch.py
```

Enter number of years: 5

For Initial Balance of \$ 10000.00 at 5.00%, for 5 years, the balance of each year is as follows:

For Year: 1, Balance is \$ 10500.00

For Year: 2, Balance is \$ 11025.00

For Year: 3, Balance is \$ 11576.25

For Year: 4, Balance is \$ 12155.06

For Year: 5, Balance is \$ 12762.82

For Initial Balance of \$ 10000.00 at 5.00%, for 2 years, the balance of each year is as follows:

For Year: 1, Balance is \$ 10500.00

For Year: 2, Balance is \$ 11025.00

Process finished with exit code 0

The screenshot shows a code editor interface with two Python files open: `Nguyen_Albert_and_Nguyen_Thomas_Lab4.py` and `Nguyen_Albert_and_Nguyen_Thomas_Lab4e.py`. The sidebar includes tabs for Project, Pull Requests, Commit, and Bookmarks. The main area displays the following code:

```
107
108 # Lab 4e
109 def bracket_print(rows=3, columns=4): # keyword parameter, default rows = 3 and columns = 4
110     print("Printing %d rows and %d columns" % (rows, columns))
111     for row in range(rows): # Range function no start parameter=start[0], stop at row 3 (array[3])
112         for column in range(columns): # Outer-row-loop only runs after nested-column-loop completes
113             print("[", end="")
114         print() # Go to next row after nested-column-loop runs and brackets are all finished printing on same row
115     # Then continue outer row for loop - print statement will execute at end of every interation of outer-row-loop
116
117
118 bracket_print() # Using default parameter values: 3 rows, 4 columns
119 bracket_print(6, 2) # Custom parameter value: 6 rows, 2 columns
120
121
122 # Lab 4f
123 def multiplication_table(n):
124     print("----" + "-" * 5 * n) # Print dash seperator: 5 dashes and 5 dashes for each n
125     print("For a multiplication table of %d numbers:" % n) # Description of how many numbers to multiply
126     print(" " * 2 + "|", end="")
127     for column_number_header in range(1, n + 1): # Start at 1 (array[0]) and stop before number+1, n (array[n+1])
128         print("%5d" % column_number_header, end="")
129     print() # After for loop, print new row
130     print("----" + "-" * 5 * n) # Print dash seperator: 5 dashes and 5 dashes for each n
131     for row_number_header in range(1, n + 1): # Print table: for loop = row header, nested for loop = multiplied values
```

The bottom status bar shows various icons and information, including Git status, file blame, and GitHub integration.

Run - Nguyen_Albert_and_Lastname2_FirstName2_Lab4.py

Run: Nguyen_Albert_and_Nguyen_Thomas_Lab4

```
C:\Python311\python.exe "D:\github.com\anguyen798\cs131-47853-homework\Chapter 4\Nguyen_Albert_and_Nguyen_Thomas_Lab4.py"

Printing 3 rows and 4 columns
[] []
[] []
[] []

Printing 6 rows and 2 columns
[] []
[] []
[] []
[] []
[] []
[] []

Process finished with exit code 0
```

The screenshot shows a code editor interface with a Python script named `Nguyen_Albert_and_Nguyen_Thomas_Lab4.py`. The code is organized into two main sections: `# Lab 4f` and `# Lab 4g`.

Lab 4f:

```
121
122     # Lab 4f
123
124     def multiplication_table(n):
125         print("----" + "-" * 5 * n) # Print dash separator: 5 dashes and 5 dashes for each n
126         print("For a multiplication table of %d numbers:" % n) # Description of how many numbers to multiply
127         print(" " * 2 + "|", end="")
128         for column_number_header in range(1, n + 1): # Start at 1 (array[0]) and stop before number+1, n (array[n+1])
129             print("%5d" % column_number_header, end="")
130         print() # After for loop, print new row
131         print("----" + "-" * 5 * n) # Print dash separator: 5 dashes and 5 dashes for each n
132         for row_number_header in range(1, n + 1): # Print table: for loop = row header, nested for loop = multiplied values
133             print("%3d" % row_number_header, end="")
134             for multiplier in range(1, n + 1): # Before each row header is created, nested-for-loop values on same line
135                 print("%5d" % (row_number_header * multiplier), end="")
136             print() # After nested-for-loop prints on same line, start a new row
137
138     n = int(input("Enter n: ")) # Using input, example: Multiplication table of 10 values, variable n shared with 4a
139     multiplication_table(n)
140     multiplication_table(5) # Multiplication table of 5 values
141
142
143     # Lab 4g
144
145     def string_counter(string):
146         print("-" * 30) # Header separator print statement
```

Lab 4g:

```
143     # Lab 4g
144
145     def string_counter(string):
146         print("-" * 30) # Header separator print statement
```

The code includes several print statements using f-strings and formatted string methods like `%d` and `%5d` to generate multiplication tables. It also demonstrates the use of nested loops and the `int()` function for user input.

```
Run - Nguyen_Albert_and_Lastname2_FirstName2_Lab4.py
Run: Nguyen_Albert_and_Nguyen_Thomas_Lab4
C:\Python311\python.exe "D:\github.com\anguyen798\cs131-47853-homework\Chapter 4\Nguyen_Albert_and_Nguyen_Thomas_Lab4.py"
Enter n: 10
-----
For a multiplication table of 10 numbers:
| 1 2 3 4 5 6 7 8 9 10
-----
1 1 2 3 4 5 6 7 8 9 10
2 2 4 6 8 10 12 14 16 18 20
3 3 6 9 12 15 18 21 24 27 30
4 4 8 12 16 20 24 28 32 36 40
5 5 10 15 20 25 30 35 40 45 50
6 6 12 18 24 30 36 42 48 54 60
7 7 14 21 28 35 42 49 56 63 70
8 8 16 24 32 40 48 56 64 72 80
9 9 18 27 36 45 54 63 72 81 90
10 10 20 30 40 50 60 70 80 90 100
-----
For a multiplication table of 5 numbers:
| 1 2 3 4 5
-----
1 1 2 3 4 5
2 2 4 6 8 10
3 3 6 9 12 15
4 4 8 12 16 20
5 5 10 15 20 25

```

Process finished with exit code 0

The screenshot shows a code editor interface with a Python script named `Nguyen_Albert_and_Nguyen_Thomas_Lab4.py`. The code defines a function `string_counter` that takes a string as input and counts the number of letters, digits, and symbols. It uses `if`, `elif`, and `else` statements to categorize each character. The script then prints the total counts for each category. At the bottom, it demonstrates the function by calling it with two strings: "Thomas1337" and "Nguyen798".

```
143 # Lab 4g
144 def string_counter(string):
145     print("-" * 30) # Header separator print statement
146     print("Below are the character counts for the string %s:" % string) # Description of string for function calls
147     # Set all count starting values to 0
148     letter_count = 0
149     digit_count = 0
150     symbol_count = 0
151
152     for character in string: # start with string[0] then end at string[len(string)]
153         if character.isalpha(): #
154             letter_count = letter_count + 1
155         elif character.isdigit():
156             digit_count = digit_count + 1
157         else: # for characters that aren't letters or numbers such as !@#$%^&*()_-+=[{ }]|\`~,<,>/?,...
158             symbol_count = symbol_count + 1
159
160     print("Total counts of Chars = %d" % letter_count) # Chars stand for letters
161     print("Total counts of Digits = %d" % digit_count)
162     print("Total counts of Symbol = %d" % symbol_count)
163
164
165     string = input("Enter a string: ") # input(), example: string="Thomas1337"
166     string_counter(string)
167     string_counter("Nguyen798")
168
```

At the bottom of the screen, there are various status icons and tabs, including Git, Python Packages, TODO, Problems, Terminal, Evernote, Services, Notes, and GitHub (Material). The status bar also shows file statistics like 184:1 CRLF UTF-8 4 spaces, and the current file name Nguyen_Albert_and_Nguyen_Thomas_Lab4.py.

```
Run - Nguyen_Albert_and_Lastname2_FirstName2_Lab4.py
Run: Nguyen_Albert_and_Nguyen_Thomas_Lab4

C:\Python311\python.exe "D:\github.com\anguyen798\cs131-47853-homework\Chapter 4\Nguyen_Albert_and_Nguyen_Thomas_Lab4.py"
Enter a string: P@#yn26at^&i5ve
-----
Below are the character counts for the string P@#yn26at^&i5ve:
Total counts of Chars = 8
Total counts of Digits = 3
Total counts of Symbol = 4
-----
Below are the character counts for the string Nguyen798:
Total counts of Chars = 7
Total counts of Digits = 3
Total counts of Symbol = 0
```

Process finished with exit code 0

The screenshot shows a code editor interface with a Python script named `Nguyen_Albert_and_Nguyen_Thomas_Lab4.py`. The code defines a function `roll_dice` that rolls two dice and prints their values. It also includes calls to `roll_dice` with arguments 10 and 5.

```
165
166 # Lab 4h
167 from random import randint # import statement usually at top, import only randint module from random
168 def roll_dice(numRolls=10): # default 10 rolls
169     print("The dice is rolled %d times" % numRolls)
170     for roll in range(numRolls): # start at roll[0] increment by 1,... range function stops at position numRolls[]
171         dieOne = randint(1, 6) # random integer between 1 and 6
172         dieTwo = randint(1, 6)
173         print(dieOne, dieTwo)
174     return dieOne, dieTwo # return value won't be printed, example of returning 2 values
175
176
177 roll_dice()
178 roll_dice(5) # 5 rolls
179
```

At the bottom of the screen, there is a toolbar with various icons for Git, Run, Python Packages, TODO, Python Console, Problems, Terminal, Evernote, Services, Notes, and a color scheme selector. The status bar at the very bottom displays file statistics and the current date and time.

The screenshot shows a Jupyter Notebook interface with a yellow sidebar on the left containing various icons. The main area displays the output of a Python script. The output starts with the command run: "C:\Python311\python.exe" followed by the path to the script file. Below this, the text "The dice is rolled 10 times" is displayed, followed by ten pairs of numbers representing dice rolls: 5 1, 6 3, 3 6, 2 2, 4 4, 2 2, 6 3, 3 3, 5 5, and 4 1. After a blank line, the text "The dice is rolled 5 times" is shown, followed by five pairs of numbers: 6 1, 6 2, 5 1, 6 3, and 2 6. At the bottom, the message "Process finished with exit code 0" is printed.

```
Run - Nguyen_Albert_and_Lastname2_FirstName2_Lab4.py
Run: Nguyen_Albert_and_Nguyen_Thomas_Lab4 x
C:\Python311\python.exe "D:\github.com\anguyen798\cs131-47853-homework\Chapter 4\Nguyen_Albert_and_Nguyen_Thomas_Lab4.py"
The dice is rolled 10 times
5 1
6 3
3 6
2 2
4 4
2 2
6 3
3 3
5 5
4 1
The dice is rolled 5 times
6 1
6 2
5 1
6 3
2 6
Process finished with exit code 0
```

4h_Output

4h_Output