**ASSIGNMENT – 3**

**Task-1:** Write a program in python to find compound interest using functions.

**Code:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Output:**

****

**Explanation:**

The code in the cell defines a Python function called compound\_interest that calculates the compound interest earned on an investment.

Here's a breakdown of the code:

* **def compound\_interest(principal, rate, time, n=1):**: This line defines the function named compound\_interest which takes four arguments:
  + principal: The initial amount of money invested.
  + rate: The annual interest rate (expressed as a decimal, e.g., 5% is 0.05).
  + time: The number of years the money is invested for.
  + n: The number of times the interest is compounded per year. It has a default value of 1, meaning annual compounding if not specified.
* **"""Docstring"""**: This is a docstring, which explains what the function does, its arguments, and what it returns.
* **amount = principal \* (1 + rate/n) \*\* (n\*time)**: This is the core formula for calculating the total amount after compound interest.
  + (1 + rate/n): This calculates the growth factor per compounding period. The annual rate is divided by the number of compounding periods per year.
  + \*\*(n\*time): This raises the growth factor to the power of the total number of compounding periods over the investment time.
  + principal \* ...: This multiplies the initial principal by the total growth factor to get the final amount.
* **interest = amount - principal**: This line calculates the compound interest earned by subtracting the initial principal from the final amount.
* **return interest**: The function returns the calculated compound interest.
* **principal\_amount = 1000**: This line sets the value for the principal amount in the example usage.
* **interest\_rate = 0.05**: This line sets the value for the annual interest rate in the example usage.
* **investment\_time = 3**: This line sets the value for the investment time in the example usage.
* **compounding\_frequency = 12**: This line sets the value for the number of times interest is compounded per year (monthly in this case).
* **calculated\_interest = compound\_interest(principal\_amount, interest\_rate, investment\_time, compounding\_frequency)**: This line calls the compound\_interest function with the specified example values and stores the returned interest in the calculated\_interest variable.
* **print(f"Compound interest after {investment\_time} years with monthly compounding: {calculated\_interest:.2f}")**: This line prints the calculated compound interest, formatted to two decimal places.

In summary, the code defines a flexible function to calculate compound interest for various compounding frequencies and then demonstrates its usage with an example of monthly compounding over 3 years.

**Task-2:** Write a function to calculate average,median, and mode of a list of numbers.

**Code:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Output:**

**A number and numbers on a white background

AI-generated content may be incorrect.**

**Explanation:**

The code in the cell defines a Python function called calculate\_stats that computes the average, median, and mode of a given list of numbers.

Here's a breakdown of the code:

* **import statistics**: This line imports the built-in Python module statistics, which provides functions for calculating common statistical measures.
* **def calculate\_stats(numbers):**: This line defines the function named calculate\_stats which takes one argument:
  + numbers: A list of numbers for which you want to calculate the statistics.
* **"""Docstring"""**: This is a docstring explaining the function's purpose, arguments, and return value.
* **if not numbers:**: This checks if the input list numbers is empty.
  + **return {"average": None, "median": None, "mode": None}**: If the list is empty, it returns a dictionary with None values for average, median, and mode, as these statistics cannot be calculated for an empty list.
* **average = statistics.mean(numbers)**: This line calculates the arithmetic mean (average) of the numbers in the list using the statistics.mean() function and stores it in the average variable.
* **median = statistics.median(numbers)**: This line calculates the median (the middle value) of the numbers in the list using the statistics.median() function and stores it in the median variable.
* **try...except statistics.StatisticsError:**: This is a try-except block to handle potential errors when calculating the mode.
  + **mode = statistics.mode(numbers)**: This line attempts to calculate the mode (the most frequent value) of the numbers in the list using the statistics.mode() function.
  + **except statistics.StatisticsError:**: This block is executed if the statistics.mode() function raises a StatisticsError. This error occurs when there is no unique mode (i.e., multiple values appear with the same highest frequency).
  + **mode = "No unique mode"**: If a StatisticsError occurs, the mode variable is set to the string "No unique mode".
* **return {"average": average, "median": median, "mode": mode}**: The function returns a dictionary containing the calculated average, median, and mode.
* **data = [1, 2, 3, 4, 5, 5, 6, 7, 8, 9, 9, 9]**: This line creates a list of numbers for the first example.
* **stats = calculate\_stats(data)**: This line calls the calculate\_stats function with the data list and stores the returned dictionary of statistics in the stats variable.
* **print(f"Data: {data}")**: This line prints the first data list.
* **print(f"Statistics: {stats}")**: This line prints the calculated statistics for the first data list.
* **data\_with\_no\_unique\_mode = [1, 2, 3, 4, 5, 5, 6, 6]**: This line creates a list of numbers for the second example, which has no unique mode.
* **stats\_no\_mode = calculate\_stats(data\_with\_no\_unique\_mode)**: This line calls the calculate\_stats function with the data\_with\_no\_unique\_mode list and stores the returned dictionary in the stats\_no\_mode variable.
* **print(f"\nData: {data\_with\_no\_unique\_mode}")**: This line prints the second data list.
* **print(f"Statistics: {stats\_no\_mode}")**: This line prints the calculated statistics for the second data list.

In essence, the code provides a reusable function to get key statistical measures from a list of numbers, handling the case where there isn't a single most frequent number.

**Task-3:** Provide multiple examples of input-output to the AI for convert\_to\_binary(num) function.

**Code:**

A screenshot of a computer

AI-generated content may be incorrect.

**Output:**

**A black text on a white background

AI-generated content may be incorrect.**

**Explanation:**

The code defines a Python function called convert\_to\_binary that converts a decimal integer to its binary representation as a string.

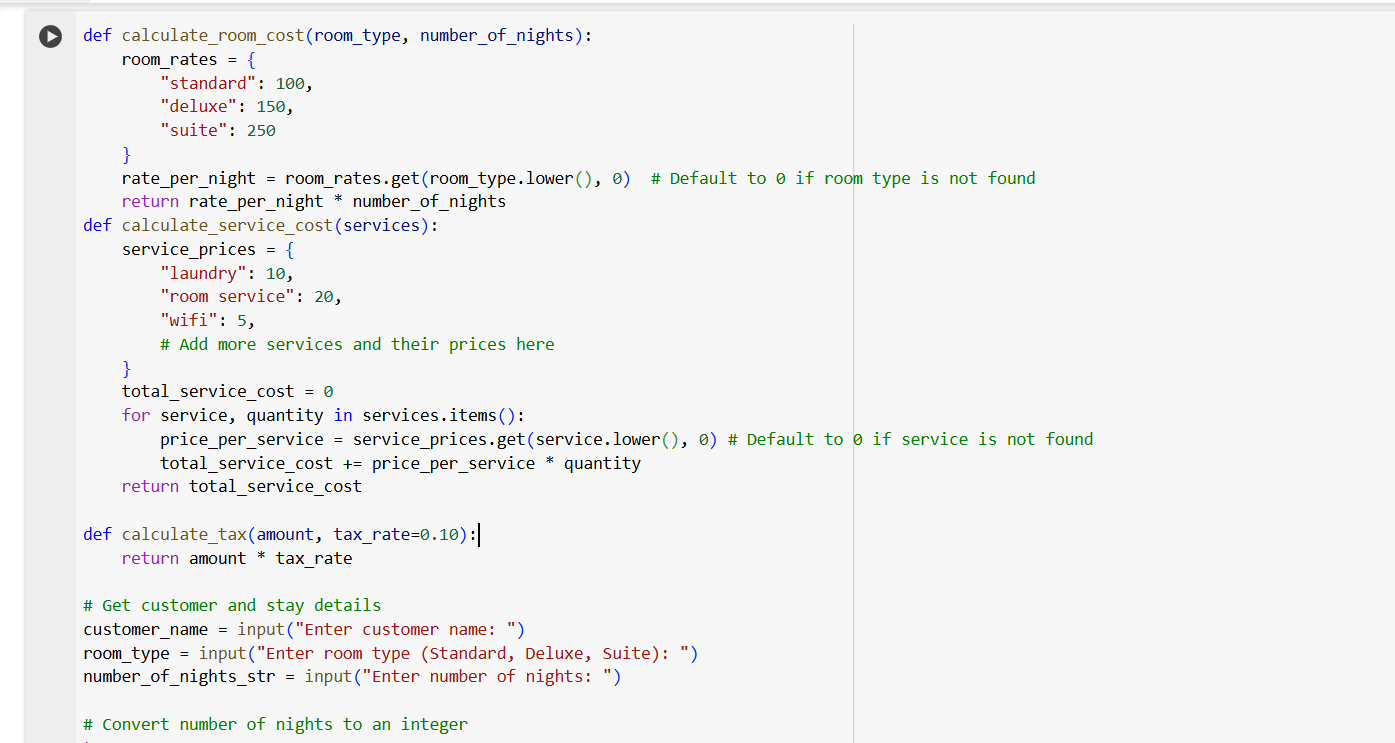
Here's a breakdown of the code:

* **def convert\_to\_binary(num):**: This line defines the function named convert\_to\_binary which takes one argument:
  + num: The decimal integer you want to convert.
* **"""Docstring"""**: This is a docstring that explains what the function does, its arguments, and what it returns.
* **if num == 0:**: This checks if the input number num is 0.
  + **return "0"**: If the number is 0, its binary representation is simply "0", so the function returns "0".
* **binary = ""**: This line initializes an empty string called binary. This string will be used to build the binary representation of the number.
* **while num > 0:**: This starts a while loop that continues as long as the value of num is greater than 0.
* **remainder = num % 2**: Inside the loop, this line calculates the remainder when num is divided by 2. This remainder will be either 0 or 1, which are the digits in a binary number.
* **binary = str(remainder) + binary**: This line takes the remainder (converted to a string) and adds it to the *beginning* of the binary string. This is how the binary digits are collected in the correct order (least significant bit first, then added to the left).
* **num = num // 2**: This line updates the value of num by performing integer division by 2. This effectively shifts the focus to the next bit in the binary conversion.
* **return binary**: Once the while loop finishes (when num becomes 0), the function returns the binary string, which now contains the binary representation of the original decimal number.
* **The print statements**: These lines demonstrate how to use the convert\_to\_binary function with different input numbers (10, 5, 0, and 16) and print the input number and its corresponding binary output.

In essence, the function repeatedly divides the decimal number by 2 and collects the remainders. These remainders, read in reverse order of their calculation, form the binary representation of the number. The code builds the binary string by adding the remainders to the beginning of the string, which effectively reverses the order as the loop progresses.

**Task-4:** Write a program in python to generate a hotel bill to input the items and quantity using GUI.

**Code:**

****

****

**A screenshot of a computer

AI-generated content may be incorrect.**

**Output:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Explanation:**

The code is divided into a few main parts:

1. **Functions for Calculations:**
   * calculate\_room\_cost(room\_type, number\_of\_nights): This function takes the room type and number of nights as input and calculates the cost of the room based on predefined rates for each room type.
   * calculate\_service\_cost(services): This function takes a dictionary of services and their quantities as input and calculates the total cost of those services based on predefined prices for each service.
   * calculate\_tax(amount, tax\_rate=0.10): This function calculates the tax amount based on a given amount and a tax rate (defaulting to 10%).
2. **Getting User Input:**
   * The code prompts the user to enter the customer's name, desired room type, and the number of nights they will be staying.
   * It also includes a loop to allow the user to enter multiple additional services and their quantities. The loop continues until the user types 'done'. Input validation is included to ensure that the number of nights and service quantities are valid integers.
3. **Calculating Total Costs:**
   * The program calls the calculate\_room\_cost function to get the room cost.
   * It calls the calculate\_service\_cost function to get the total cost of the services used.
   * It then calculates the total amount before tax by adding the room cost and service cost.
   * The calculate\_tax function is called to determine the tax amount.
   * Finally, the total amount due is calculated by adding the total before tax and the tax amount.
4. **Generating and Displaying the Bill:**
   * The code prints a formatted header for the hotel bill.
   * It displays the customer information (name, room type, and number of nights).
   * It lists the services used and their quantities.
   * It provides a summary of the calculation, showing the room cost, total service cost, and tax amount.
   * Finally, it displays the total amount due in a formatted way.

In essence, the code collects all the necessary information from the user, performs the required calculations using the defined functions, and then presents a clear and organized hotel bill.

**Detailed Explanation of Functions:**

* **calculate\_room\_cost(room\_type, number\_of\_nights):**
  + This function has a dictionary called room\_rates that stores the price per night for different room types ("standard", "deluxe", "suite").
  + It takes the room\_type as input and converts it to lowercase using .lower() to ensure that the input is matched correctly with the keys in the room\_rates dictionary, regardless of how the user capitalizes it.
  + room\_rates.get(room\_type.lower(), 0) attempts to retrieve the rate for the given room type. If the room\_type is not found in the dictionary, it defaults to 0, which prevents an error and means that an invalid room type will result in a room cost of 0.
  + Finally, it calculates the total room cost by multiplying the rate\_per\_night by the number\_of\_nights.
* **calculate\_service\_cost(services):**
  + This function has a dictionary called service\_prices that stores the price per unit for different services ("laundry", "room service", "wifi"). You can add more services and their prices to this dictionary.
  + It initializes total\_service\_cost to 0.
  + It then iterates through the services dictionary, which contains the service name as the key and the quantity as the value.
  + For each service, it retrieves the price\_per\_service from the service\_prices dictionary, again using .get() with a default of 0 for services not found.
  + It calculates the cost for the current service by multiplying the price\_per\_service by the quantity.
  + This cost is added to the total\_service\_cost.
  + After iterating through all the services, the function returns the total\_service\_cost.
* **calculate\_tax(amount, tax\_rate=0.10):**
  + This is a simple function that takes an amount and a tax\_rate as input.
  + The tax\_rate has a default value of 0.10 (10%), so if you don't provide a tax\_rate when calling the function, it will use 10%.
  + It calculates the tax by multiplying the amount by the tax\_rate and returns the result.

**Detailed Explanation of User Input Handling:**

* **Getting Customer and Stay Details:**
  + input("Enter customer name: ") prompts the user to enter their name, and the entered value is stored in the customer\_name variable.
  + input("Enter room type (Standard, Deluxe, Suite): ") prompts for the room type, storing it in room\_type.
  + input("Enter number of nights: ") prompts for the number of nights, storing the input as a string in number\_of\_nights\_str.
  + The try-except block is used for error handling when converting the number\_of\_nights\_str to an integer.
    - try: attempts to convert the string to an integer using int(number\_of\_nights\_str).
    - except ValueError: catches the error if the input string cannot be converted to an integer (e.g., if the user enters text instead of a number). In this case, it prints an error message and sets number\_of\_nights to 0 to prevent the program from crashing.
* **Getting Service Details:**
  + services\_used = {} initializes an empty dictionary to store the services and their quantities.
  + The while True: loop allows the user to continuously enter service details until they type 'done'.
  + input("Service name: ").strip() prompts for the service name and uses .strip() to remove any leading or trailing whitespace.
  + If the entered service\_name is 'done' (case-insensitive) or an empty string, the break statement exits the loop.
  + The inner while True: loop is for getting the quantity for the current service and includes input validation similar to the number of nights. It ensures the quantity is a positive integer.
  + if service\_name in services\_used: checks if the entered service is already in the services\_used dictionary. If it is, the new quantity is added to the existing quantity.
  + else: if the service is not already in the dictionary, it's added as a new entry with the entered quantity.

This more detailed explanation should give you a better understanding of how each part of the code works to generate the hotel bill.

**Task-5:** Write a program in python to convert the degrees to Fahrenheit using functions.

**Code:**

****

**Output:**

****

**Explanation:**

The code I provided is a simple Python program to convert a temperature from Celsius to Fahrenheit using a function.

Here's a breakdown:

1. **def celsius\_to\_fahrenheit(celsius):**: This line defines a function named celsius\_to\_fahrenheit. It takes one argument, celsius, which is expected to be the temperature in degrees Celsius.
2. **"""Converts Celsius to Fahrenheit."""**: This is a docstring, which explains what the function does. It's good practice to include these for your functions.
3. **fahrenheit = (celsius \* 9/5) + 32**: This is the core of the conversion. It applies the standard formula to convert Celsius to Fahrenheit: multiply Celsius by 9/5 and then add 32. The result is stored in the variable fahrenheit.
4. **return fahrenheit**: This line returns the calculated fahrenheit value from the function.
5. **celsius\_temp = 25**: This line sets a variable celsius\_temp to the value 25, which is the temperature in Celsius we want to convert.
6. **fahrenheit\_temp = celsius\_to\_fahrenheit(celsius\_temp)**: This line calls the celsius\_to\_fahrenheit function, passing the value of celsius\_temp (25) as the argument. The returned Fahrenheit value is stored in the variable fahrenheit\_temp.
7. **print(f"{celsius\_temp} degrees Celsius is equal to {fahrenheit\_temp} degrees Fahrenheit.")**: This line prints the result in a user-friendly format using an f-string. It displays the original Celsius temperature and the calculated Fahrenheit temperature.

In summary, the code defines a reusable function for the conversion and then demonstrates how to use that function with a specific example.