Appcard Inc Back-End Python Developer

Appcard Inc Back-End Developer Hiring Test Powered by HackerRank

The challenge Appcard Inc Back-End. Developer Hiring Test. Duration: 150 min. 10 tasks.

1. **Python class Car gets two arguments speed and units of speed must be implemented return string based on arguments.**
   Example of a Python class called Car that takes in two arguments, speed and units, and implements a method to return a string based on those arguments:

```
class Car:
    def __init__(self, speed, units):
        self.speed = speed
        self.units = units

    def get_speed_string(self):
        return f"The car is traveling at {self.speed} {self.units}."

# Example usage
car1 = Car(60, "mph")
print(car1.get_speed_string())  # Output: The car is traveling at 60 mph.

car2 = Car(100, "km/h")
print(car2.get_speed_string())  # Output: The car is traveling at 100 km/h.
```

   For example, the Car class has an __init__ method that initializes the speed and unit attributes of the car. The get_speed_string method returns a string using the speed and unit attributes.
   Create instances of the Car class and call the get_speed_string method to get the desired output based on the arguments provided.

2. **Build the Subsequences**
   A subsequence of a string is obtained by deleting zero or more characters from the string while maintaining order. Given a string, generate an array of all subsequences sorted alphabetically ascending.
   Example s = "xyz"
   Not including the empty string, the sorted subsequences are ['x', 'xy', 'xyz', 'xz', 'y', 'yz', 'z'].
   Function Description:
   def buildSubsequences(s)
   has the following parameter(s): str s: the string to process
   returns str[]: an array of strings comprising all subsequences of the given string sorted alphabetically, ascending.
   Constraints:
   1 < length of s < 16
   s is a string of distinct lowercase English alphabetic letters ascii[a-z].

To generate all subsequences of a given string and sort them alphabetically, you can use a recursive approach. Here's the Python code that solves this problem efficiently:

```
def buildSubsequences(s):
    if len(s) == 0:
        return []

    # Recursive case
    char = s[0]
    subsequences = buildSubsequences(s[1:])
    result = []

    for subseq in subsequences:
        result.append(subseq)
        result.append(char + subseq)

    result.append(char)

    # Sort the result in ascending order
    result.sort()

    return result

s = "xyz"
subsequences = buildSubsequences(s)
print(subsequences)

##['x', 'xy', 'xyz', 'xz', 'y', 'yz', 'z']
```

This code defines a function buildSubsequences that takes a string s as input and returns an array of strings comprising all subsequences of the given string, sorted alphabetically in ascending order.

The function starts by checking the base case where the length of the string is zero, in which case an empty list is returned.

For the recursive case, the function selects the first character of the string (char) and recursively generates the subsequences of the remaining characters in s. It then combines char with each subsequence and appends it to the result array.

Finally, the function adds the individual characters of the string to the result array and sorts it in ascending order.

3. **REST API: Capital City**

Given a country name, query the REST API at
https://jsonmock.hackerrank.com/api/countries?name=country
and return the capital city's name.
The response is a JSON object with 5 fields.
The essential field is data:
data: Either an empty array or an array with a single object that contains the country's record.

In the data array, the country has the following schema:
name: The name of the country (String)
capital: The name of the capital city (String)

A number of fields that are not of interest.
page, per_page, total, total_pages, etc., are not required for this task.
If the country is found, the data array contains exactly 1 element.
If not, it is empty, and the function should return '-1'.

If the country name is 'Italy', for example, query
https://jsonmock.hackerrank.com/api/countries?name=Italy.
A portion of the country record for Italy is:


```
{
   "name":"Italy",
        "nativeName":"Italia",
        "topLevelDomain":[".it"],
        "alpha2Code":"IT",
        "numericCode":"380",
        "alpha3Code":"ITA",
        "currencies":["EUR"],
        "callingCodes":["39"],
        "capital":"Rome"
}
```


Function Description:
Complete the **getCapitalCity** function.
getCapitalCity has the following parameters:
string country: the country to query
Returns
string: the capital city or '-1'
Constraints
The returned JSON object contains either 0 or 1 record in data.
The country name may contain uppercase and lowercase English letters and <space> (ascii 32)

To solve this problem, you can use the requests library in Python to send an HTTP GET request to the provided REST API endpoint and retrieve the capital city's name for the given country. Here's the Python code for the getCapitalCity function:

```
import requests

def getCapitalCity(country):
    url = f"https://jsonmock.hackerrank.com/api/countries?name={country}"
    response = requests.get(url)
    data = response.json()

    if len(data["data"]) == 0:
        return '-1'

    capital = data["data"][0]["capital"]
    return capital


country = "Italy"
capital_city = getCapitalCity(country)
print(capital_city)
 # Rome
```

In this code, we construct the URL by appending the country parameter to the API endpoint. We then send an HTTP GET request using the requests.get() function and retrieve the response. The response is parsed as JSON using the response.json() method, and we access the data field of the JSON object.
If the length of the data array is 0, it means no country was found, so we return '-1' as specified in the problem statement.
Otherwise, we retrieve the capital city's name from the first element of the data array and return it.

4. **Which of the following are true regarding good URI design?**

   Pick ONE OR MORE options:

   1 URIs should never be changed.
   2 URIs must be constructed by the client.
   3 URIs should be short in length.
   4 URIs should be case-sensitive.
   5 HTTP verbs should be used instead of operation names in URIs.
   6 Use spaces when designing a URI.
   7 Redirection must be used if a change in URI is required.

Good URI design is an important aspect of web development. Let's go through the options and determine which ones are true regarding good URI design:

URIs should never be changed.

1. **False.** While it is generally recommended to maintain stable URIs over time to ensure consistency and avoid breaking existing links, there may be situations where URI changes are necessary. For example, if the resource being identified by the URI undergoes a significant change, it may require a URI update. However, minimizing URI changes is generally considered a good practice.

URIs must be constructed by the client.

2. **False.** URIs are typically constructed by the server hosting the resources. Clients (such as web browsers or APIs) use the URIs provided by the server to access the resources.

URIs should be short in length.

3. **True.** Short and concise URIs are generally preferred for ease of use, readability, and shareability. Long and complex URIs can be more challenging to type, remember, and communicate.

URIs should be case-sensitive.

4. **False.** URIs are generally treated as case-insensitive. While some servers may be case-sensitive by default, it is recommended to ensure case-insensitivity in URIs to avoid confusion and potential errors.

HTTP verbs should be used instead of operation names in URIs.

5. **True.** It is considered good practice to use appropriate HTTP verbs (such as GET, POST, PUT, DELETE) to perform operations on resources instead of incorporating operation names directly into URIs. This approach adheres to the principles of RESTful design and provides a standardized and clear way of interacting with resources.

Use spaces when designing a URI.

6. **False.** Spaces are not allowed in URIs. Instead, spaces should be encoded as "%20" or replaced with other URL-safe characters, such as hyphens or underscores.

Redirection must be used if a change in URI is required.

7. **False.** While redirection can be used to guide clients from old URIs to new ones, it is not mandatory. It depends on the specific requirements and circumstances. If possible, it's generally better to avoid URI changes altogether or provide proper notifications and updates to clients to avoid breaking existing references.

Based on the explanations above, the options that are true regarding good URI design are:

- URIs should be short in length.
- HTTP verbs should be used instead of operation names in URIs.

5. Workloads on AWS

There are a number of workloads running on AWS that have embedded RDS database connection strings within each web server hosting the applications. After failing a security audit, a different approach is needed to store secrets securely and to automatically rotate the database credentials.

Which AWS service is most appropriate for this use case?

Pick ONE option:

1 SSM Parameter Store
2 Systems Manager
3 KMS
4 Secrets Manager

For securely storing secrets and automating the rotation of database credentials on AWS, the most appropriate service for this use case is **"Secrets Manager"** (option 4).

AWS Secrets Manager is a fully managed service that helps you protect access to your applications, services, and IT resources without the upfront investment and on-going maintenance costs of operating your infrastructure. It allows you to securely store and manage secrets such as database credentials, API keys, and other sensitive information. With Secrets Manager, you can centrally manage secrets, control access, and automatically rotate credentials to enhance security.

Using Secrets Manager, you can store the RDS database connection strings securely and configure automatic rotation of the database credentials. This ensures that the credentials are regularly rotated without manual intervention, reducing the risk of unauthorized access.

Option 1, SSM Parameter Store, is also a valid option for storing secrets, but it lacks the built-in automatic rotation functionality provided by Secrets Manager.

Option 2, Systems Manager, is a broader service that provides various management capabilities but is not specifically designed for secret storage and rotation.

Option 3, KMS (Key Management Service), is primarily used for encryption and key management rather than secret storage and rotation.

Therefore, the most appropriate AWS service for securely storing secrets and automating database credential rotation in this use case is "Secrets Manager" (option 4).

6. AWS EC2 - Suitable Load Balancer

This is an architectural diagram of a simple content management deployment infrastructure. The service is hosted in the Amazon cloud.
The application is a content management platform that consists of components like images, scripts, and application services. The services are deployed on different EC2 instances and managed by an auto-scaling group. Here the role of the load balancer is to route user requests based on the request path to a specified instance.

Which type of load balancer is suitable?
Pick ONE option

1 Network Load Balancer
2 Gateway Load Balancer
3 Path Load Balancer
4 Application Load Balancer

Based on the description of the architecture and the requirement to route user requests based on the request path to a specified instance, the most suitable type of load balancer would be **"Application Load Balancer" (option 4).**
The Application Load Balancer (ALB) is a layer 7 load balancer provided by AWS that operates at the application layer of the OSI model. It can intelligently distribute traffic to different targets (in this case, EC2 instances) based on various factors, including the request path, host header, or query string.
In your content management deployment infrastructure, the ALB can be configured with rules to route user requests based on the request path. For example, you can define rules to route requests for image services to one set of EC2 instances, requests for script services to another set of EC2 instances, and so on. This allows for granular control over routing traffic to specific components of the application based on the request path.
Option 1, Network Load Balancer, is a layer 4 load balancer that operates at the transport layer of the OSI model. It does not provide the request path-based routing capability required in this scenario.
Option 2, Gateway Load Balancer, is used for routing traffic between virtual and on-premises networks, and it's not suitable for this content management deployment scenario.
Option 3, Path Load Balancer, is not a load balancer service provided by AWS. It's likely a fictional or non-standard term in this context.
Therefore, the most suitable type of load balancer for routing user requests based on the request path in this content management deployment infrastructure is an "Application Load Balancer" (option 4).

7. MySQL: Join Type That Left Then Right

Which of these MySQL join types **return all records** from the left table and the matched records from the right table?

Pick ONE OR MORE options:

1 LEFT JOIN
2 LEFT INNER JOIN
3 LEFT RIGHT JOIN
4 LEFT OUTER JOIN

The MySQL join type that returns all records from the left table and the matched records from the right table is **"LEFT JOIN" (option 1).**
In a left join, all the rows from the left table are included in the result set, regardless of whether there is a match in the right table. If a match exists in the right table, the corresponding rows are also included in the result set. However, if there is no match in the right table, the result set will contain NULL values for the columns of the right table.
Option 2, "LEFT INNER JOIN," is not a valid join type. It seems to be a combination of "LEFT JOIN" and "INNER JOIN," but it is not a recognized join syntax in MySQL.
Option 3, "LEFT RIGHT JOIN," is not a valid join type either. The correct syntax is "LEFT JOIN" and "RIGHT JOIN," but they are used separately to perform left or right joins, not combined as "LEFT RIGHT JOIN."
Option 4, "LEFT OUTER JOIN," is another name for "LEFT JOIN." They refer to the same join type that returns all records from the left table and the matched records from the right table.
So, the correct answer is:

- LEFT JOIN (option 1)
- LEFT OUTER JOIN (option 4)

8. Powers of 2

Given an array of integers, determine whether each is a power of 2,
where powers of 2 are [1, 2, 4, 8, 16, 32, 64,...].
For each integer evaluated, append to an array a value of 1 if the number is a power of 2 or 0 otherwise.

Example
arr = [1, 3, 8, 12, 16]
1 = 20, 8 = 23, and 16 = 24. The return array is [1, 0, 1, 0, 1].
Function Description
Complete the function
def **isPower(arr)**
isPower has the following parameter(s):
int arr[n]: an array of integers
Returns
int[n]: array of binary integers where each index i.
contains a 1 if arr[i] is a power of 2 or a 0 if it is not
Constraints
1 <= n <= 100
0 <= arr[i] <=100000000

```
def isPower(arr):
    result = []
    for num in arr:
        if num == 0:  # 0 is not a power of 2
            result.append(0)
        elif num & (num - 1) == 0:  # Check if the number is a power of 2
            result.append(1)
        else:
            result.append(0)
    return result

# Example usage
arr = [1, 3, 8, 12, 16]
result = isPower(arr)
print(result)  # Output: [1, 0, 1, 0, 1]
```

In this code, the isPower function takes an array arr as input. It initializes an empty result list. Then, for each number num in the array, it checks if the number is equal to 0 (which is not a power of 2) or if it satisfies the condition for being a power of 2 by using the bitwise operation num & (num - 1) == 0.
If the number is a power of 2, it appends 1 to the result list. Otherwise, it appends 0. Finally, the function returns the result list.
In the example usage, the isPower function is called with the given input array [1, 3, 8, 12, 16]. The resulting list [1, 0, 1, 0, 1] is then printed as the output.

9. MySQL: Joins

Which expression is not correct?
Pick ONE option:

1 MySQL joins are used to join rows from two or more tables based on the associated column between them.
2 The INNER JOIN keyword selects records that have matching values in both tables.
3 A self-join is just a normal join, but the table joins itself.
4 Using the RIGHT JOIN keyword returns only records that are joined in the right way.

The expression that is not correct is option 4: "Using the RIGHT JOIN keyword returns only records that are joined in the right way."
This statement is incorrect. The RIGHT JOIN keyword in MySQL returns all records from the right table and the matched records from the left table. It does not mean that the records are joined in the "right" way; rather, it specifies the side from which all records are included.
In a RIGHT JOIN, the result set includes all records from the right table, and only the matching records from the left table. If there are no matches, NULL values are included for the columns from the left table.
For example, consider the following query:

```
SELECT *
FROM table1
RIGHT JOIN table2
ON table1.id = table2.id;
```

This query will return all records from table2, and only the matching records from table1 based on the join condition. If there are no matching records in table1, the result will include NULL values for the columns from table1.
Therefore, the **correct answer is option 4**.

10. MySQL: PRIMARY KEY Constraint

Which expression is not correct?
Pick ONE option:
1 The primary key uniquely identifies each record in the table.
2 The primary key must contain unique values.
3 The primary key can contain NULL values, but this is highly undesirable for performance reasons.
4 A table can have only one primary key.

The expression that is not correct is option 3: "The primary key can contain NULL values, but this is highly undesirable for performance reasons."
This statement is incorrect. In MySQL, the primary key cannot contain NULL values. The primary key is a column or a combination of columns that uniquely identifies each record in the table. By definition, a primary key must have unique values, and NULL is not considered a valid value for the primary key.
When defining a primary key on a column or a set of columns in MySQL, you can enforce the uniqueness constraint by using the PRIMARY KEY constraint. This ensures that each value in the primary key column(s) is unique and not NULL.
Therefore, **the correct answer is option 3**.

11. MySQL: GROUP BY Statement

Select the expression that is not correct.
Pick ONE option:

1 The GROUP BY statement groups rows with the same value into total rows.
2 Aggregate functions (COUNT(), MAX(), MIN(), etc.) can be used only with an explicit GROUP BY statement.
3 You can apply an additional condition after the grouping using the HAVING clause.
4 None of the above, all expressions are correct.

The expression that is not correct is option 2: "Aggregate functions (COUNT(), MAX(), MIN(), etc.) can be used only with an explicit GROUP BY statement."
This statement is incorrect. Aggregate functions can be used without an explicit GROUP BY statement. When an aggregate function is used without GROUP BY, it performs the aggregation on the entire result set, resulting in a single row of aggregated values.
For example, you can use an aggregate function like COUNT() to count the total number of rows in a table without using GROUP BY:
SELECT COUNT(*) FROM your_table;
This query will return a single row with the total count of rows in the table.
Therefore, the correct **answer is option 2.**

Appcard Inc Back-End Python Developer

appointment 5

A code year containing a stack with several different API interfaces. For example, there were three API interfaces in the existing code, each receiving output. In the new code, there is only one API and it receives a single output. The new code is intended to replace the functionality of the existing code, but the final result should remain the same.
Question 1: Describe your approach to problem-solving. What steps should be taken to solve this problem?
Question 2: How to verify that the new code is functioning correctly?

Answer to question 1:
1. Understand the functionality of each API: Examine the old code that uses the API and analyze its purpose and behavior. Understand the inputs required for each of them, the actions they perform, and the results they produce.
2. Identify common functions: Look for similarities and patterns in the old code. Identify the key functions that are essential for achieving the desired end result. This will help you determine what needs to be implemented in the new code.
3. Design the new code: Based on the common functions identified in step 2, design the new API. Define the input parameters it will receive and the output it will produce. Consider the necessary operations and algorithms to replicate the behavior of the original API stack.
4. Implement the new code: Write the code for the new API and ensure that it performs the required actions and produces the same results as the original API stack. Validate the correctness and accuracy of the new code.
5. Test the new code: Replace the old API stack with the new API in the code file. Run the code with different test scenarios and compare the output results of the new implementation with the previous implementation.

Answer to question 2:

Here is the proposed approach:

1. Test Creation: Before making changes to the code, I suggest writing tests that check the current output of your API set. This way, you can ensure that after the changes, the output remains the same.
2. Output Comparison: Run the old API set and save its output. Then, run the new API set and compare its output to the previous result. If they are identical, it means the final output remained the same.
3. Log Usage: I recommend enabling logs in the code before and after the changes in the API set. Record the execution results of both versions in log files. Afterward, you can compare the log files to verify that the final result hasn't changed.
4. Comparison of Mean Results: If your API set returns mean results before the final output, it's important to ensure that they are preserved after the changes. You can compare the mean results of the old and new API sets to check their consistency.
5. Manual Verification: Check the previous API's output manually and compare it to the output of the new API. Look for changes or discrepancies in the results to identify potential issues.