# Technical Interview – Homework - Sibina

### Junior Python Developer – Interview Questions

|  |  |
| --- | --- |
| 1. What is a **set**? How does it work? | A set in Python is an unordered collection of unique elements. It is defined by enclosing the elements within curly braces {}.   1. **Uniqueness:** A set does not allow duplicate elements. 2. **Unordered:** Unlike lists or tuples, the elements in a set do not have a specific order. |
| 2. What is a **tuple**? How does it work? | A tuple is an ordered, immutable collection of elements. This means that once a tuple is created, its elements cannot be changed or modified. Tuples are defined by enclosing the elements within parentheses ().  Key characteristics of tuples include:   1. **Ordered:** Unlike sets, the elements in a tuple have a specific order, and this order is maintained. 2. **Immutable:** Once a tuple is created, you cannot add, remove, or modify elements. This immutability makes tuples suitable for situations where you want to ensure that the data remains constant.   To create a tuple, it can use parentheses and separate the elements with commas:  my\_tuple = (1, 2, 3, 4, 5) |
| 3. Write a **function** that takes a list of integers and returns a new list where each element is the sum of the corresponding elements of two input lists. | A simple Python function that takes two lists of integers and returns a new list where each element is the sum of the corresponding elements from the input lists:  def sum\_lists(list1, list2):  # Use a list comprehension to add corresponding elements  result = [x + y for x, y in zip(list1, list2)]  return result  # Example usage:  list\_a = [1, 2, 3]  list\_b = [4, 5, 6]  result\_list = sum\_lists(list\_a, list\_b)  print(result\_list)  This function uses the zip function to iterate through corresponding elements of both lists, adds them together, and creates a new list with the results. |
| 4. Can you tell me what is **binary search** and how does it work? | Binary search is an efficient algorithm for finding a specific value in a sorted sequence, such as an array or a list.  Here's a step-by-step explanation of how binary search works:   1. **Initialize the search interval:** Start with the entire sorted sequence as the search interval. 2. **Find the middle element:** Calculate the middle index of the current interval. If the middle element is equal to the target value, you've found it. If not, proceed to the next step. 3. **Adjust the search interval:** If the target value is less than the middle element, narrow the search interval to the left half (excluding the middle element). If the target value is greater, narrow the search interval to the right half. 4. **Repeat:** Repeat steps 2 and 3 until the target value is found or the search interval becomes empty. |
| 5. What is “**list**”? How does it work? | A list is a versatile and mutable data structure that allows you to store a collection of items. Lists are ordered, meaning the order of elements is maintained, and they are defined by enclosing the elements within square brackets [].  **- Creating a list:**  my\_list = [1, 2, 3, 'a', 'b', 'c']  - **Accessing elements:** You can access individual elements in a list using indexing. Python uses zero-based indexing, so the first element is at index 0.  print(my\_list[0]) # Output: 1  - **List methods:** Lists in Python come with a variety of built-in methods. For example:   * append(): Adds an element to the end of the list. * remove(): Removes the first occurrence of a specified value. * pop(): Removes and returns the element at a specified index. * len(): Returns the number of elements in the list.   **- Slicing:** You can extract a portion of a list using slicing. This is done by specifying a range of indices.  **- Mutability:** Lists are mutable, which means you can modify their elements by assignment or using various list methods.  my\_list[0] = 10 # Modify the first element |
| 6. Write a function that takes a string and returns True if the string is **palindrome**, and false otherwise. | A simple Python function that checks if a given string is a palindrome:  def is\_palindrome(input\_str):  # Remove spaces and convert to lowercase for  case-insensitive comparison  clean\_str = ''.join(input\_str.split()).lower()  # Check if the cleaned string is equal to its reverse  return clean\_str == clean\_str[::-1]    # Example usage:  test\_string = "A man a plan a canal Panama"  result = is\_palindrome(test\_string)  print(result)  This function first removes spaces and converts the string to lowercase to ensure a case-insensitive comparison. Then, it checks if the cleaned string is equal to its reverse, which determines if it's a palindrome. |
| 7. Can you write a **function** that takes a string and returns the number of vowels in the string? | A simple Python function that counts the number of vowels in a given string:  python  def count\_vowels(input\_str):  vowels = "aeiouAEIOU"  # Using a list comprehension to count vowels  vowel\_count = sum(1 for char in input\_str if char in vowels)  return vowel\_count  # Example usage:  test\_string = "Hello, World!"  result = count\_vowels(test\_string)  print(result)  This function defines a set of vowels and then uses a list comprehension to iterate through each character in the input string, checking if it's a vowel. The sum function is then used to count the number of True values, giving the total count of vowels. |
| 8. Can you tell me what is a **polymorphism**? Give me an example. | Polymorphism is a concept in object-oriented programming (OOP) that allows objects of different types to be treated as objects of a common type. In other words, it allows a single interface to represent different types of objects. Polymorphism can manifest in different forms, including method overloading and method overriding. |
| 9. What is **encapsulation**? Give me an example. | Encapsulation is one of the fundamental principles of object-oriented programming (OOP) that involves bundling the data (attributes) and the methods (functions) that operate on the data into a single unit known as a class. It restricts direct access to some of an object's components and can prevent unintended interference and misuse. |
| 10. You are building a system to manage a company's employee hierarchy. Each employee has a name, job title, and salary. The company has different departments such as IT, Marketing, and Sales. Draw **an object-oriented hierarchy** to represent the employees and departments. | A simplified object-oriented hierarchy in Python to represent employees and departments:  class Employee:  def \_\_init\_\_(self, name, job\_title, salary):  self.name = name  self.job\_title = job\_title  self.salary = salary  def display\_info(self):  print(f"Name: {self.name}, Job Title: {self.job\_title}, Salary: {self.salary}")  class Department:  def \_\_init\_\_(self, name):  self.name = name  self.employees = []  def add\_employee(self, employee):  self.employees.append(employee)  def display\_department\_info(self):  print(f"Department: {self.name}")  print("Employees:")  for employee in self.employees:  employee.display\_info()  print("\n")  # Example Usage:  it\_department = Department("IT")  marketing\_department = Department("Marketing")  sales\_department = Department("Sales")  employee1 = Employee("John Doe", "Software Engineer", 80000)  employee2 = Employee("Jane Smith", "Marketing Specialist", 70000)  employee3 = Employee("Bob Johnson", "Sales Representative", 75000)  it\_department.add\_employee(employee1)  marketing\_department.add\_employee(employee2)  sales\_department.add\_employee(employee3)  it\_department.display\_department\_info()  marketing\_department.display\_department\_info()  sales\_department.display\_department\_info()  In this example:   * The Employee class represents an individual employee with attributes like name, job title, and salary. * The Department class represents a department with a name and a list of employees. It has methods to add employees and display department information. * The example usage at the end demonstrates creating instances of employees and departments, adding employees to departments, and displaying department information. |
| 11. Can you write a Postgres query to find the **total** number of orders for **each** customer in the Orders table? | Using PostgreSQL, here's an example query to find the total number of orders for each customer in the "Orders" table:  SELECT CustomerID, COUNT(OrderID) AS TotalOrders  FROM Orders  GROUP BY CustomerID;  PostgreSQL uses the same SQL syntax for standard operations, so I can use the above query directly in PostgreSQL.  Make sure to replace 'Orders', 'CustomerID', and 'OrderID' with the actual names of your table and columns if they differ from these examples. |
| 12. Can you write a Postgres query to find the **top** **5** highest paid employees in the Employees table? | Assuming you have a PostgreSQL database, you can use the following query to find the top 5 highest paid employees in the "Employees" table:  sql  SELECT EmployeeID, FirstName, LastName, Salary  FROM Employees  ORDER BY Salary DESC  LIMIT 5;  In this query:   * SELECT EmployeeID, FirstName, LastName, Salary: Specifies the columns you want to retrieve. * FROM Employees: Specifies the table from which to retrieve the data. * ORDER BY Salary DESC: Orders the results based on the Salary column in descending order (highest salary first). * LIMIT 5: Limits the result set to the top 5 rows.   It will give you the EmployeeID, FirstName, LastName, and Salary of the top 5 highest paid employees in the "Employees" table. Adjust the column names and table name based on your actual database schema. |
| 13.How does Postgres **inner** join work? Can you provide example? | PostgreSQL, like other relational databases, uses the INNER JOIN operation to combine rows from two or more tables based on a related column. Here's an example using PostgreSQL:  Let's consider two tables: employees and departments.  **employees Table:**  sql  CREATE TABLE employees (  employee\_id SERIAL PRIMARY KEY,  first\_name VARCHAR(50),  last\_name VARCHAR(50),  department\_id INT  );  INSERT INTO employees (first\_name, last\_name, department\_id)  VALUES  ('John', 'Doe', 1),  ('Jane', 'Smith', 2),  ('Bob', 'Johnson', 1),  ('Alice', 'Williams', 3);  **departments Table:**  sql  CREATE TABLE departments (  department\_id SERIAL PRIMARY KEY,  department\_name VARCHAR(50)  );  INSERT INTO departments (department\_name)  VALUES  ('IT'),  ('Marketing'),  ('Sales');  Now, let's perform an INNER JOIN to retrieve employees along with their department names:  sql  SELECT employees.employee\_id, employees.first\_name, employees.last\_name, departments.department\_name  FROM employees  INNER JOIN departments ON employees.department\_id = departments.department\_id;  In this query:   * INNER JOIN departments: Specifies that you want to combine rows from the employees and departments tables. * ON employees.department\_id = departments.department\_id: Specifies the condition for the join, indicating that the department\_id column in the employees table should match the department\_id column in the departments table.   The result will be:  markdown  employee\_id | first\_name | last\_name | department\_name  -------------+------------+-----------+------------------  1 | John | Doe | IT  2 | Jane | Smith | Marketing  3 | Bob | Johnson | IT  4 | Alice | Williams | Sales  This result set combines information from both tables based on the matching department\_id. Only rows where there is a match in both tables are included in the result. |
| 14. Your web app has a form for users to submit their contact information. How would you use an MVC framework to handle form submissions and **save** it to a database? | Handling a form submission in a web app involves various steps, especially when using an MVC (Model-View-Controller) framework. It is using a Python web framework. Model (Data Handling): Define a model or use an ORM (Object-Relational Mapping) to interact with the database.View (HTML Form): Create an HTML form in your template to allow users to input their contact information.Controller (Form Submission):Create a controller to handle form submissions, validate data, and interact with the model to save data to the database. In this example:   * The models.py file defines the data model (Contact) and sets up the database. * The templates/index.html file contains the HTML form for users to input their contact information. * The app.py file is the main application file, defining routes, handling form submissions, and interacting with the database. * The templates/thank\_you.html file is a simple thank-you page. |
| 15. How would you create a RESTful API using Django to allow users to **retrieve** and **manipulate** data from a database? | Creating a RESTful API using Django involves defining models, serializers, views, and URLs 1. Install Django and Django REST framework2. Create a Django Project and App3. Define Models4. Create Serializers5. Create Views6. Configure URLs7. Include App URLs in Project URLs8. Migrate Database9. Run the Development Server10. Test Your API |
| 16. You are building a web app that allows users to create and **edit** their profiles. How would you use an MVC framework to handle user requests to **view**, **update**, or **delete** their profile information? | Using an MVC framework involves defining models for user profiles, creating views for user interfaces, and implementing controllers to handle user requests 1. Model: Define a model to represent user profiles. This model should interact with the database to store and retrieve user profile information.2. View: Create views to display user profile information and capture user input for updates or deletions.3. Controller: Create a controller to handle user requests, interact with the model, and update the view.4. Putting It All Together: Instantiate the model, view, and controller, and use them to handle user requests. |
| 17. You are building an e-commerce website that sells different types of products such as clothing, electronics, and home appliances. Draw an **object-oriented hierarchy** to represent the products. | The hierarchy represent different types of products in an e-commerce website.   * Product is the base class representing common attributes for all products. * Clothing, Electronics, and HomeAppliance are subclasses that inherit from Product and add specific attributes for each type of product. * Each class has a display\_info method to print information about the product.   It can extend or modify it based on your specific requirements and types of products |
| 18. You need to implement client-side form validation in JavaScript. How would you ensure that form data is **valid** before submitting it to the server? | Implementing client-side form validation in JavaScript is crucial for enhancing user experience and reducing unnecessary server requests. 1. HTML Form: Create an HTML form with appropriate input fields and a submit button. Assign unique IDs to form elements f or easy identification in JavaScript.2. JavaScript Validation Function3. Prevent Default Form Submission4. Submit Button Activation5. Displaying Error Messages6. HTML5 Validation Attributes By combining these steps, it can implement client-side form validation to ensure that form data is valid before it is submitted to the server, improving user experience and reducing unnecessary server requests. |
| 19. How would you implement a simple web form using Flask to **accept user input** and **store** **it** in a database? | To implement a simple web form using Flask to accept user input and store it in a database, follow these steps: 1. Install Flask and Set Up Project2. Create a Flask App3. Create HTML Templates4. Run the Flask App This is a basic example, and you can expand it by adding more fields to the form, improving validation, and enhancing the user interface. |
| 20. How would you implement user **authentication** using Django to **restrict** access to certain pages or functionality on your web application? | To implement user authentication using Django and restrict access to certain pages or functionality, follow these steps: 1. Install Django2. Create a Django Project and App3. Set Up Database4. Define User Model5. Migrate Database6. Create User Registration and Login Views7. Create Templates8. Update URLs9. Restrict Access to Pages10. Implement Logout Link11. Run the Development Server It can customize it further based on your application's requirements, such as adding additional user information, handling password reset, and using more advanced authentication features provided by Django. |