Viva questions and answer

1. **What is Python?**
    - Python is a high-level, interpreted, and dynamically typed programming language known for its readability and versatility.
2. **What is a dynamically typed language?**
    - In a dynamically typed language like Python, variable types are determined at runtime rather than during compilation.
3. **What is an Interpreted language?**
    - Python is an interpreted language, meaning that the code is executed line by line without prior compilation.
4. **What is PEP 8 and why is it important?**
    - PEP 8 (Python Enhancement Proposal 8) is the official style guide for Python code.
    - It promotes consistent and readable code by providing guidelines on naming conventions, indentation, and other aspects.
5. **What is Scope in Python?**
    - Scope refers to the region of code where a variable is accessible.
    - Python has local, enclosing, global, and built-in scopes.
6. **What are lists and tuples? What is the key difference between the two?**
    - Both lists and tuples are ordered collections of elements.
    - The key difference is that lists are mutable (can be modified), while tuples are immutable (cannot be changed after creation).
7. **What are the common built-in data types in Python?**
    - Common data types include integers, floats, strings, lists, tuples, dictionaries, and sets.
8. **What is pass in Python?**
    - `pass` is a placeholder statement that does nothing. It is often used as a stub or a placeholder for future code.
9. **What are modules and packages in Python?**
    - Modules are files containing Python code that can be imported into other programs.
    - Packages are directories containing multiple modules.
10. **What is the use of `self` in Python?**
    - `self` refers to the instance of a class and is used to access its attributes and methods within the class.
11. **What is `__init__`?**
    - `__init__` is a special method (constructor) in Python classes. It is automatically called when an object is created from a class.
12. **What is slicing in Python?**
    - Slicing allows you to extract a portion of a sequence (such as a list or string) by specifying start, end, and step values.
13. **How can you make a Python Script executable on Unix?**
    - You can add a shebang line (`#!/usr/bin/env python`) at the beginning of your script and make it executable using `chmod +x filename.py`.
14. **What is the difference between Python Arrays and lists?**
    - Python arrays are a part of the `array` module and are more efficient for numerical computations.
    - Lists are more versatile and can hold elements of different data types.
15. **How is memory managed in Python?**

   - Python uses automatic memory management (garbage collection) to reclaim memory occupied by objects no longer in use.

16. **What are decorators in Python?**
   - Decorators are functions that modify the behavior of other functions or methods.
   - They are often used for logging, authentication, and memoization.
17. **What is lambda in Python? Why is it used?**
   - A lambda function is an anonymous function defined using the `lambda` keyword.
   - It is used for short, simple operations and is often passed as an argument to higher-order functions.
18. **How do you copy an object in Python?**
   - You can use the `copy` module to create shallow or deep copies of objects.
19. **What are generators in Python?**
   - Generators are special iterators that yield values one at a time, saving memory and improving performance.
20. **What is the difference between `xrange` and `range` in Python?**
   - In Python 2, `xrange` generates values lazily, while `range` creates a list of values.
   - In Python 3, `range` behaves like `xrange`.
21. **What is the Global Interpreter Lock (GIL) in Python?**
   - The GIL is a mutex that allows only one thread to execute Python code at a time.
   - It prevents multiple threads from executing in parallel, affecting multi-core processors.
22. **Explain the difference between shallow copy and deep copy in Python.**
   - A shallow copy creates a new object but references the original nested objects.
   - A deep copy creates a completely independent copy of the original object and its nested objects.
23. **What are decorators in Python? Provide an example.**
   - Decorators modify the behavior of functions or methods.
   - Example:

Python

```python
def log_decorator(func):
    def wrapper(*args, **kwargs):
        print(f"Calling {func.__name__}")
        result = func(*args, **kwargs)
        print(f"{func.__name__} returned {result}")
        return result
    return wrapper

@log_decorator
def add(a, b):
    return a + b

add(3, 5)
```

AI-generated code. Review and use carefully. More info on FAQ.

24. **What is the purpose of the `__name__` attribute in Python?**
   - The `__name__` attribute holds the name of the current module or script.

- When a script is run directly, `__name__` is set to `"__main__"`.

25. **How do you handle exceptions in Python?**
    - Use `try`, `except`, `else`, and `finally` blocks to handle exceptions.
    - Example:

Python

```
try:
    result = 10 / 0
except ZeroDivisionError:
    print("Cannot divide by zero")
else:
    print("No exception occurred")
finally:
    print("Cleanup code")
```

AI-generated code. Review and use carefully. More info on FAQ.

26. **What is the purpose of the `if __name__ == "__main__":` block?**
    - It ensures that code within the block runs only when the script is executed directly (not when imported as a module).
27. **How do you reverse a string in Python?**
    - You can use slicing:

Python

```
reversed_string = original_string[::-1]
```

AI-generated code. Review and use carefully. More info on FAQ.

28. **What is the difference between `append()` and `extend()` for lists?**
    - `append()` adds a single element to the end of a list.
    - `extend()` adds all elements from an iterable (e.g., another list) to the end of a list.
29. **What is a closure in Python?**
    - A closure is a function that remembers the environment in which it was created.
    - It retains access to variables from its enclosing scope even after that scope has finished executing.
30. **How do you remove duplicates from a list in Python?**
    - Use `set()` to convert the list to a set (which automatically removes duplicates).
    - Convert it back to a list if order matters.
31. **What are virtualenvs?**
    - A virtualenv is an isolated environment for developing, running, and debugging Python code.
    - It allows you to isolate a Python interpreter along with a specific set of libraries and settings.
    - With virtualenvs, you can develop, deploy, and run multiple applications on a single host, each with its own version of the Python interpreter and a separate set of libraries.

32. **What are Wheels and Eggs? What is the difference?**
    - Wheels and Eggs are both packaging formats that aim to provide install artifacts without requiring building or compilation.
    - Key differences:
        1. **Wheel**:
            1. Introduced by PEP 427 in 2012.
            2. A distribution format (packaging format).
            3. Does not include `.pyc` files.
            4. Uses PEP376-compliant `.dist-info` directories.
        2. **Egg**:
            1. Introduced by setuptools in 2004.
            2. Both a distribution format and a runtime installation format (if left zipped).
            3. Designed to be importable.
            4. Used `.egg-info` directories.
33. **What are global, protected, and private attributes in Python?**
    - **Global attributes**: Defined at the module level and accessible from any part of the code.
    - **Protected attributes**: Conventionally marked with a single leading underscore (e.g., `_protected`).
    - **Private attributes**: Conventionally marked with a double leading underscore (e.g., `__private`).
    - Note that Python does not enforce true privacy; these conventions are for readability and convention.
34. **What is the purpose of `self` in Python?**
    - `self` refers to the instance of a class and is used to access its attributes and methods within the class.
    - It is the first parameter in instance methods.
35. **What is the difference between `==` and `is` in Python?**
    - `==` compares the values of two objects.
    - `is` compares the identities (memory addresses) of two objects.
    - In other words, `==` checks for equality, whereas `is` checks for identity.
36. **What are virtualenvs?**
    - A virtualenv is an isolated environment for developing, running, and debugging Python code.
    - It allows you to isolate a Python interpreter along with a specific set of libraries and settings.
    - With virtualenvs, you can develop, deploy, and run multiple applications on a single host, each with its own version of the Python interpreter and a separate set of libraries.

37. **What are Wheels and Eggs? What is the difference?**
    - Wheels and Eggs are both packaging formats that aim to provide install artifacts without requiring building or compilation.
    - Key differences:
        1. **Wheel**:
            1. Introduced by PEP 427 in 2012.
            2. A distribution format (packaging format).
            3. Does not include `.pyc` files.
            4. Uses PEP376-compliant `.dist-info` directories.

2. **Egg**:
    1. Introduced by setuptools in 2004.
    2. Both a distribution format and a runtime installation format (if left zipped).
    3. Designed to be importable.
    4. Used `.egg-info` directories.

38. **What are global, protected, and private attributes in Python?**
    - **Global attributes**: Defined at the module level and accessible from any part of the code.
    - **Protected attributes**: Conventionally marked with a single leading underscore (e.g., `_protected`).
    - **Private attributes**: Conventionally marked with a double leading underscore (e.g., `__private`).
    - Note that Python does not enforce true privacy; these conventions are for readability and convention.
39. **What is the purpose of `self` in Python?**
    - `self` refers to the instance of a class and is used to access its attributes and methods within the class.
    - It is the first parameter in instance methods.
40. **What is the difference between `==` and `is` in Python?**
    - `==` compares the values of two objects.
    - `is` compares the identities (memory addresses) of two objects.
    - In other words, `==` checks for equality, whereas `is` checks for identity.
41. **What is DBMS?**
    - A **Database Management System (DBMS)** is a program that controls the creation, maintenance, and use of a database.
    - It acts as a file manager for managing data in a structured way, rather than saving it in file systems.
42. **What is RDBMS?**
    - **RDBMS** stands for **Relational Database Management System**.
    - RDBMS stores data in tables (collections of rows and columns) related by common fields.
    - It provides relational operators to manipulate the data stored in these tables.
43. **What is SQL?**
    - **SQL (Structured Query Language)** is used to communicate with databases.
    - It allows you to perform tasks such as retrieval, updating, insertion, and deletion of data from a database.
44. **What is a Database?**
    - A **database** is an organized form of data for easy access, storage, retrieval, and management.
    - It can be accessed in various ways and is often used for applications like school management or bank management.
45. **What are tables and fields?**
    - A **table** is a set of data organized in rows and columns.
    - Columns represent fields, and rows represent records.
    - For example:
        1. Table: Employee
        2. Fields: Emp ID, Emp Name, Date of Birth
        3. Data: 201456, David, 11/15/1960

46. **What is a primary key?**
    - A **primary key** uniquely specifies a row in a table.
    - It has an implicit **NOT NULL** constraint, meaning primary key values cannot be NULL.
47. **What is a unique key?**
    - A **unique key** uniquely identifies each record in the database.
    - Unlike a primary key, it does not have an automatic unique constraint defined on it.
48. **What is a foreign key?**
    - A **foreign key** relates one table to the primary key of another table.
    - It establishes a relationship between tables by referencing the primary key of another table.
49. **What is a join?**
    - A **join** is used to query data from multiple tables based on their relationship.
    - Keys play a major role when performing joins.
50. **What are the types of joins?**
    - Common types of joins include:
        1. **INNER JOIN**: Retrieves matching records from both tables.
        2. **LEFT JOIN (or LEFT OUTER JOIN)**: Retrieves all records from the left table and matching records from the right table.
        3. **RIGHT JOIN (or RIGHT OUTER JOIN)**: Retrieves all records from the right table and matching records from the left table.
        4. **FULL JOIN (or FULL OUTER JOIN)**: Retrieves all records from both tables.
51. **What is OOPS?**
    - **OOPS** stands for **Object-Oriented Programming System**.
    - It is a programming paradigm based on the concept of "objects."
    - OOPS aims to make programming more modular, reusable, and maintainable.
    - Objects represent real-world entities or concepts and have properties (attributes) and behaviors (methods) associated with them.
52. **What are the key principles of OOPS?**
    - **Encapsulation**: Bundling data (attributes) and methods (functions) that operate on the data into a single unit (class).
    - **Inheritance**: Creating a new class by inheriting properties and behaviors from an existing class.
    - **Polymorphism**: Providing a single interface to different data types or classes.
53. **What are classes and objects in Python?**
    - A **class** is a blueprint for creating objects.
    - An **object** is an instance of a class.
54. **What is the difference between a class and an object?**
    - A **class** defines the structure and behavior of objects.
    - An **object** is an instance of a class, representing a specific entity.
55. **What is the purpose of constructors in Python?**
    - Constructors (usually named `__init__`) initialize the attributes of an object when it is created.
    - They allow you to set initial values for object properties.
56. **What is method overloading in Python?**
    - Method overloading allows a class to have multiple methods with the same name but different parameters.

- Python does not support traditional method overloading, but you can achieve it using default arguments or variable-length arguments.

57. **What is method overriding in Python?**
    - Method overriding occurs when a subclass provides a specific implementation for a method that is already defined in its superclass.
    - The overridden method in the subclass has the same name, parameters, and return type as the method in the superclass.
58. **What is encapsulation?**
    - Encapsulation refers to bundling data (attributes) and methods (functions) that operate on the data into a single unit (class).
    - It helps in hiding the internal details of an object and exposing only necessary information.
59. **What is inheritance?**
    - Inheritance allows a new class (subclass or derived class) to inherit properties and behaviors from an existing class (superclass or base class).
    - It promotes code reusability and establishes a parent-child relationship between classes.
60. **What is polymorphism?**
    - Polymorphism allows objects of different classes to be treated as objects of a common superclass.
    - It enables dynamic method dispatch, where the appropriate method is called based on the actual object type at runtime.
61. **How is Tkinter used to create GUI applications in Python?**
    - Tkinter is Python’s standard GUI library for creating desktop applications.
    - To create a GUI application using Tkinter:
        1. Import the module.
        2. Create an instance of the `Tk` class (main window).
        3. Add widgets (like buttons, labels, and entry fields) to the window.
        4. Customize widgets and handle events using methods like `pack()`, `grid()`, and `bind()`.
62. **How would you handle events in Tkinter?**
    - Events are handled using the `bind()` method.
    - Syntax: `widget.bind(event, handler).`
    - Example:

Python

```
from tkinter import Tk, Button

def print_message(event):
    print("Button clicked")

root = Tk()
button = Button(root, text="Click me")
button.bind("<Button-1>", print_message)
button.pack()
root.mainloop()
```

63. 0**What is the purpose of the `mainloop()` method in Tkinter?**

  - The `mainloop()` method starts the event loop, allowing the application to respond to user interactions.
  - It keeps the window open until the user closes it.

64. **How do you create a new window (Toplevel) in Tkinter?**
  - Use the `Toplevel()` constructor to create a new top-level window.
  - Example:

Python

```
from tkinter import Tk, Toplevel

root = Tk()
new_window = Toplevel(root)
new_window.title("New Window")
```

65. **What are frames in Tkinter?**
  - Frames are containers used to group and organize widgets.
  - They provide a way to manage layout and improve organization within a window.

66. **How do you create a menu bar in Tkinter?**
  - Create a `Menu` widget and add items (commands or submenus) to it.
  - Attach the menu to the main window using `menu()` method.
  - Example:

Python

```
from tkinter import Tk, Menu

root = Tk()
menu_bar = Menu(root)
root.config(menu=menu_bar)
```

67. **How can you create a canvas in Tkinter?**
  - Use the `Canvas` widget to create a drawing area.
  - Example:

Python

```
from tkinter import Tk, Canvas

root = Tk()
canvas = Canvas(root, width=200, height=100)
canvas.pack()
```