What is __init__?Explain with an example ?

The __init__ method is a special method in Python classes that is called when a new instance of the class is created. It initializes the instance with default or provided values and sets up the initial state of the object.

```python
class Person:
    def __init__(self, name, age):
        # This method is called when a new instance of Person is created
        self.name = name
        self.age = age

    def greet(self):
        print(f"Hello, my name is {self.name} and I am {self.age} years old.")

# Creating an instance of the Person class
person1 = Person("Alice", 30)

# Calling the greet method
person1.greet()
```

```
Hello, my name is Alice and I am 30 years old.
```

What is docstring in Python?Explain with an example ?

A docstring is a string literal that appears right after the definition of a module, class, method, or function. It is used to describe what the code does, its parameters, return values, and any other relevant information. Docstrings are enclosed in triple quotes (""" or '''), which allows them to span multiple lines.

```python
class Calculator:

    def __init__(self):

        pass

    def add(self, a, b):

        return a + b

    def subtract(self, a, b):

        return a - b

# Creating an instance of Calculator
calc = Calculator()

# Accessing the docstring of the Calculator class
print(Calculator.__doc__)

# Accessing the docstring of the add method
print(calc.add.__doc__)
```

```
None
None
```

What are unit tests in Python

Unit tests are automated tests written to check the functionality of small, isolated units of code, typically functions or methods. These tests verify that each unit of the code performs as expected in various scenarios, including edge cases.

What is break, continue and pass in Python ?

1. break

Definition: The break statement is used to exit a loop prematurely, regardless of the loop's condition. When encountered, it terminates the nearest enclosing loop (for or while) and proceeds with the code following the loop

2. continue

Definition: The continue statement is used to skip the current iteration of a loop and proceed with the next iteration. It does not exit the loop but rather causes the loop to skip the remaining code for the current iteration and continue with the next iteration.

3. pass

Definition:

The pass statement is a null operation or placeholder. It is used when a statement is syntactically required but you have nothing to execute. It allows you to write code that does nothing and is often used as a placeholder for future code.

```
for i in range(10):
    if i == 5:
        break
    print(i)

for i in range(10):
    if i % 2 == 0:
        continue
    print(i)
    def placeholder_function():
    pass

for i in range(10):
    if i < 5:
        pass  # Placeholder for future code
    else:
        print(i)
```

What is the use of self in Python

self is a conventional name for the first parameter of methods in a class. It refers to the instance of the class itself, allowing access to the instance's attributes and methods. While self is not a keyword in Python, it is a strong convention that makes the code easier to understand and maintain.

What are global, protected and private attributes in Python

1. Global Attributes

Definition: Global attributes are variables that are defined outside of any class or function, making them accessible from anywhere in the code. They are not tied to any particular class or instance.

2. Protected Attributes

Definition: Protected attributes are intended to be accessible only within the class that defines them and its subclasses. In Python, they are indicated by a single leading underscore (_). This is a convention rather than enforced access control.

3. Private Attributes

Definition: Private attributes are intended to be accessible only within the class that defines them. They are indicated by a double leading underscore (__). Python performs name mangling for private attributes to make it harder (but not impossible) to access them from outside the class.

What are modules and packages in Python

Modules

Definition: A module is a single file that contains Python code. It can define functions, classes, and variables, and it can include runnable code. Modules help break down large programs into smaller, manageable

```
# math_utils.py

def add(a, b):

    return a + b


def subtract(a, b):

    return a - b


PI = 3.14159
```

Packages

Definition: A package is a directory that contains multiple modules and possibly other packages. It allows you to organize related modules into a hierarchical structure. A package is identified by the presence of an __init__.py file (which can be empty) in the directory.

```
# mypackage/module1.py

def function1():
    return "Function 1 from module1"
```

What are lists and tuples? What is the key difference between the two

Lists

- **Mutable**: Lists are mutable, meaning you can change, add, or remove elements after the list is created.
- **Ordered**: Lists maintain the order of the elements as they are inserted.
- **Syntax**: Lists are defined using square brackets [].
- **Dynamic**: Lists can grow or shrink in size as elements are added or removed.

Tuples

- **Immutable**: Tuples are immutable, meaning once a tuple is created, you cannot change, add, or remove elements.
- **Ordered**: Tuples also maintain the order of elements.
- **Syntax**: Tuples are defined using parentheses () or simply by separating items with commas.
- **Fixed Size**: Since they are immutable, tuples have a fixed size after creation.

What is an Interpreted language & dynamically typed language?Write 5 differences between them?

Interpreted Language

An interpreted language is a type of programming language in which most of the code is executed directly by an interpreter rather than being compiled into machine code ahead of time. Examples of interpreted languages include Python, Ruby, and JavaScript.

Characteristics:

- **Execution**: Code is executed line-by-line by an interpreter.
- **Portability**: Generally more portable across different platforms because they rely on an interpreter available for each platform.
- **Ease of Debugging**: Often easier to debug because you can execute code incrementally and see results immediately.

Dynamically Typed Language

A dynamically typed language is a type of programming language in which the type of a variable is determined at runtime rather than at compile-time. Examples of dynamically typed languages include Python, JavaScript, and Ruby.

Characteristics:

- **Type Checking**: Variable types are checked at runtime, which can provide flexibility in how variables are used.
- **Ease of Use**: Often easier to write and read because you don't need to declare variable types explicitly.
- **Flexibility**: Allows for more flexible and concise code but can lead to runtime errors if not used carefully.

Differences Between Interpreted Languages and Dynamically Typed Languages

1. **Aspect of Language:**
    - **Interpreted Language**: Refers to how the code is executed (i.e., by an interpreter rather than being compiled).
    - **Dynamically Typed Language**: Refers to how and when the types of variables are determined (i.e., at runtime).
2. **Execution vs. Typing:**
    - **Interpreted Language**: Focuses on the execution model where the code is run by an interpreter.
    - **Dynamically Typed Language**: Focuses on the type system where variable types are determined dynamically.
3. **Portability:**
    - **Interpreted Language**: Often more portable across different systems since the interpreter can be implemented for multiple platforms.
    - **Dynamically Typed Language**: Portability is not directly related to the typing system; rather, it's more about how types are handled.
4. **Performance:**
    - **Interpreted Language**: Typically slower execution because the code is parsed and executed on the fly.
    - **Dynamically Typed Language**: Performance impact is not directly related to the typing system but can introduce runtime overhead due to type checking.
5. **Flexibility and Ease of Use:**
    - **Interpreted Language**: Allows for easier experimentation and rapid development cycles because of the immediate execution feedback.
    - **Dynamically Typed Language**: Provides flexibility in coding by allowing variable types to change, which can simplify code development and readability.

What are Dict and List comprehensions

List Comprehensions

List comprehensions offer a compact way to create lists. They consist of an expression followed by a for clause, and can also include optional if conditions.

Dictionary Comprehensions

Dictionary comprehensions allow for the creation of dictionaries in a similarly concise manner. They use a similar syntax but with key-value pairs.

Key Points

- **Readability**: Comprehensions make the code more readable and concise by reducing the need for explicit loops and temporary variables.
- **Efficiency**: Comprehensions are often more efficient than using traditional loops because they are optimized for performance in Python.
- **Flexibility**: They support conditional logic, allowing you to filter items dynamically.

What are decorators in Python? Explain it with an example.Write down its use cases ?

Decorators in Python

Decorators are a powerful feature in Python that allows you to modify the behavior of functions or methods. They are a way to wrap another function in order to extend or alter its behavior without permanently modifying the original function.

```
def function():

    pass
```

Use Cases:

1. **Logging**: Automatically log function calls, parameters, and return values.
2. **Authorization**: Check if a user has the right permissions to execute a function.
3. **Caching**: Cache results of expensive function calls and reuse them.
4. **Timing**: Measure the time taken by a function to execute.
5. **Validation**: Validate inputs to a function before executing it.
6. **Retry Logic**: Retry a function call if it fails due to transient errors.
7. **Resource Management**: Manage resources like opening and closing files or database connections.

How is memory managed in Python

Memory Management in Python

Python manages memory using a private heap containing all Python objects and data structures. Here's how memory management works:

1. **Reference Counting**: Python uses reference counting to track objects' memory allocation. Each object has a reference count, which increases when a reference to the object is created and decreases when a reference is deleted. When the reference count reaches zero, the memory occupied by the object is deallocated.
2. **Garbage Collection**: Python also includes a garbage collector to detect and clean up circular references (objects that reference each other). The garbage collector uses algorithms like generational collection to efficiently manage memory.

3. **Memory Pools**: Python uses a memory allocator for small objects called pymalloc, which optimizes memory allocation for small and frequent allocations.

What is lambda in Python? Why is it used

Lambda in Python

A **lambda** function is an anonymous function expressed as a single statement. It can have any number of input parameters but only one expression.

A lambda function to add two numbers

add = lambda x, y: x + y

print(add(2, 3)) # Output: 5

Use Cases:

- **Short Functions**: Useful for creating short, throwaway functions without formally defining them using def.
- **Functional Programming**: Often used in higher-order functions like map(), filter(), and reduce().
- **Sorting and Key Functions**: Used to define simple custom sort or key extraction functions.

Explain split() and join() functions in Python

split()

The split() function is used to split a string into a list of substrings based on a specified delimiter.

separator: The delimiter string. Default is whitespace.

maxsplit: Maximum number of splits to do. Default is -1 (all occurrences).

join()

The join() function is used to concatenate a list of strings into a single string with a specified delimiter.

```
words = ['Hello', 'world', 'Welcome', 'to', 'Python']
sentence = " ".join(words)  # Using space as separator
print(sentence)  # Output: Hello world Welcome to Python

fruits = ['apple', 'banana', 'cherry']
csv_line = ",".join(fruits)  # Using comma as separator
print(csv_line)  # Output: apple,banana,cherry
```

Both split() and join() are essential for text processing and manipulation tasks in Python.

What are iterators , iterable & generators in Python

Iterables are objects that can return an iterator, one element at a time. Examples of iterable objects include lists, tuples, strings, and dictionaries. An iterable implements the __iter__() method, which returns an iterator.

Iterators are objects that represent a stream of data. They implement the __iter__() and __next__() methods, allowing you to traverse through all the elements in an iterable. When you call next() on an iterator, it returns the next element. If there are no more elements, it raises a StopIteration exception

Generators are a special type of iterator that is created using a function with the yield statement. They allow you to iterate over data without storing the entire dataset in memory, which is useful for handling large data sets efficiently.

What is the difference between xrange and range in Python

range(): Returns a list containing all the numbers within the specified range. It generates all numbers at once, which can lead to high memory usage for large ranges.

xrange(): Returns an xrange object, which generates numbers on the fly and is more memory efficient for large ranges.

Pillars of Oops?

Object-oriented programming is based on four main principles:

1. **Encapsulation**: Bundling data and methods that operate on the data within one unit (e.g., a class) and restricting access to some of the object's components. This is done using access modifiers.
2. **Abstraction**: Hiding complex implementation details and showing only the essential features of an object. This allows users to work with objects at a higher level without needing to understand the inner workings.
3. **Inheritance**: Creating new classes based on existing ones to promote code reuse and establish a relationship between classes. It allows a class to inherit attributes and methods from another class.
4. **Polymorphism**: Allowing objects to be treated as instances of their parent class, enabling a single function or method to work in different ways depending on the object it is acting upon. This is usually achieved through method overriding and operator overloading.

How will you check if a class is a child of another class

Checking if a Class is a Child of Another Class

In Python, you can check if a class is a subclass of another class using the issubclass() function or by using the isinstance() function to check if an object is an instance of a specific class or its subclass.

How does inheritance work in python? Explain all types of inheritance with an example?

Inheritance in Python

Inheritance allows a class to inherit attributes and methods from another class, promoting code reuse and establishing a hierarchy.

Types of Inheritance:

- **Single Inheritance**: A child class inherits from a single parent class.
- **Multiple Inheritance**: A child class inherits from multiple parent classes.
- **Multilevel Inheritance**: A class is derived from another class, which is also derived from another class.
- **Hierarchical Inheritance**: Multiple child classes inherit from the same parent class.
- **Hybrid Inheritance**: A combination of two or more types of inheritance.

What is encapsulation? Explain it with an example?

Encapsulation in Python

Encapsulation is the concept of wrapping data and methods into a single unit, a class, and restricting access to the inner workings of that class. It is used to prevent direct access to data, which helps in data hiding and protecting the integrity of the object.

In Python, encapsulation is implemented using private and protected access specifiers.

- **Private members** are declared with a double underscore __ prefix, making them inaccessible from outside the class.
- **Protected members** are declared with a single underscore _ prefix, indicating that they should not be accessed directly, but they can be accessed by subclasses.

```python
class MyClass:
    def __init__(self):
        self.__private_variable = "I am private"
        self._protected_variable = "I am protected"
        self.public_variable = "I am public"

    def __private_method(self):
        print("This is a private method.")

    def _protected_method(self):
        print("This is a protected method.")

    def public_method(self):
        print("This is a public method.")


obj = MyClass()
print(obj.public_variable)
obj.public_method()


print(obj._protected_variable)
obj._protected_method()


print(obj._MyClass__private_variable)
obj._MyClass__private_method()
```

What is polymorphism? Explain it with an example.

Polymorphism in Python

Polymorphism is the ability of different classes to be treated as instances of the same class through a common interface. It allows the same function or method to behave differently depending on the object it is called upon.

There are two main types of polymorphism:

1. **Method Overriding**: A subclass provides a specific implementation for a method that is already defined in its parent class.

```
class Animal:
    def sound(self):
        print("Some generic sound")

class Dog(Animal):
    def sound(self):
        print("Woof!")

class Cat(Animal):
    def sound(self):
```

Question 1. 2. Which of the following identifier names are invalid and why?

1st_Room: Invalid because it starts with a digit (1).
**Hundred$**: Invalid because it contains a special character ($).
total-Marks: Invalid because it contains a hyphen (-), which is not allowed.
Total Marks: Invalid because it contains a space.
True: Invalid because True is a reserved keyword in Python..

Serial_no: Valid because it contains only alphanumeric characters and underscores, and it doesn't start with a digit.
Total_Marks: Valid for the same reasons as above.
_Percentag: Valid because it starts with an underscore, which is allowed.