In [ ]:

```
Data types, variable,
Decorator, Iterator, Generator
Lamda, map, filter
Data handling
Flask -
Tkinter -
SQL -
```

In [ ]:

```
1. Question: Write a function that takes a list of numbers and returns the sum.
    def sum_of_list(nums):
return sum(nums)
```

In [ ]:

```
2. Question: How do you reverse a string in Python?
def reverse_string(s):
return s[::-1]
```

In [ ]:

```
3. Question: What is the difference between a tuple and a list?
Answer: Lists are mutable (can be modified),
while tuples are immutable (cannot be modified once created).
```

In [ ]:

```
Intermediate
```

In [ ]:

```
4. Question: Write a function that checks if a given word is a palindrome.
    def is_palindrome(word):
return word == word[::-1]
```

In [ ]:

```
5. Question: How can you remove duplicates from a list?
def remove_duplicates(lst):
return list(set(lst))
Note: This method will not maintain the original order of the list. To maintain the
you can use a loop or a list comprehension.
```

In [ ]:

```
6. Question: How do you handle exceptions in Python?
Answer: Using try and except blocks. For example:
try:
x = 1 / 0
except ZeroDivisionError:
print("Cannot divide by zero!")
```

In [ ]:

```
Advanced
```

In [ ]:

```
7. Question: What is a lambda function? Provide an example.
```

In [ ]:

```
Answer: A lambda function is a small, anonymous function.
It can take any number of arguments but can only have one expression.
    multiply = lambda x, y: x * y
print(multiply(3, 4)) # Output: 12
```

In [ ]:

```
Find Factorial
Question: Calculate the factorial of a number.
Answer:
```

```
def factorial(n):
if n <= 1:
return 1
return n * factorial(n-1)
```

In [ ]:

```
8. Question: Write a function that returns the n-th Fibonacci number using recursio
Answer:
def fibonacci(n):
if n <= 1:
return n
else:
return fibonacci(n-1) + fibonacci(n-2)
```

In [ ]:

```
9. Question: What are decorators in Python and how are they used?
Answer: Decorators provide a way to modify or extend the behavior of callable objec
often used in frameworks to add functionality to functions or methods.
Example:
def my_decorator(func):
def wrapper():
print("Something is happening before the function is called.")
func()
print("Something is happening after the function is called.")
return wrapper
@my_decorator
def say_hello():
print("Hello!")
say_hello()
```

In [ ]:

```
Intermediate
```

In [ ]:

```
10. Question: How do you deep copy an object in Python?
Answer: You can use the copy module's deepcopy method.
import copy
original = [[1, 2, 3], [4, 5, 6]]
copied = copy.deepcopy(original)
```

In [ ]:

```
11. Question: What is list comprehension and provide an example?
Answer: List comprehension is a concise way to create lists in Python.
squared_numbers = [x**2 for x in range(10)]
```

In [ ]:

```
12. Question: What is the difference between == and is?
Answer: == checks for value equality, while is checks for identity
(whether two references point to the same object in memory).
```

In [ ]:

```
Advanced
```

In [ ]:

```
13. Question: Explain the concept of *args and **kwargs in Python.
Answer: *args allows you to pass a variable number of positional arguments to a fun
while **kwargs allows you to pass a variable number of keyword arguments.
Example:
def function_example(*args, **kwargs):
for arg in args:
print(arg)
for key, value in kwargs.items():
```

```
print(f"{key} = {value}")
function_example(1, 2, 3, a=4, b=5)
```

In [ ]:

```
14. Question: What is a generator and how is it different from a list?
Answer: A generator is an iterable that yields items one at a time using a yield
statement, whereas a list holds all its items in memory.
    Generators are more memory-efficient
for large data sets.
Example of a generator:
def count_up_to(n):
count = 1
while count <= n:
yield count
count += 1
```

In [ ]:

```
15. Question: How can you achieve multi-threading in Python?
Answer: Python has a threading module which can be used to achieve
multi-threading.
    import threading
def print_numbers():
for i in range(1, 6):
print(i)
def print_letters():
for letter in 'abcde':
print(letter)
t1 = threading.Thread(target=print_numbers)
t2 = threading.Thread(target=print_letters)
t1.start()
t2.start()
t1.join()
t2.join()
```

In [ ]:

```
16. Question: What are metaclasses in Python?
Answer: Metaclasses are a deep and advanced topic in Python. Essentially,
they are "classes of a class" that define how a class behaves.
The default metaclass is type, but you can create your own metaclass to
customize class behavior.
```

In [ ]:

```
17. Question: Describe the Global Interpreter Lock (GIL) and its implications.
Answer: The GIL is a mutex (or a lock) that allows only one thread to execute Pytho
bytecode at a time in CPython (the standard Python implementation). This means that
even on multi-core systems, only one thread is executed at a time. This can be a
bottleneck for CPU-bound programs, but it is not generally an issue for I/O-bound
programs.
I hope these additional questions assist in your preparations! If you need more or
Here are more Python-related interview questions, spanning from intermediate to adv
```

In [ ]:

```
Intermediate
```

In [ ]:

```
18. Question: What is the difference between staticmethod, classmethod, and regular
Answer:
staticmethod: Doesn't take any specific first parameter (neither self nor cls),
    and acts just like a regular function but belongs to a class's namespace.
classmethod: Takes the class as its first parameter (usually named cls).
```

```
    It can be called on the class itself, rather than an instance.
Instance method: Takes the instance (object) as its first parameter
(usually named self) and operates on it.
```

In [ ]:

```
19. Question: How do you sort a dictionary by its values?
Answer:
d = {'apple': 15, 'banana': 10, 'cherry': 20}
sorted_d = dict(sorted(d.items(), key=lambda item: item[1]))
```

In [ ]:

```
20. Question: How is string interpolation done in Python?
Answer: There are several ways:
Using % formatting.
Using .format() method.
Using f-strings (from Python 3.6+).
name = 'Alice'
# Using % formatting
print("Hello, %s!" % name)
# Using .format()
print("Hello, {}!".format(name))
# Using f-strings
print(f"Hello, {name}!")
```

In [ ]:

```
Advanced
```

In [ ]:

```
21. Question: How can you implement a singleton pattern in Python?
Answer: One way is by using a class attribute to check if an instance already
exists. If it does, return that. Otherwise, create a new instance.
class Singleton:
_instance = None
def __new__(cls):
if cls._instance is None:
cls._instance = super().__new__(cls)
return cls._instance
```

In [ ]:

```
22. Question: What is the difference between shallow copy and deep copy?
Answer:
Shallow copy: Creates a new object, but does not create copies of objects that
the original object references. Instead, it copies references.
Deep copy: Creates a new object and also recursively creates copies of objects
found in the original.
```

In [ ]:

```
23. Question: How do you swap two variables in Python?
Answer: In Python, swapping can be done without a temporary variable.
a, b = 5, 10
a, b = b, a
```

In [ ]:

```
24. Question: Explain Python's garbage collection process.
Answer: Python uses reference counting and a cyclic garbage collector.
Reference counting means objects are automatically deallocated once their reference
count drops to zero. The cyclic garbage collector finds and cleans up reference
cycles, which are situations where a group of objects reference each other
but are not referenced anywhere else.
```

In [ ]:

```
25. Question: What are Python's magic (dunder) methods, and how are they used?
Provide an example.
Answer: Magic or dunder (double underscore) methods in Python are special
methods that have double underscores at the beginning and end of their names.
They allow developers to emulate built-in behavior or implement operator overloadin
Example:
class Book:
def __init__(self, pages):
self.pages = pages
def __add__(self, other):
return Book(self.pages + other.pages)
book1 = Book(100)
book2 = Book(150)
book3 = book1 + book2
print(book3.pages) # Output: 250
```

In [ ]:

```
26. Question: How do you check if a variable is an instance of a particular type?
Answer: You can use the isinstance() function.
x = [1, 2, 3]
if isinstance(x, list):
print("x is a list")
```

In [ ]:

```
27. Question: What does the else clause in a loop do?
Answer: The else clause in a loop is executed when the loop finishes execution
(i.e., when the loop condition becomes False). It won't execute if the loop was
exited using a break statement.
for i in range(5):
print(i)
else:
print("Loop finished")
```

In [ ]:

```
28. Question: What is the purpose of the pass statement in Python?
Answer: The pass statement is a no-op (does nothing). It's used as a placeholder wh
code.
```

In [ ]:

```
29. Question: How do you retrieve all the keys, values, and items from a dictionary
Answer: You can use the methods keys(), values(), and items() respectively.
d = {"a": 1, "b": 2}
print(d.keys()) # dict_keys(['a', 'b'])
print(d.values()) # dict_values([1, 2])
print(d.items()) # dict_items([('a', 1), ('b', 2)])
```

In [ ]:

```
Advanced
```

In [ ]:

```
30. Question: What is the difference between __new__ and __init__ in a class?
Answer: __new__ is responsible for creating and returning a new instance of the
class, while __init__ is responsible for initializing the created object.
```

In [ ]:

```
31. Question: What is the difference between an Iterable and an Iterator?
Answer:
Iterable: An object which has an __iter__ method that returns an iterator.
Iterator: An object that can return its items one at a time using
the __next__ method and implements the __iter__ method.
```

In [ ]:

```
32. Question: How does the map function work in Python?
Answer: The map function applies a given function to all the items in an input list
numbers = [1, 2, 3, 4]
squared = map(lambda x: x**2, numbers)
print(list(squared)) # Output: [1, 4, 9, 16]
```

In [ ]:

```
33. Question: What are context managers in Python? Provide an example.
Answer: Context managers allow resources to be properly managed and cleaned up
after use. The most common example is opening files using the with statement.

with open('file.txt', 'r') as file:
content = file.read()
Here, the file is automatically closed after reading, even if an exception occurs w
```

In [ ]:

```
34. Question: How can you dynamically create a new class at runtime?
Answer: You can use the type() function.
MyClass = type('MyClass', (object,), {'x': 10})
obj = MyClass()
print(obj.x) # Output: 10
```

In [ ]:

```
35. Question: What does the zip function do in Python?
Answer: The zip function takes two or more iterables as arguments and returns an it
paired based on their order.
names = ['Alice', 'Bob', 'Charlie']
ages = [25, 30, 35]
result = zip(names, ages)
print(list(result)) # Output: [('Alice', 25), ('Bob', 30), ('Charlie', 35)]
```

In [ ]:

```
36. Question: How can you merge two dictionaries?
Answer: In Python 3.5+, you can use the {**d1, **d2} syntax or the update() method.
d1 = {'a': 1, 'b': 2}
d2 = {'b': 3, 'c': 4}
merged = {**d1, **d2}
print(merged) # Output: {'a': 1, 'b': 3, 'c': 4}
```

In [ ]:

```
37. Question: What does the @property decorator do in Python?
Answer: The @property decorator allows you to define methods in a class that can be
the getter behavior.
class Circle:
def __init__(self, radius):
self._radius = radius
@property
def diameter(self):
return self._radius * 2
circle = Circle(5)
print(circle.diameter) # Output: 10
```

In [ ]:

```
38. Question: What is the difference between asyncio and multi-threading?
Answer: asyncio is a Python library used for writing concurrent code using the
async/await syntax. It's single-threaded and uses cooperative multitasking. On the
other hand,
multi-threading involves multiple threads of a single process, with
each thread executing independently and possibly concurrently.
```

In [ ]:

```
39. Question: How can you make an immutable class in Python?
Answer: By ensuring that all of its attributes are immutable and preventing any mod
or using private attributes.
class ImmutableClass:
def __init__(self, value):
self.__value = value
@property
def value(self):
return self.__value
```

In [ ]:

```
40. Question: What is the purpose of the __slots__ attribute in a Python class?
Answer: The __slots__ attribute is used to define a static set of attributes for in
creation of the default __dict__ for the object, which normally stores object attri
class MyClass:
__slots__ = ['x', 'y']
def __init__(self, x, y):
self.x = x
self.y = y
```

In [ ]:

```
41. Question: How can you run Python code in parallel?
Answer: You can use the multiprocessing module, which allows for the creation of
separate processes, or the concurrent.futures module, which provides a high-level
interface for asynchronously executing functions using threads or processes.
    from multiprocessing import Pool
def square(x):
return x * x
with Pool(4) as p: # Use 4 processes
result = p.map(square, [1, 2, 3, 4])
print(result) # Output: [1, 4, 9, 16]
```

In [ ]:

```
42. Question: What are Python descriptors?
Answer: Descriptors are objects that define the behavior of attributes in other
objects when they are accessed, set, or deleted. Descriptors are defined using at
least one of the
methods __get__, __set__, or __delete__.
Example:
class Descriptor:
def __get__(self, instance, owner):
return instance._value
def __set__(self, instance, value):
instance._value = value.upper()
class MyClass:
attribute = Descriptor()
def __init__(self, value):
self._value = value
obj = MyClass('hello')
print(obj.attribute) # hello
obj.attribute = 'world'
print(obj.attribute) # WORLD
```

In [ ]:

```
43. Question: How can you reverse a string in Python?
Answer: You can reverse a string using slicing.
s = "hello"
```

```
reversed_string = s[::-1]
print(reversed_string) # Output: "olleh"
```

In [ ]:

```
Question: What is the difference between a list and a tuple in Python?
Answer:
List:
Mutable, meaning you can modify its contents.
Defined using square brackets [].
**Tuple**: - Immutable, so once you create it, you can't alter its contents.
- Defined using parentheses `()`.
```

In [ ]:

```
45. Question: How can you catch multiple exceptions in a single line?
Answer: You can use a tuple to specify multiple exception types in a single except
try:
# some code
except (TypeError, ValueError) as e:
print(f"Caught an exception: {e}")
```

In [ ]:

```
46. Question: What is a metaclass in Python?
Answer: A metaclass in Python is a class of a class that defines how a class
behaves. In other words, just as a class defines how instances of the class behave,
a metaclass defines how classes themselves behave.
```

In [ ]:

```
47. Question: How do you define a class method and when would you use it?
Answer: A class method is a method that's bound to the class, not the instance. You
methods that are concerned with the class itself rather than specific instances.
class MyClass:
count = 0
@classmethod
def increment_count(cls, value):
cls.count += value
```

In [ ]:

```
48. Question: What is the Global Interpreter Lock (GIL)?
Answer: The GIL is a mutex in CPython (the default Python interpreter) that ensures
only one thread executes Python bytecode at a time, even on multi-core systems.
This is why multi-threaded CPU-bound programs may not see a performance improvement
    in CPython.
```

In [ ]:

```
49. Question: How can you achieve inheritance in Python?
Answer: Inheritance is achieved by defining a new class, derived from an
existing class. The derived class inherits attributes and behaviors of the base
class and can also have additional attributes or behaviors.
class Animal:
def speak(self):
pass
class Dog(Animal):
def speak(self):
return "Woof"
```

In [ ]:

```
50. Question: What is the super() function, and why might you use it?
Answer: The super() function returns a temporary object of the superclass,
allowing you to call its methods. It's commonly used in the __init__ method to
ensure that initializers of parent classes get called.
    class Animal:
```

```
def __init__(self, species):
self.species = species
class Dog(Animal):
def __init__(self, species, name):
super().__init__(species)
self.name = name
```

In [ ]:

```
Question: What is the __str__ method in a class and when is it used?
Answer: The __str__ method is a special method that should return a string represen
function when outputting the object.
class Person:
def __init__(self, name):
self.name = name
def __str__(self):
return f"Person named {self.name}"
p = Person("Alice")
print(p) # Output: "Person named Alice"
```

In [ ]:

```
52. Question: How can you remove duplicate items from a list?
Answer: One common way is to convert the list to a set and then back to a list.
mylist = [1, 2, 2, 3, 4, 4, 5]
mylist = list(set(mylist))
print(mylist) # Output: [1, 2, 3, 4, 5]
```

In [ ]:

```
53. Question: What are decorators in Python?
Answer: Decorators provide a way to modify or enhance functions or methods without
symbol above the function or method.
def my_decorator(func):
def wrapper():
print("Something is happening before the function is called.")
func()
print("Something is happening after the function is called.")
return wrapper
@my_decorator
def say_hello():
print("Hello!")
say_hello()
```

In [ ]:

```
54. Question: How can you implement method overloading in Python?
Answer: Python doesn't support explicit method overloading like some other language
argument lists, or keyword arguments.
class Greet:
def hello(self, name=None):
if name is not None:
print(f"Hello, {name}")
else:
print("Hello, ")
```

In [ ]:

```
56. Question: How can you achieve multi-level inheritance in Python?
Answer: Multi-level inheritance involves inheriting from a derived class,
forming a chain of inheritance.
class Grandparent:
pass
class Parent(Grandparent):
pass
```

```
class Child(Parent):
pass
```

In [ ]:

```
57. Question: What is the *args and **kwargs syntax in
function signatures, and how is it used?
Answer: *args and **kwargs are conventions used in Python to pass a variable
number of non-keyword and keyword arguments, respectively, to a function.
*args: Passes variable-length non-keyworded arguments list.
**kwargs: Passes variable-length keyworded arguments dictionary.
def function_example(*args, **kwargs):
for arg in args:
print(arg)
for key in kwargs:
print(f"{key} = {kwargs[key]}")
function_example(1, 2, 3, a=4, b=5)
Remember, these are just conventions; you could technically use *var and **vars,
but the aforementioned are widely recognized in the Python community.
```

In [ ]:

```
58. Question: How can you implement a stack in Python?
Answer: You can use a list to implement a stack, utilizing the append()
method for push operation and the pop() method for pop operation.
stack = []
stack.append(1) # Push
stack.append(2)
print(stack.pop()) # Pop: 2
```

In [ ]:

```
59. Question: What is the difference between a list and a dictionary?
Answer: A list is an ordered collection of items, while a dictionary is an
unordered collection of key-value pairs. Lists are indexed by integers, starting
from zero, whereas dictionaries are indexed by unique keys.
```

In [ ]:

```
60. Question: What is NumPy and when might you use it?
Answer: NumPy is a library for the Python programming language, adding support
for large, multi-dimensional arrays and matrices, along with a collection of
mathematical functions to operate on these arrays. It's often used in scientific
computing, data analysis, and machine learning for tasks that require mathematical
operations on large datasets.
```

In [ ]:

```
61. Question: What are virtual environments in Python, and why are they useful?
Answer: Virtual environments are tools that help to keep dependencies required by
different projects separate by creating isolated environments for them.
This is especially useful when different projects have different requirements and
can prevent conflicts between versions.
```

In [ ]:

```
62. Question: What is Flask?
Answer: Flask is a micro web framework written in Python. It does not include
built-in abstracted tools like form validation or database integration
but is lightweight and easily extensible, making it a popular choice
for small web applications or as a backend for more complex projects.
```

In [ ]:

```
63. Question: How can you create a basic route in Django?
Answer: In Django, a route is defined in urls.py using the url() function or path()
from django.urls import path
from . import views
```

```
urlpatterns = [
path('hello/', views.hello, name='hello'),
]
```

In [ ]:

```
64. Question: What is unittest in Python?
Answer: unittest is a built-in library in Python used for testing Python code.
It supports test automation, sharing of setup and shutdown code, aggregation of
tests into collections, and more.
```

In [ ]:

```
65. Question: How can you set a breakpoint in your code to aid debugging?
Answer: You can use the breakpoint() function (introduced in Python 3.7)
to set a breakpoint in your code. When the code execution reaches the breakpoint(),
it'll pause, allowing you to inspect the current state using a debugger.
```

In [ ]:

```
66. Question: What does the async keyword do in Python?
Answer: The async keyword is used to define asynchronous functions in Python.
These functions return an asynchronous iterator. To call them,
you'd typically use the await keyword.
Asynchronous functions allow for concurrency, meaning tasks can yield control
and let other tasks run without necessarily completing.
```

In [ ]:

```
67. Question: What is the difference between a thread and a coroutine?
Answer: A thread is a smallest unit of a process that runs concurrently with
other threads of the process, managed by the operating system. A coroutine,
on the other hand, is a generalization of a subroutine, allowing multiple
entry points and yielding control back to the caller without necessarily exiting.
Coroutines are cooperative, meaning they yield control by choice, whereas threads
can be preempted by the OS scheduler.
```

In [ ]:

```
68. Question: What is the threading module in Python?
Answer: The threading module in Python is used to create and manage threads.
Threads allow for parallel execution of code, which can lead to faster execution
for I/O-bound tasks.
```

In [ ]:

```
69. Question: How do you create and start a new thread using the threading module?
Answer: You can create a thread using threading.Thread and
then start it using the start() method.
import threading
def print_numbers():
for i in range(10):
print(i)
# Create a thread and start it
thread = threading.Thread(target=print_numbers)
thread.start()
```

In [ ]:

```
71. Question: How can you ensure thread-safety when accessing shared resources in
Python?
Answer: You can use locks, like threading.Lock, to ensure that only one thread
accesses a shared resource at a time.
import threading
lock = threading.Lock()
counter = 0
def increment_counter():
global counter
```

```
with lock:
counter += 1
print(counter)
threads = []
for _ in range(10):
thread = threading.Thread(target=increment_counter)
thread.start()
threads.append(thread)
for thread in threads:
thread.join()
```

In [ ]:

```
72. Question: What's the difference between a Thread and a ThreadPoolExecutor in
Python?
Answer: While Thread allows you to manage individual threads, ThreadPoolExecutor
from the concurrent.futures module provides a higher-level interface for
asynchronously executing callables. It manages a pool of worker threads,
which can be more efficient than spawning a new thread for every task, especially
for a large number of small tasks.
from concurrent.futures import ThreadPoolExecutor
def task(n):
return n * n
with ThreadPoolExecutor(max_workers=4) as executor:
results = list(executor.map(task, range(10)))
print(results)
```

In [ ]:

```
73. Question: What is a Semaphore, and how can it be useful in threading?
Answer: A Semaphore is a synchronization primitive that maintains a count between z
an acquire() method to decrease it. Semaphores can be used to control access to a r
import threading
semaphore = threading.Semaphore(2)
def access_resource(tid):
print(f"Thread {tid} waiting")
with semaphore:
print(f"Thread {tid} accessing")
# simulate some work
threading.sleep(2)
print(f"Thread {tid} releasing")
threads = [threading.Thread(target=access_resource, args=(i,)) for i in range(4)]

for thread in threads:
thread.start()
for thread in threads:
thread.join()
These are just a few sample questions on threading in Python. The topic can be quit
like deadlock and race conditions.
```

In [ ]:

```
74. Question: What is a deadlock and how can you avoid it?
Answer: A deadlock is a situation in which two or more threads are unable to
proceed with their execution because each is waiting for the other to release a
resource. Deadlocks can be avoided by:
Ensuring that locks are always acquired in a fixed order.
Using timeouts when trying to acquire locks.
Deadlock detection, where the system periodically checks for deadlock conditions
and breaks them.
# An example of a potential deadlock situation:
```

```
import threading
lock1 = threading.Lock()
lock2 = threading.Lock()
def worker1():
with lock1:
with lock2:
print("Worker 1")
def worker2():
with lock2: # If worker1 and worker2 try to acquire the locks at the same time,
                #a deadlock can occur.
with lock1:
print("Worker 2")
```

In [ ]:

```
75. Question: How can you share data between threads?
Answer: Data can be shared between threads using global variables or by
passing data structures like lists or dictionaries to the thread functions.
However, care must be taken to synchronize access to shared data to prevent
race conditions.
import threading
data = []
def worker(value):
global data
data.append(value)
threads = [threading.Thread(target=worker, args=(i,)) for i in range(5)]
for thread in threads:
thread.start()
for thread in threads:
thread.join()
print(data)
```

In [ ]:

```
76. Question: What is a Condition object in threading, and how is it used?
Answer: A Condition object provides a way for one thread to wait for a condition
to be satisfied by another thread. It uses a lock internally and provides methods
like wait(),
notify(), and notify_all().
import threading
condition = threading.Condition()
data = []
def producer():
for i in range(5):
with condition:
data.append(i)
condition.notify()
def consumer():
with condition:
while not data:
condition.wait()
print(data.pop(0))
thread1 = threading.Thread(target=producer)
thread2 = threading.Thread(target=consumer)
thread1.start()
thread2.start()
thread1.join()
thread2.join()
```

In [ ]:

```
77. Question: What is a Barrier in threading?
Answer: A Barrier is a threading primitive that blocks until a specified number
of threads have reached it. Once that number is reached, all waiting threads are
released
simultaneously.
import threading
barrier = threading.Barrier(3)
def worker(tid):
print(f"Thread {tid} waiting")
barrier.wait()
print(f"Thread {tid} proceeding")
threads = [threading.Thread(target=worker, args=(i,)) for i in range(3)]
for thread in threads:
thread.start()
for thread in threads:
thread.join()
```

In [ ]:

```
78. Question: What is a race condition? Provide an example.
Answer: A race condition occurs when two or more threads can access shared data
and try to change it at the same time. The result of the change depends on the
timing of how
the threads run.
import threading
counter = 0
def increment():
global counter
for _ in range(1000000):
counter += 1
thread1 = threading.Thread(target=increment)
thread2 = threading.Thread(target=increment)
thread1.start()
thread2.start()
thread1.join()
thread2.join()
print(counter) # Expected 2000000, but due to race condition, the result might
            #be different.
```

In [ ]:

```
Question: Given a list of numbers, write a Python function to find the second
highest number.
Answer: We can first convert the list into a set to remove duplicates.
    Then, we'll convert it back to a list and sort it. We can retrieve
    the second last element to get the second highest number.
def second_highest(numbers):
numbers = list(set(numbers))
numbers.sort()
return numbers[-2]
numbers = [1, 3, 2, 4, 4, 5, 6, 6]
print(second_highest(numbers)) # Output: 5
```

In [ ]:

```
2. Question: Write a function to compute the factorial of a number using recursion.
Answer:
def factorial(n):
if n == 0:
return 1
return n * factorial(n-1)
number = 5
print(factorial(number)) # Output: 120
```

In [ ]:

```
3. Question: You are given a list of strings. Write a function to filter out all
strings that are palindromes.
Answer: A palindrome is a word, phrase, number, or other sequences of characters
that reads the same forward and backward (ignoring spaces, punctuation,
                                        and capitalization).
def is_palindrome(s):
s = ''.join(e for e in s if e.isalnum()) # Remove punctuation and spaces
return s.lower() == s.lower()[::-1]
def filter_palindromes(strings):
return [s for s in strings if is_palindrome(s)]
words = ["radar", "python", "level", "world"]
print(filter_palindromes(words)) # Output: ['radar', 'level']
```

In [ ]:

```
4. Question: Given a string, write a function to check if it is an anagram of
another string.
Answer: An anagram is a word or phrase formed by rearranging the letters of a
different word or phrase, typically using all the original letters exactly once.
def are_anagrams(s1, s2):
return sorted(s1) == sorted(s2)
str1 = "listen"
str2 = "silent"
print(are_anagrams(str1, str2)) # Output: True
```

In [ ]:

```
5. Question: Write a function to flatten a nested list.
Answer:
def flatten(lst):
result = []
for i in lst:
if isinstance(i, list):
result.extend(flatten(i))
else:
result.append(i)
return result
nested_list = [1, [2, 3, [4, 5]], 6]
print(flatten(nested_list)) # Output: [1, 2, 3, 4, 5, 6]
Remember, these solutions can be optimized or presented in different ways depending
on the context and requirements of the interview.
```

In [ ]:

```
6. Question: Given two lists, write a function that returns the elements
that are common to both lists.
Answer:
def common_elements(list1, list2):
return list(set(list1) &amp; set(list2))
list1 = [1, 2, 3, 4, 5]
list2 = [4, 5, 6, 7, 8]
print(common_elements(list1, list2)) # Output: [4, 5]
```

In [ ]:

```
7. Question: Write a function that returns the number of words in a string.
Answer:
def word_count(s):
return len(s.split())
```

```
sentence = "The quick brown fox"
print(word_count(sentence)) # Output: 4
```

In [ ]:

```
8. Question: Write a Python function to merge two dictionaries.
    If both dictionaries have the same key, prefer the second dictionary's value.
Answer:
def merge_dicts(dict1, dict2):
merged = dict1.copy()
merged.update(dict2)
return merged
dict1 = {'a': 1, 'b': 2}
dict2 = {'b': 3, 'c': 4}
print(merge_dicts(dict1, dict2)) # Output: {'a': 1, 'b': 3, 'c': 4}
```

In [ ]:

```
9. Question: Write a function that finds the most repeated character in a string.
Answer:
def most_repeated(s):
char_count = {}
for char in s:
if char in char_count:
char_count[char] += 1
else:
char_count[char] = 1
max_char = max(char_count, key=char_count.get)
return max_char
string = "aabbbcddddde"
print(most_repeated(string)) # Output: 'd'
```

In [ ]:

```
10. Question: Write a function that checks if a string contains all letters of
the alphabet at least once.
Answer:
import string
def contains_all_alphabets(s):
alphabet = set(string.ascii_lowercase)
return set(s.lower()) >= alphabet
test_string = "The quick brown fox jumps over the lazy dog"
print(contains_all_alphabets(test_string)) # Output: True
```

In [ ]:

```
11. Question: Write a function that checks if a given string is a valid IPv4 addres
Answer:
def is_valid_ipv4(ip):
parts = ip.split(".")
if len(parts) != 4:
return False
for item in parts:
if not item.isdigit():
return False
num = int(item)
if num < 0 or num > 255:
return False
return True
address = "192.168.1.1"
print(is_valid_ipv4(address)) # Output: True
```

In [ ]:

```
12. Question: Given a list of numbers, write a function to compute the mean, median
Answer:
from statistics import mean, median, mode
def compute_stats(numbers):
return {
"mean": mean(numbers),
"median": median(numbers),
"mode": mode(numbers)
}
numbers = [1, 2, 3, 4, 4, 5, 5, 5, 6]
print(compute_stats(numbers)) # Output: {'mean': 3.89, 'median': 4, 'mode': 5}
```

In [ ]:

```
13. Question: Write a function to compute the Fibonacci series up to n.
Answer:
def fibonacci(n):
series = [0, 1]
while len(series) < n:
series.append(series[-1] + series[-2])
return series
number = 10
print(fibonacci(number)) # Output: [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

In [ ]:

```
14. Question: Given a string, write a function that returns the first non-repeated
Answer:
def first_non_repeated(s):
char_count = {}
for char in s:
if char in char_count:
char_count[char] += 1
else:
char_count[char] = 1
for char in s:
if char_count[char] == 1:
return char
return None
string = "swiss"
print(first_non_repeated(string)) # Output: 'w'
```

In [ ]:

```
15. Question: Write a function to check if two strings are a rotation of each other
Answer:
def are_rotations(str1, str2):
if len(str1) != len(str2):
return False
return str1 in str2 + str2
s1 = "abcde"
s2 = "cdeab"
print(are_rotations(s1, s2)) # Output: True
```

In [ ]:

```
16. Question: Write a function to determine if a string has all unique characters
    (i.e., no character is repeated).
Answer:
def has_unique_chars(s):
return len(s) == len(set(s))
```

```
string = "abcdef"
print(has_unique_chars(string)) # Output: True
```

In [ ]:

```
17. Question: Write a function that returns the longest consecutive subsequence
in a list of numbers.
Answer:
def longest_consecutive_subsequence(nums):
if not nums:
return []
nums = sorted(set(nums))
longest_streak = []
current_streak = [nums[0]]
for i in range(1, len(nums)):
if nums[i] - nums[i - 1] == 1:
current_streak.append(nums[i])
else:
if len(current_streak) > len(longest_streak):
longest_streak = current_streak
current_streak = [nums[i]]
return longest_streak if len(longest_streak) > len(current_streak)
else current_streak
numbers = [1, 2, 3, 5, 6, 7, 8, 10]
print(longest_consecutive_subsequence(numbers)) # Output: [5, 6, 7, 8]
```

In [ ]:

```
18. Question: Write a function to compute the square root of a given
non-negative integer n without using built-in square root functions or libraries.
    Return the floor value of the result.
Answer:
def sqrt(n):
if n < 0:
return None
if n == 1:
return 1
start, end = 0, n
while start <= end:
mid = (start + end) // 2
if mid * mid == n:
return mid
elif mid * mid < n:
start = mid + 1
ans = mid
else:
end = mid - 1
return ans
number = 17
print(sqrt(number)) # Output: 4
```

In [ ]:

```
19. Question: Given a list of integers, write a function to move all zeros
to the end of the list while maintaining the order of the other elements.
Answer:
def move_zeros(nums):
count = nums.count(0)
nums = [num for num in nums if num != 0]
nums.extend([0] * count)
return nums
```

```
numbers = [1, 2, 0, 4, 0, 5, 6, 0]
print(move_zeros(numbers)) # Output: [1, 2, 4, 5, 6, 0, 0, 0]
```

In [ ]:

```
20. Question: Write a function that returns the sum of two numbers represented
as strings. Your function should not use built-in arithmetic operators or functions
Answer:
def add_strings(num1, num2):
res, carry, i, j = "", 0, len(num1) - 1, len(num2) - 1
while i >= 0 or j >= 0 or carry:
n1 = int(num1[i]) if i >= 0 else 0
n2 = int(num2[j]) if j >= 0 else 0
temp_sum = n1 + n2 + carry
res = str(temp_sum % 10) + res
carry = temp_sum // 10
i, j = i - 1, j - 1
return res
n1 = "123"
n2 = "789"
print(add_strings(n1, n2)) # Output: "912"
```

In [ ]:

```
21. Question: Write a function that checks if a given binary tree is a valid
binary search tree.
Answer:
class TreeNode:
def __init__(self, value=0, left=None, right=None):
self.value = value
self.left = left
self.right = right
def is_valid_bst(root, left=None, right=None):
if not root:
return True
if left and root.value <= left.value:
return False
if right and root.value >= right.value:
return False
return is_valid_bst(root.left, left, root) and is_valid_bst(root.right, root, right
# Example usage:
root = TreeNode(2, TreeNode(1), TreeNode(3))
print(is_valid_bst(root)) # Output: True
```

In [ ]:

```
22. Question: Write a function to find the longest common prefix of a list of strin
Answer:
def longest_common_prefix(strings):
if not strings:
return ""
prefix = strings[0]
for s in strings[1:]:
while not s.startswith(prefix):
prefix = prefix[:-1]
return prefix
strings = ["flower", "flow", "flight"]
print(longest_common_prefix(strings)) # Output: "fl"
```

In [ ]:

```
23. Question: Write a function that returns the intersection of two sorted arrays.
    Assume each array does not have duplicates.
```

```
Answer:
def intersection_of_sorted_arrays(nums1, nums2):
i, j = 0, 0
intersection = []
while i < len(nums1) and j < len(nums2):
if nums1[i] == nums2[j]:
intersection.append(nums1[i])
i += 1
j += 1
elif nums1[i] < nums2[j]:
i += 1
else:
j += 1
return intersection
arr1 = [1, 2, 4, 5, 6]
arr2 = [2, 3, 5, 7]
print(intersection_of_sorted_arrays(arr1, arr2)) # Output: [2, 5]
```

In [ ]:

```
24. Question: Write a function to determine if two strings are one edit
(or zero edits) away.
Answer:
def is_one_edit_away(s1, s2):
if abs(len(s1) - len(s2)) > 1:
return False
if len(s1) > len(s2):
s1, s2 = s2, s1
i, j, found_difference = 0, 0, False
while i < len(s1) and j < len(s2):
if s1[i] != s2[j]:
if found_difference:
return False
found_difference = True
if len(s1) == len(s2):
i += 1
else:
i += 1
j += 1
return True
print(is_one_edit_away("pale", "ple")) # Output: True
print(is_one_edit_away("pales", "pale")) # Output: True
print(is_one_edit_away("pale", "bale")) # Output: True
print(is_one_edit_away("pale", "bake")) # Output: False
```

In [ ]:

```
25. Question: Write a function that returns the shortest path in a maze from a
start point to an end point, given that you can only move up, down, left, or right.
The maze is represented as a 2D list where 0 represents an open path and 1 represen
Answer:
def shortest_path(maze, start, end):
if not maze or not maze[0]:
return None
from collections import deque
queue = deque([(start, 0)])
visited = set([start])
while queue:
(x, y), steps = queue.popleft()
```

```
if (x, y) == end:
return steps
for dx, dy in [(-1, 0), (1, 0), (0, -1), (0, 1)]:
nx, ny = x + dx, y + dy
if 0 <= nx < len(maze) and 0 <= ny < len(maze[0]) and maze[nx][ny] == 0 and (nx, ny
visited.add((nx, ny))
queue.append(((nx, ny), steps + 1))
return -1
maze = [
[0, 1, 0, 0, 0],
[0, 1, 0, 1, 0],
[0, 0, 0, 1, 0],
[0, 1, 1, 1, 1],
[0, 0, 0, 0, 0]
    ]
start = (0, 0)
end = (4, 4)
print(shortest_path(maze, start, end)) # Output: 12 (or -1 if there's no path)
Remember to adapt and explain your code as necessary during an interview,
ensuring you understand every line and are prepared to discuss alternative
solutions or optimizations.
```

In [ ]:

```
26. Question: Write a function that returns the nth number in the Fibonacci
sequence using recursion.
Answer:
def fibonacci_recursive(n):
if n <= 1:
return n
else:
return fibonacci_recursive(n-1) + fibonacci_recursive(n-2)
print(fibonacci_recursive(7)) # Output: 13
```

In [ ]:

```
27. Question: Write a function to flatten a nested list of integers.
    Assume each element is either an integer or a list.
Answer:
def flatten(nested_list):
flat_list = []
for item in nested_list:
if isinstance(item, list):
flat_list.extend(flatten(item))
else:
flat_list.append(item)
return flat_list
nested = [1, [2, 3, [4, 5], 6], 7, [8, 9]]
print(flatten(nested)) # Output: [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

In [ ]:

```
28. Question: Write a function to check if a given string is a palindrome.
Answer:
def is_palindrome(s):
return s == s[::-1]
string = "radar"
print(is_palindrome(string)) # Output: True
```

In [ ]:

```
29. Question: Given a string containing just the characters '(', ')', '{', '}',
'[' and ']', determine if the input string is valid. An input string is valid if:
```

```
Open brackets are closed by the same type of brackets.
Open brackets are closed in the correct order.
Answer:
    def is_valid_brackets(s):
stack = []
mapping = {")": "(", "}": "{", "]": "["}
for char in s:
if char in mapping:
top_element = stack.pop() if stack else '#'
if mapping[char] != top_element:
return False
else:
stack.append(char)
return not stack
brackets = "{[]}"
print(is_valid_brackets(brackets)) # Output: True
```

In [ ]:

```
30. Question: Write a function to find the two numbers in a list that sum up to a
specific target.
Answer:
def two_sum(nums, target):
num_dict = {}
for i, num in enumerate(nums):
complement = target - num
if complement in num_dict:
return [num_dict[complement], i]
num_dict[num] = i
return None
numbers = [2, 7, 11, 15]
target_value = 9
print(two_sum(numbers, target_value)) # Output: [0, 1]
```

In [ ]:

```
31. Question: Write a function that reverses a string, but maintains the
position of all non-alphabetic characters.
Answer:
def reverse_alphabet_only(s):
s = list(s)
i, j = 0, len(s) - 1
while i < j:
if not s[i].isalpha():
i += 1
elif not s[j].isalpha():
j -= 1
else:
s[i], s[j] = s[j], s[i]
i += 1
j -= 1
return ''.join(s)
string = "ab@cd#ef$gh"
print(reverse_alphabet_only(string)) # Output: "hg@fe#dc$ba"
```

In [ ]:

```
32. Question: Write a function to find the first non-repeated character in a string
Answer:
def first_unique_char(s):
char_count = {}
```

```
for char in s:
char_count[char] = char_count.get(char, 0) + 1
for char in s:
if char_count[char] == 1:
return char
return None
string = "swiss"
print(first_unique_char(string)) # Output: "w"
```

In [ ]:

```
33. Question: Write a function to find all the prime numbers less than a
given number n.
Answer:
def find_primes(n):
if n <= 2:
return []
primes = [True] * n
primes[0], primes[1] = False, False
for i in range(2, int(n ** 0.5) + 1):
if primes[i]:
for j in range(i * i, n, i):
primes[j] = False
return [i for i, val in enumerate(primes) if val]
number = 30
print(find_primes(number)) # Output: [2, 3, 5, 7, 11, 13, 17, 19, 23, 29]
```

In [ ]:

```
34. Question: Given two strings s and t, write a function to check
if t is an anagram of s.
Answer:
def is_anagram(s, t):
return sorted(s) == sorted(t)
s1 = "listen"
t1 = "silent"
print(is_anagram(s1, t1)) # Output: True
```

In [ ]:

```
35. Question: Write a function to compute the factorial of a number using iteration
Answer:
def factorial_iterative(n):
result = 1
for i in range(2, n+1):
result *= i
return result
number = 5
print(factorial_iterative(number)) # Output: 120
```

In [ ]:

```
36. Question: Write a function that checks if a given word is an isogram
(a word with no repeating letters).
Answer:
def is_isogram(word):
word = word.lower()
return len(word) == len(set(word))
word = "background"
print(is_isogram(word)) # Output: True
```

In [ ]:

```
37. Question: Write a function to rotate an array to the right by k steps,
where k is non-negative.
```

```
Answer:
def rotate(nums, k):
k = k % len(nums) # in case k is larger than the length of nums
nums[:] = nums[-k:] + nums[:-k]
return nums
array = [1,2,3,4,5,6,7]
steps = 3
print(rotate(array, steps)) # Output: [5,6,7,1,2,3,4]
```

In [ ]:

```
38. Question: Write a function to convert a given integer to its Roman numeral
representation.
Answer:
def int_to_roman(num):
val = [
1000, 900, 500, 400,
100, 90, 50, 40,
10, 9, 5, 4, 1
]
syms = [
"M", "CM", "D", "CD",
"C", "XC", "L", "XL",
"X", "IX", "V", "IV",
"I"
]
roman_num = ''
i = 0
while num > 0:
for _ in range(num // val[i]):
roman_num += syms[i]
num -= val[i]
i += 1
return roman_num
number = 3549
print(int_to_roman(number)) # Output: "MMMDXLIX"
```

In [ ]:

```
39. Question: Write a function that finds the longest common subsequence (LCS) of
two strings.
Answer:
def lcs(X, Y):
m = len(X)
n = len(Y)
dp = [[None] * (n + 1) for i in range(m + 1)]
for i in range(m + 1):
for j in range(n + 1):
if i == 0 or j == 0:
dp[i][j] = 0
elif X[i-1] == Y[j-1]:
dp[i][j] = dp[i-1][j-1] + 1
else:
dp[i][j] = max(dp[i-1][j], dp[i][j-1])
return dp[m][n]
str1 = "ABCBDAB"
str2 = "BDCAB"
print(lcs(str1, str2)) # Output: 4 (because "BCAB" is a common subsequence)
```

In [ ]:

```
40. Question: Write a function to find the square root of a number using the
Newton-Raphson method.
Answer:
def sqrt_newton(n, tolerance=1e-10, guess=1.0):
while True:
better_guess = (guess + n / guess) / 2
if abs(better_guess - guess) < tolerance: # Close enough
return better_guess
guess = better_guess
number = 25
print(sqrt_newton(number)) # Output: 5.0 (or very close to it)
```

In [ ]:

```
41. Question: Write a function that detects a cycle in a linked list.
Answer:
class ListNode:
def __init__(self, value=0, next=None):
self.value = value
self.next = next
def has_cycle(head):
slow, fast = head, head
while fast and fast.next:
slow = slow.next
fast = fast.next.next
if slow == fast:
return True
return False
# Example Usage:
# node1 = ListNode(1)
# node2 = ListNode(2)
# node3 = ListNode(3)
# node1.next = node2
# node2.next = node3
# node3.next = node1 # Creates a cycle
# print(has_cycle(node1)) # Output: True
```

In [ ]:

```
42. Question: Write a function that finds the intersection point of two linked list
Answer:
def get_intersection_node(headA, headB):
if not headA or not headB:
return None
ptrA, ptrB = headA, headB
while ptrA != ptrB:
ptrA = ptrA.next if ptrA else headB
ptrB = ptrB.next if ptrB else headA
return ptrA
# Assuming ListNode class definition from the previous question
# Example Usage:
# A: 1 -> 2 -> 3 -> 4
# â†˜
# 5 -> 6 -> 7
# â†—
# B: 8 -> 9
# print(get_intersection_node(A, B).value) # Output: 5
```

In [ ]:

```
43. Question: Write a function that computes the power of a number without
using the built-in power function or the ** operator.
Answer:
def power(base, exp):
if exp == 0:
return 1
if exp < 0:
base = 1 / base
exp = -exp
result = 1
current_product = base
while exp > 0:
if exp % 2 == 1:
result = result * current_product
current_product = current_product * current_product
exp //= 2
return result
print(power(2, 3)) # Output: 8
print(power(3, -2)) # Output: 0.1111 (or close to it)
```

In [ ]:

```
44. Question: Write a function to validate if a given string contains only
balanced parentheses. (Only '(' and ')' are considered).
Answer:
def is_balanced(s):
stack = []
for char in s:
if char == '(':
stack.append(char)
elif char == ')':
if not stack:
return False
stack.pop()
return len(stack) == 0
print(is_balanced("(())")) # Output: True
print(is_balanced("()()")) # Output: True
print(is_balanced("(()")) # Output: False
print(is_balanced(")(")) # Output: False
```

In [ ]:

```
45. Question: Write a function that returns the longest
substring without repeating characters.
Answer:
def length_of_longest_substring(s):
n = len(s)
ans = 0
char_index = {} # Current index of character
i = 0 # The sliding window left pointer
for j in range(n):
if s[j] in char_index:
i = max(char_index[s[j]], i)
ans = max(ans, j - i + 1)
char_index[s[j]] = j + 1
return ans
print(length_of_longest_substring("abcabcbb"))
# Output: 3 (because "abc" is the longest substring without repeating characters)
```

In [ ]:

```
46. Question: Given a string s and a string t,
find all the start indices of t's anagrams in s. Strings consist of lowercase
English letters only and the length of both strings s and
t will not be larger than 20,000.
Answer:
from collections import Counter
def find_anagrams(s, t):
t_counter = Counter(t)
s_counter = Counter(s[:len(t)-1])
res = []
for i in range(len(t)-1, len(s)):
s_counter[s[i]] += 1 # include a new char in the window
if s_counter == t_counter: # This step is O(1), as there are at most 26 English let
res.append(i-len(t)+1) # append the starting index
s_counter[s[i-len(t)+1]] -= 1 # decrease the count of oldest char in the window
if s_counter[s[i-len(t)+1]] == 0:
del s_counter[s[i-len(t)+1]] # remove the count if it is 0
return res
s = "cbaebabacd"
t = "abc"
print(find_anagrams(s, t)) # Output: [0, 6]
```

In [ ]:

```
47. Question: Given an unsorted integer array, find the smallest missing
positive integer.
Answer:
def first_missing_positive(nums):
n = len(nums)
# First, mark all negative values as 'n + 1'
for i in range(n):
if nums[i] <= 0:
nums[i] = n + 1
# Place each number in its correct position
for num in nums:
if 1 <= num <= n:
nums[num-1], num = num, nums[num-1]
# The first place where its number is not correct
for i, num in enumerate(nums, 1):
if num != i:
return i
return n + 1
nums = [3, 4, -1, 1]
print(first_missing_positive(nums)) # Output: 2
```

In [ ]:

```
48. Question: Given a set of non-overlapping intervals, insert a new interval
into the intervals (merge if necessary). You may assume that the intervals were
initially sorted according to their start times.
    def insert_interval(intervals, new_interval):
merged = []
i, n = 0, len(intervals)
# Add all the intervals starting before new_interval
while i < n and intervals[i][1] < new_interval[0]:
merged.append(intervals[i])
i += 1
# Merge all overlapping intervals to one considering new_interval
while i < n and intervals[i][0] <= new_interval[1]:
```

```
new_interval[0] = min(new_interval[0], intervals[i][0])
new_interval[1] = max(new_interval[1], intervals[i][1])
i += 1
# Add the union of intervals we got
merged.append(new_interval)
# Add all the rest
while i < n:
merged.append(intervals[i])
i += 1
return merged
intervals = [[1, 3], [6, 9]]
new_interval = [2, 5]
print(insert_interval(intervals, new_interval)) # Output: [[1, 5], [6, 9]]
```

In [ ]:

```
49. Question: Implement a basic calculator to evaluate a simple expression
string containing non-negative integers, '+', '-', '*', and '/' operators.
    You can assume the given expression is always valid.
Answer:
def calculate(s):
if not s:
return 0
stack, num, sign = [], 0, "+"
for i in range(len(s)):
if s[i].isdigit():
num = num * 10 + int(s[i])
if s[i] in "+-*/" or i == len(s) - 1:
if sign == "+":
stack.append(num)
elif sign == "-":
stack.append(-num)
elif sign == "*":
stack.append(stack.pop() * num)
else: # division
top = stack.pop()
if top < 0:
stack.append(-(-top // num))
else:
stack.append(top // num)
num = 0
sign = s[i]
return sum(stack)
expression = "3+2*2"
print(calculate(expression)) # Output: 7
```

In [ ]:

```
50. Question: Design a data structure that supports the following two operations:
void addWord(word)
bool search(word)
The search function should be able to search a literal word or a regular expression
string containing only letters a-z or .. The . period should be able to represent
any one letter.
    class WordDictionary:
def __init__(self):
self.trie = {}
def addWord(self, word):
node = self.trie
```

```
for w in word:
if w not in node:
node[w] = {}
node = node[w]
node['$'] = True
def search(self, word):
def search_in_node(word, node):
for i, ch in enumerate(word):
if not ch in node:
# If the current character is '.', check all possible nodes at this level
if ch == '.':
for x in node:
if x != '$' and search_in_node(word[i + 1:], node[x]):
return True
# if no nodes lead to answer, or the current character != '.'
return False
# if the character is found, go down to the next level in trie
node = node[ch]
return '$' in node
return search_in_node(word, self.trie)
# Example Usage:
# dictionary = WordDictionary()
# dictionary.addWord("bad")
# dictionary.addWord("dad")
# dictionary.addWord("mad")
# print(dictionary.search("pad")) # Output: False
# print(dictionary.search("bad")) # Output: True
# print(dictionary.search(".ad")) # Output: True
# print(dictionary.search("b..")) # Output: True
```

In [ ]:

```
51. Question: Given a list of words, group the anagrams together.
Answer:
from collections import defaultdict
def group_anagrams(words):
anagrams = defaultdict(list)
for word in words:
# Use sorted word as a key. All anagrams will result in the same key.
sorted_word = ''.join(sorted(word))
anagrams[sorted_word].append(word)
return list(anagrams.values())
words = ["eat", "tea", "tan", "ant", "bat"]
print(group_anagrams(words)) # Output: [['eat', 'tea'], ['tan', 'ant'], ['bat']]
Time Complexity:
Sorting each word takes O(KlogK) where K is the maximum length of a word.
Doing this for all words takes O(NKlogK) where N is the number of words.
```

In [ ]:

```
52. Question: Given an array nums and a target value, find the two numbers in the
array that sum up to the target value.
Answer:
def two_sum(nums, target):
num_to_index = {}
for i, num in enumerate(nums):
if target - num in num_to_index:
return [num_to_index[target - num], i]
num_to_index[num] = i
```

```
nums = [2, 7, 11, 15]
target = 9
print(two_sum(nums, target)) # Output: [0, 1]
Time Complexity:
O(N) where N is the number of elements in the array.
```

In [ ]:

```
53. Question: Find the longest palindromic substring in a string.
Answer:
def longest_palindrome(s):
if not s:
return ""
longest = ""
for i in range(len(s)):
# Odd length palindromes
p1 = expand_from_center(s, i, i)
if len(p1) > len(longest):
longest = p1
# Even length palindromes
p2 = expand_from_center(s, i, i + 1)
if len(p2) > len(longest):
longest = p2
return longest
def expand_from_center(s, l, r):
while l >= 0 and r < len(s) and s[l] == s[r]:
l -= 1
r += 1
return s[l + 1:r]
s = "babad"
print(longest_palindrome(s)) # Output: "bab" or "aba"
Time Complexity:
O(N^2) where N is the length of the string.
```

In [ ]:

```
54. Question: Implement a function to serialize and deserialize a binary tree.
Answer:
class TreeNode:
def __init__(self, val=0, left=None, right=None):
self.val = val
self.left = left
self.right = right
def serialize(root):
def helper(node):
if not node:
return ["null"]
return [str(node.val)] + helper(node.left) + helper(node.right)
return ','.join(helper(root))
def deserialize(data):
def helper(nodes):
val = nodes.pop(0)
if val == "null":
return None
node = TreeNode(int(val))
node.left = helper(nodes)
node.right = helper(nodes)
return node
nodes = data.split(',')
```

```
return helper(nodes)
# Usage:
# node = TreeNode(1, TreeNode(2), TreeNode(3, TreeNode(4), TreeNode(5)))
# s = serialize(node)
# print(s) # Output: "1,2,null,null,3,4,null,null,5,null,null"
# new_node = deserialize(s)
```

In [ ]:

```
55. Question: Determine if a given binary tree is a valid binary search tree.
    def is_valid_bst(root):
def helper(node, lower=float('-inf'), upper=float('inf')):
if not node:
return True
val = node.val
if val <= lower or val >= upper:
return False
if not helper(node.right, val, upper):
return False
if not helper(node.left, lower, val):
return False
return True
return helper(root)
# Assuming TreeNode class definition from the previous question
# Example Usage:
# node = TreeNode(2, TreeNode(1), TreeNode(3))
# print(is_valid_bst(node)) # Output: True
```

In [ ]:

```
56. Question: Given an array of integers, find out whether there are two distinct
indices i and j in the array such that the absolute difference between nums[i]
and nums[j] is at most t and the absolute difference between i and j is at most k.
Answer:
from sortedcontainers import SortedList
def contains_nearby_almost_duplicate(nums, k, t):
if t < 0: return False
slist, n = SortedList(), len(nums)
for i in range(n):
if i > k: slist.remove(nums[i - k - 1])
pos1 = slist.bisect_left(nums[i] - t)
pos2 = slist.bisect_right(nums[i] + t)
if pos1 != pos2:
return True
slist.add(nums[i])
return False
nums = [1, 2, 3, 1]
k = 3
t = 0
print(contains_nearby_almost_duplicate(nums, k, t)) # Output: True
```

In [ ]:

```
57. Question: Find the kth largest element in an unsorted array.
Answer:
import heapq
def find_kth_largest(nums, k):
return heapq.nlargest(k, nums)[-1]
nums = [3, 2, 1, 5, 6, 4]
k = 2
print(find_kth_largest(nums, k)) # Output: 5
```

In [ ]:

```
58. Question: Given a non-empty string s and a dictionary wordDict containing a
list of non-empty words, determine if s can be segmented into a space-separated
sequence of
one or more dictionary words.
Answer:
def word_break(s, wordDict):
wordSet, n = set(wordDict), len(s)
dp = [False] * (n + 1)
dp[0] = True
for i in range(1, n + 1):
for j in range(i):
if dp[j] and s[j:i] in wordSet:
dp[i] = True
break
return dp[-1]
s = "leetcode"
wordDict = ["leet", "code"]
print(word_break(s, wordDict)) # Output: True
```

In [ ]:

```
59. Question: Given a sorted array and a target value, return the index
if the target is found. If not, return the index where it would be if it were
inserted in order.
Answer:
def search_insert(nums, target):
left, right = 0, len(nums) - 1
while left <= right:
mid = (left + right) // 2
if nums[mid] == target:
return mid
elif nums[mid] < target:
left = mid + 1
else:
right = mid - 1
return left
nums = [1, 3, 5, 6]
target = 5
print(search_insert(nums, target)) # Output: 2
```

In [ ]:

```
60. Question: Rotate an array to the right by k steps.
Answer:
def rotate(nums, k):
n = len(nums)
k %= n
nums[:] = nums[-k:] + nums[:-k]
nums = [1, 2, 3, 4, 5, 6, 7]
```

```
k = 3
rotate(nums, k)
print(nums) # Output: [5, 6, 7, 1, 2, 3, 4]
```

In [ ]:

```
61. Question: Given an array of integers, every element appears
twice except for one. Find that single one.
Answer:
def singleNumber(nums):
result = 0
for num in nums:
result ^= num
return result
nums = [4, 1, 2, 1, 2]
print(singleNumber(nums)) # Output: 4
```

In [ ]:

```
62. Question: Write a function to determine the number of bits you would need
to flip to convert integer A to integer B.
Answer:
def bitSwapRequired(A, B):
count = 0
c = A ^ B # c will have 1s wherever A and B are different
while c:
count += c &amp; 1
c >>= 1
return count
A = 29 # 11101
B = 15 # 01111
print(bitSwapRequired(A, B)) # Output: 2
```

In [ ]:

```
63. Question: Given two strings, write a method to decide if one is a
permutation of the other.
Answer:
from collections import Counter
def is_permutation(str1, str2):
return Counter(str1) == Counter(str2)
str1 = "listen"
str2 = "silent"
print(is_permutation(str1, str2)) # Output: True
```

In [ ]:

```
64. Question: You are given an n x n 2D matrix representing an image.
    Rotate the image by 90 degrees (clockwise).
Answer:
def rotate(matrix):
n = len(matrix)
# Transpose the matrix
for i in range(n):
for j in range(i, n):
matrix[i][j], matrix[j][i] = matrix[j][i], matrix[i][j]
# Reverse the columns
for row in matrix:
row.reverse()
matrix = [
[1, 2, 3],
[4, 5, 6],
[7, 8, 9]
]
rotate(matrix)
print(matrix) # Output: [[7, 4, 1], [8, 5, 2], [9, 6, 3]]
```

In [ ]:

```
65. Question: Given a string containing just the characters
'(', ')', '{', '}', '[' and ']', determine if the input string is valid.
    An input string is valid if the brackets are closed in the
correct order.
Answer:
    def isValid(s):
stack = []
mapping = {")": "(", "}": "{", "]": "["}
for char in s:
if char in mapping:
top_element = stack.pop() if stack else '#'
if mapping[char] != top_element:
return False
else:
stack.append(char)
return not stack
s = "{[]}"
print(isValid(s)) # Output: True
```

In [ ]:

```
66. Question: Write a function to detect a cycle in a linked list.
Answer:
class ListNode:
def __init__(self, x):
self.val = x
self.next = None
def hasCycle(head):
slow, fast = head, head
while fast and fast.next:
slow = slow.next
fast = fast.next.next
if slow == fast:
return True
return False
```

In [ ]:

```
67. Question: Given a sorted linked list, delete all duplicates such that each
element appears only once.
Answer:
```

In [ ]:

```
class ListNode:
def __init__(self, x):
self.val = x
self.next = None
def deleteDuplicates(head):
current = head
while current and current.next:
if current.next.val == current.val:
current.next = current.next.next
else:
current = current.next
return head
```

In [ ]:

```
68. Question: Implement a basic calculator to evaluate a simple expression
string containing non-negative integers, '+', '-', '*', and '/' operators.
    Assume the expression is always
valid.
Answer:
def calculate(s):
stack, num, sign = [], 0, '+'
for i, c in enumerate(s):
if c.isdigit():
num = num * 10 + int(c)
if c in "+-*/" or i == len(s) - 1:
if sign == '+':
stack.append(num)
elif sign == '-':
stack.append(-num)
elif sign == '*':
stack[-1] *= num
elif sign == '/':
stack[-1] = int(stack[-1] / num)
num, sign = 0, c
return sum(stack)
s = "3+2*2"
print(calculate(s)) # Output: 7
```

In [ ]:

```
69. Question: Design and implement a TwoSum class. It should support the following
operations: add and find.
add: Add the number to an internal data structure.
find: Find if there exists any pair of numbers which sum is equal to the value.
Answer:
class TwoSum:
def __init__(self):
self.data = {}
def add(self, number):
if number in self.data:
self.data[number] += 1
else:
self.data[number] = 1
def find(self, value):
for num in self.data:
complement = value - num
if complement in self.data:
if complement != num or self.data[num] > 1:
return True
return False
```

In [ ]:

```
70. Question: Write a function to flatten a nested dictionary.
    Namespace the keys with a period.
Answer:
def flatten_dictionary(d, parent_key='', sep='.'):
items = {}
for k, v in d.items():
new_key = f"{parent_key}{sep}{k}" if parent_key else k
if isinstance(v, dict):
items.update(flatten_dictionary(v, new_key, sep=sep))
else:
```

```
items[new_key] = v
return items
nested_dict = {
"a": 1,
"b": {
"c": 2,
"d": {
"e": 3
}
}
}
print(flatten_dictionary(nested_dict)) # Output: {'a': 1, 'b.c': 2, 'b.d.e': 3}
```

In [ ]:

```
71. Question: Find the longest substring without repeating characters.
Answer:
    def length_of_longest_substring(s):
n = len(s)
set_ = set()
ans = 0
i, j = 0, 0
while i < n and j < n:
if s[j] not in set_:
set_.add(s[j])
j += 1
ans = max(ans, j - i)
else:
set_.remove(s[i])
i += 1
return ans
s = "abcabcbb"
print(length_of_longest_substring(s)) # Output: 3
```

In [ ]:

```
72. Question: Serialize and deserialize a binary tree.
Answer:
class TreeNode:
def __init__(self, x):
self.val = x
self.left = None
self.right = None
class Codec:
def serialize(self, root):
if not root:
return 'None'
return str(root.val) + ',' + self.serialize(root.left) + ','
+ self.serialize(root.right)
def deserialize(self, data):
def helper(data_list):
if data_list[0] == 'None':
data_list.pop(0)
return None
root = TreeNode(data_list[0])
data_list.pop(0)
root.left = helper(data_list)
root.right = helper(data_list)
return root
```

```
data_list = data.split(',')
return helper(data_list)
```

In [ ]:

```
73. Question: Write a function to match string s against pattern p, where p can
have characters and also . which matches any character, and * which matches zero
or more of the preceding element.
    def is_match(s, p):
if not p:
return not s
first_match = bool(s) and p[0] in {s[0], '.'}
if len(p) >= 2 and p[1] == '*':
return (is_match(s, p[2:]) or
first_match and is_match(s[1:], p))
else:
return first_match and is_match(s[1:], p[1:])
s = "mississippi"
p = "mis*is*p*."
print(is_match(s, p)) # Output: False
```

In [ ]:

```
74. Question: Find the peak element in an array. A peak element is an element
which is greater than or equal to its neighbors. Assume the array is sorted in
ascending order, and then a peak is found, then it is sorted in descending order.
    Also, assume the array may have duplicates.
Answer:
def find_peak_element(nums):
l, r = 0, len(nums) - 1
while l < r:
mid = (l + r) // 2
if nums[mid] < nums[mid + 1]:
l = mid + 1
else:
r = mid
return l
nums = [1, 2, 3, 4, 5, 6, 7, 5, 4, 3, 2]
print(find_peak_element(nums)) # Output: 6
```

In [ ]:

```
75. Question: Implement the strStr() function. Return the index of the first
occurrence of needle in haystack, or -1 if needle is not part of haystack.
Answer:
def strStr(haystack, needle):
if not needle:
return 0
needle_length = len(needle)
for i in range(len(haystack) - needle_length + 1):
if haystack[i:i + needle_length] == needle:
return i
return -1
haystack = "hello"
needle = "ll"
print(strStr(haystack, needle)) # Output: 2
```

In [ ]:

```
76. Question: Find the shortest path in a binary matrix from the top-left corner
to the bottom-right corner. You can move up, down, left, right, and diagonally if
the adjacent cells contain a 0. The path should avoid cells with a 1.
Answer:
```

```
from collections import deque
def shortest_path_binary_matrix(grid):
if not grid or not grid[0] or grid[0][0] or grid[-1][-1]:
return -1
n, m = len(grid), len(grid[0])
directions = [(0, 1), (1, 0), (1, 1), (-1, -1), (0, -1), (-1, 0), (1, -1), (-1, 1)]
queue = deque([(0, 0, 1)])
while queue:
x, y, dist = queue.popleft()
if x == n - 1 and y == m - 1:
return dist
for dx, dy in directions:
nx, ny = x + dx, y + dy
if 0 <= nx < n and 0 <= ny < m and not grid[nx][ny]:
grid[nx][ny] = 1
queue.append((nx, ny, dist + 1))
return -1
grid = [[0,0,0],[1,1,0],[1,1,0]]
print(shortest_path_binary_matrix(grid)) # Output: 4
```

In [ ]:

```
77. Question: Design a data structure that supports the following two operations:
void addWord(word) and bool search(word). The search method can search a literal
word or a regular expression string containing only letters a-z or .. A . means
it can represent any one-letter.
    class TrieNode:
def __init__(self):
self.children = {}
self.is_end = False
class WordDictionary:
def __init__(self):
self.root = TrieNode()
def addWord(self, word):
node = self.root
for ch in word:
if ch not in node.children:
node.children[ch] = TrieNode()
node = node.children[ch]
node.is_end = True
def search(self, word):
return self.match(word, 0, self.root)
def match(self, word, index, node):
if index == len(word):
return node.is_end
if word[index] != '.':
return word[index] in node.children and self.match(word, index + 1,
                                                   node.children[word[index]])
for child in node.children.values():
if self.match(word, index + 1, child):
return True
return False
wd = WordDictionary()
wd.addWord("bad")
wd.addWord("dad")
wd.addWord("mad")
print(wd.search("pad")) # Output: False
print(wd.search("bad")) # Output: True
print(wd.search(".ad")) # Output: True
```

In [ ]:

```
78. Question: Find the kth largest element in an unsorted array.
Answer:
def findKthLargest(nums, k):
import heapq
return heapq.nlargest(k, nums)[-1]
nums = [3,2,3,1,2,4,5,5,6]
k = 4
print(findKthLargest(nums, k)) # Output: 4
```

In [ ]:

```
79. Question: Given a list of integers, return the number of good pairs.
    A pair (i, j) is called good if nums[i] == nums[j] and i < j.
Answer:
def numIdenticalPairs(nums):
from collections import Counter
count = Counter(nums)
return sum(v*(v-1)//2 for v in count.values())
nums = [1,2,3,1,1,3]
print(numIdenticalPairs(nums)) # Output: 4
```

In [ ]:

```
80. Question: Find if a given string can be formed by a sequence of one or
more palindrome strings.
Answer:
def can_form_palindrome(s):
from collections import Counter
count = Counter(s)
return sum(v % 2 for v in count.values()) <= 1
s = "aabb"
print(can_form_palindrome(s)) # Output: True
```

In [ ]:

```
5. Linked List Cycle Detection
Question: Detect if there is a cycle in a linked list.
Answer:
class ListNode:
def __init__(self, value=0, next=None):
self.value = value
self.next = next
def has_cycle(node):
slow, fast = node, node
while fast and fast.next:
slow = slow.next
fast = fast.next.next
if slow == fast:
return True
return False
```

In [ ]:

```
6. Merge Two Sorted Lists
Question: Merge two sorted linked lists.
Answer:
def merge_sorted_lists(l1, l2):
dummy = ListNode(0)
current = dummy
while l1 and l2:
```

```
if l1.value < l2.value:
current.next, l1 = l1, l1.next
else:
current.next, l2 = l2, l2.next
current = current.next
current.next = l1 or l2
return dummy.next
```

In [ ]:

```
7. Find the Middle of Linked List
Question: Find the middle element of a linked list.
Answer:
def find_middle(node):
slow, fast = node, node
while fast and fast.next:
slow = slow.next
fast = fast.next.next
return slow
```

In [ ]:

```
8. Maximum Subarray Sum
Question: Find the maximum subarray sum using Kadane's algorithm.
Answer:
def max_subarray(nums):
max_current = max_global = nums[0]
for i in range(1, len(nums)):
max_current = max(nums[i], max_current + nums[i])
max_global = max(max_global, max_current)
return max_global
```

In [ ]:

```
9. Check if a Tree is Balanced
Question: Check if a binary tree is balanced.
Answer:
class TreeNode:
def __init__(self, value=0, left=None, right=None):
self.value = value
self.left = left
self.right = right
def is_balanced(root):
def check_balance(node):
if not node:
return 0, True
left_height, left_balanced = check_balance(node.left)
right_height, right_balanced = check_balance(node.right)
return max(left_height, right_height) + 1, left_balanced and right_balanced
                and abs(left_height - right_height) <= 1
return check_balance(root)[1]
```

In [ ]:

```
10. Breadth-first Search (BFS) in Graph
Question: Implement BFS for a graph.
Answer:
from collections import deque
def bfs(graph, start):
visited = set()
queue = deque([start])
while queue:
vertex = queue.popleft()
```

```
if vertex not in visited:
visited.add(vertex)
queue.extend(graph[vertex] - visited)
return visited
```

In [ ]:

```
11. Depth-first Search (DFS) in Graph
Question: Implement DFS for a graph.
    def dfs(graph, start, visited=None):
if visited is None:
visited = set()
visited.add(start)
for vertex in graph[start] - visited:
dfs(graph, vertex, visited)
return visited
```

In [ ]:

```
12. Implement a Stack
Question: Implement a stack using linked list.
Answer:
class StackNode:
def __init__(self, value=0, next=None):
self.value = value
self.next = next
class Stack:
def __init__(self):
self.top = None
def push(self, value):
self.top = StackNode(value, self.top)
def pop(self):
if not self.top:
return None
value = self.top.value
self.top = self.top.next
return value
def peek(self):
return None if not self.top else self.top.value
def is_empty(self):
return self.top is None
```

In [ ]:

```
13. Implement a Queue
Question: Implement a queue using two stacks.
Answer:
class Queue:
def __init__(self):
self.stack1 = []
self.stack2 = []
def enqueue(self, value):
self.stack1.append(value)
def dequeue(self):
if not self.stack2:
while self.stack1:
self.stack2.append(self.stack1.pop())
return self.stack2.pop() if self.stack2 else None
```

In [ ]:

```
Implement a Priority Queue
Question: Implement a priority queue using a heap.
```

```
Answer:
import heapq
class PriorityQueue:
def __init__(self):
self.queue = []
def enqueue(self, value, priority=0):
heapq.heappush(self.queue, (priority, value))
def dequeue(self):
return heapq.heappop(self.queue)[1] if self.queue else None
```

In [ ]:

```
15. Implement Hashmap
Question: Implement a simple hashmap.
Answer:
class Hashmap:
def __init__(self):
self.size = 1000
self.map = [None] * self.size
def _hash(self, key):
return hash(key) % self.size
def put(self, key, value):
key_hash = self._hash(key)
self.map[key_hash] = value
def get(self, key):
key_hash = self._hash(key)
return self.map[key_hash]
def remove(self, key):
key_hash = self._hash(key)
self.map[key_hash] = None
```

In [ ]:

```
16. Binary Search
Question: Implement binary search for a sorted list.
Answer:
def binary_search(arr, x):
l, r = 0, len(arr) - 1
while l <= r:
mid = (l + r) // 2
if arr[mid] == x:
return mid
elif arr[mid] < x:
l = mid + 1
else:
r = mid - 1
return -1
```

In [ ]:

```
17. Implement Trie (Prefix Tree)
Question: Implement a basic trie for word insert, search and prefix search.
Answer:
    class TrieNode:
def __init__(self):
self.children = {}
self.is_end_of_word = False
class Trie:
def __init__(self):
self.root = TrieNode()
def insert(self, word):
```

```
node = self.root
for char in word:
if char not in node.children:
node.children[char] = TrieNode()
node = node.children[char]
node.is_end_of_word = True
def search(self, word):
node = self.root
for char in word:
if char not in node.children:
return False
node = node.children[char]
return node.is_end_of_word
def starts_with(self, prefix):
node = self.root
for char in prefix:
if char not in node.children:
return False
node = node.children[char]
return True
```

In [ ]:

```
18. Find First and Last Position of Element in Sorted Array
Question: Given a sorted array of integers and a target value, find the starting an
Answer:
def search_range(nums, target):
def find_left_boundary(nums, target):
left, right = 0, len(nums) - 1
while left <= right:
mid = (left + right) // 2
if nums[mid] < target:
left = mid + 1
else:
right = mid - 1
return left
left, right = find_left_boundary(nums, target), find_left_boundary(nums, target + 1
if left <= right:
return [left, right]
return [-1, -1]
```

In [ ]:

```
19. Topological Sort
Question: Implement a topological sort for a directed graph.
Answer:
    from collections import defaultdict, deque
def topological_sort(vertices, edges):
graph = defaultdict(list)
in_degree = {v: 0 for v in vertices}
for u, v in edges:
graph[u].append(v)
in_degree[v] += 1
queue = deque([v for v, d in in_degree.items() if d == 0])
order = []
while queue:
vertex = queue.popleft()
order.append(vertex)
for neighbor in graph[vertex]:
```

```
in_degree[neighbor] -= 1
if in_degree[neighbor] == 0:
queue.append(neighbor)
return order if len(order) == len(vertices) else []
```

In [ ]:

```
20. Check if a String Contains All Binary Codes of Size K
Question: Given a binary string s and an integer k, check if all binary codes
of length k is a substring of s.
Answer:
def has_all_codes(s, k):
needed = 1 << k
seen = set()
for i in range(len(s) - k + 1):
substring = s[i:i+k]
if substring not in seen:
seen.add(substring)
needed -= 1
if needed == 0:
return True
return False
```

In [ ]:

```
21. Implement QuickSort
Question: Implement the quicksort algorithm.
Answer:
def quicksort(arr):
if len(arr) <= 1:
return arr
pivot = arr[len(arr) // 2]
left = [x for x in arr if x < pivot]
middle = [x for x in arr if x == pivot]
right = [x for x in arr if x > pivot]
return quicksort(left) + middle + quicksort(right)
```

In [ ]:

```
22. Implement MergeSort
Question: Implement the mergesort algorithm.
    def mergesort(arr):
if len(arr) <= 1:
return arr
mid = len(arr) // 2
left = arr[:mid]
right = arr[mid:]
return merge(mergesort(left), mergesort(right))
def merge(left, right):
result = []
i = j = 0
while i < len(left) and j < len(right):
if left[i] < right[j]:
result.append(left[i])
i += 1
else:
result.append(right[j])
j += 1
result.extend(left[i:])
result.extend(right[j:])
return result
```

In [ ]:

```
23. Maximum Depth of Binary Tree
Question: Find the maximum depth of a binary tree.
Answer:
def max_depth(root):
if not root:
return 0
left_depth = max_depth(root.left)
right_depth = max_depth(root.right)
return max(left_depth, right_depth) + 1
```

In [ ]:

```
24. Count the Number of Islands
Question: Given a 2D grid consisting of '1's (land) and '0's (water),
count the number of islands. An island is surrounded by water and is formed by
connecting adjacent lands horizontally or vertically.
Answer:
def num_islands(grid):
if not grid:
return 0
count = 0
for i in range(len(grid)):
for j in range(len(grid[0])):
if grid[i][j] == '1':
dfs(grid, i, j)
count += 1
return count
def dfs(grid, i, j):
if (i < 0 or i >= len(grid) or j < 0 or j >= len(grid[0]) or grid[i][j] != '1'):
return
grid[i][j] = '#'
dfs(grid, i-1, j)
dfs(grid, i+1, j)
dfs(grid, i, j-1)
dfs(grid, i, j+1)
```

In [ ]:

```
Coin Change
Question: You are given coins of different denominations and a total amount of
money amount. Write a function to compute the fewest number of coins that you
need to make up that amount. If that amount of money cannot be made up by any
combination of the coins, return -1.
Answer:
def coin_change(coins, amount):
dp = [float('inf')] * (amount + 1)
dp[0] = 0
for coin in coins:
for x in range(coin, amount + 1):
dp[x] = min(dp[x], dp[x - coin] + 1)
return dp[amount] if dp[amount] != float('inf') else -1
```

In [ ]:

```
Longest Increasing Subsequence
Question: Given an unsorted array of integers, find the length of longest increasin
Answer:
def length_of_lis(nums):
if not nums:
return 0
dp = [1] * len(nums)
for i in range(len(nums)):
for j in range(i):
if nums[i] > nums[j]:
dp[i] = max(dp[i], dp[j] + 1)
return max(dp)
```

In [ ]:

```
36. Intersection of Two Arrays II
Question: Given two arrays, write a function to compute their intersection.
    Each element in the result should appear as many times as it shows in both
arrays. The result can be in any order.
Answer:
from collections import Counter
def intersect(nums1, nums2):
c1, c2 = Counter(nums1), Counter(nums2)
return list((c1 &amp; c2).elements())
```

In [ ]:

```
37. Single Number
Question: Given a non-empty array of integers nums, every element appears twice
except for one. Find that single one.
Answer:
def single_number(nums):
res = 0
for num in nums:
res ^= num
return res
```

In [ ]:

```
38. Rotate Array
Question: Given an array, rotate the array to the right by k steps, where
k is non-negative.
Answer:
def rotate(nums, k):
```

```
k %= len(nums)
nums[:] = nums[-k:] + nums[:-k]
```