80 questions :

* 10 mixed
* Easy mistakes : 21 questions
* Medium mistakes : 33 questions
* Hard mistakes : 16 questions

Geeks for Geeks<https://www.geeksforgeeks.org/python-exercises-practice-questions-and-solutions/>

Github

[100+ Python challenging programming exercises.txt](https://github.com/zhiwehu/Python-programming-exercises/blob/master/100%2B%20Python%20challenging%20programming%20exercises.txt)

Toptal

<https://www.toptal.com/python/top-10-mistakes-that-python-programmers-make>

Medium  
<https://sheriffjbabu.medium.com/top-18-mistakes-of-python-developers-454c2ad4a08>

**CODE SNIPPETS/QUESTIONS -**

***Mixed Mistakes/Topics: (gpt/medium 10 questions)***

Certainly! Here is one debugging question per topic:

### 1. Misusing Expressions as Defaults for Function Arguments

1. \*\*Misusing Expressions as Defaults for Function Arguments\*\*:

- Aim: The aim of this code snippet is to correctly use a default mutable argument in a function.

- Wrong Code Snippet:

```python

def append\_item(item, my\_list=[]):

my\_list.append(item)

return my\_list

list1 = append\_item(1)

list2 = append\_item(2)

print(list1)

print(list2)

```

<!-- The default mutable argument 'my\_list' causes shared references across calls -->

- Corrected Code Snippet:

```python

def append\_item(item, my\_list=None):

if my\_list is None:

my\_list = []

my\_list.append(item)

return my\_list

list1 = append\_item(1)

list2 = append\_item(2)

print(list1)

print(list2)

```

### 2. Using Class Variables Incorrectly

2. \*\*Using Class Variables Incorrectly\*\*:

- Aim: The aim of this code snippet is to correctly use class and instance variables.

- Wrong Code Snippet:

```python

class Counter:

count = 0 # Class variable intended to store count

def increment(self):

self.count += 1

c1 = Counter()

c2 = Counter()

c1.increment()

c2.increment()

print(c1.count)

print(c2.count)

```

<!-- Class variable 'count' shared among all instances -->

- Corrected Code Snippet:

```python

class Counter:

def \_\_init\_\_(self):

self.count = 0 # Instance variable intended to store count

def increment(self):

self.count += 1

c1 = Counter()

c2 = Counter()

c1.increment()

c2.increment()

print(c1.count)

print(c2.count)

```

### 3. Specifying Parameters Incorrectly for an Exception Block

3. \*\*Specifying Parameters Incorrectly for an Exception Block\*\*:

- Aim: The aim of this code snippet is to correctly handle multiple exceptions with appropriate parameters.

- Wrong Code Snippet:

```python

try:

result = 10 / "0"

except (ZeroDivisionError, TypeError), e:

print("Error:", e)

```

<!-- Incorrect syntax for handling multiple exceptions -->

- Corrected Code Snippet:

```python

try:

result = 10 / "0"

except (ZeroDivisionError, TypeError) as e:

print("Error:", e)

```

### 4. Misunderstanding Python Scope Rules

4. \*\*Misunderstanding Python Scope Rules\*\*:

- Aim: The aim of this code snippet is to correctly modify a variable in an enclosing scope using nested functions.

- Wrong Code Snippet:

```python

def outer\_function():

x = "hello"

def inner\_function():

x = "world"

print(x)

inner\_function()

print(x)

outer\_function()

```

<!-- Inner function does not modify the variable in the outer scope -->

- Corrected Code Snippet:

```python

def outer\_function():

x = "hello"

def inner\_function():

nonlocal x

x = "world"

print(x)

inner\_function()

print(x)

outer\_function()

```

### 5. Modifying a List While Iterating Over It

5. \*\*Modifying a List While Iterating Over It\*\*:

- Aim: The aim of this code snippet is to correctly remove even numbers from a list while iterating over it.

- Wrong Code Snippet:

```python

nums = [1, 2, 3, 4, 5, 6]

for num in nums:

if num % 2 == 0:

nums.remove(num)

print(nums)

```

<!-- Modifying the list while iterating causes unexpected behavior -->

- Corrected Code Snippet:

```python

nums = [1, 2, 3, 4, 5, 6]

nums\_to\_remove = [num for num in nums if num % 2 == 0]

for num in nums\_to\_remove:

nums.remove(num)

print(nums)

```

### 6. Confusing How Python Binds Variables in Closures

6. \*\*Confusing How Python Binds Variables in Closures\*\*:

- Aim: The aim of this code snippet is to correctly capture the current value of the loop variable in a closure.

- Wrong Code Snippet:

```python

def create\_functions():

funcs = []

for i in range(3):

def func():

return i

funcs.append(func)

return funcs

for func in create\_functions():

print(func())

```

<!-- The loop variable 'i' is captured incorrectly, leading to all functions using the last value of 'i' -->

- Corrected Code Snippet:

```python

def create\_functions():

funcs = []

for i in range(3):

def func(i=i):

return i

funcs.append(func)

return funcs

for func in create\_functions():

print(func())

```

### 7. Creating Circular Module Dependencies

7. \*\*Creating Circular Module Dependencies\*\*:

- Aim: The aim of this code snippet is to correctly structure module imports to avoid circular dependencies.

- Wrong Code Snippet:

```python

# file module1.py

import module2

def function1():

print("Function 1")

module2.function2()

# file module2.py

import module1

def function2():

print("Function 2")

module1.function1()

```

<!-- Circular import causes an ImportError -->

- Corrected Code Snippet:

```python

# file module1.py

def function1():

print("Function 1")

# file module2.py

from module1 import function1

def function2():

print("Function 2")

function1()

function2()

```

### 8. Name Clashing with Python Standard Library Modules

8. \*\*Name Clashing with Python Standard Library Modules\*\*:

- Aim: The aim of this code snippet is to correctly use a custom module without clashing with a Python Standard Library module.

- Wrong Code Snippet:

```python

# file collections.py

def my\_function():

return "Custom collections function"

import collections

print(collections.my\_function())

```

<!-- The custom module 'collections' clashes with the Python Standard Library 'collections' module -->

- Corrected Code Snippet:

```python

# file my\_collections.py

def my\_function():

return "Custom collections function"

import my\_collections

print(my\_collections.my\_function())

```

### 9. Misusing the `\_\_del\_\_` Method

9. \*\*Misusing the `\_\_del\_\_` Method\*\*:

- Aim: The aim of this code snippet is to properly use the `\_\_del\_\_` method without causing issues.

- Wrong Code Snippet:

```python

class Resource:

def \_\_del\_\_(self):

print("Resource cleanup")

def create\_resource():

res = Resource()

print("Resource created")

create\_resource()

```

<!-- The \_\_del\_\_ method may not be called immediately due to delayed garbage collection -->

- Corrected Code Snippet:

```python

class Resource:

def cleanup(self):

print("Resource cleanup")

def create\_resource():

res = Resource()

print("Resource created")

res.cleanup()

create\_resource()

```

### 10. Writing Code That Only Works on One Platform

10. \*\*Writing Code That Only Works on One Platform\*\*:

- Aim: The aim of this code snippet is to write platform-independent code.

- Wrong Code Snippet:

```python

import os

def list\_files():

os.system('ls') # This works only on Unix-based systems

list\_files()

```

<!-- Using Unix-specific command 'ls' which won't work on Windows -->

- Corrected Code Snippet:

```python

import os

def list\_files():

for file in os.listdir('.'):

print(file)

list\_files()

```

***Easy Mistakes/Topics: (gpt 11+10 questions)***

***GPT QUESTIONS***

### Easy Mistakes/Topics:

1. \*\*Syntax Errors\*\*:

- Aim: The aim of this code snippet is to correctly use a for loop to print numbers from 0 to 4.

- Wrong Code Snippet:

```python

for i in range(5)

print(i)

```

<!-- Missing colon after 'range(5)' -->

- Corrected Code Snippet:

```python

for i in range(5):

print(i)

```

2. \*\*Variable Naming\*\*:

- Aim: The aim of this code snippet is to correctly use variable names in a print statement.

- Wrong Code Snippet:

```python

number = 10

print(Number)

```

<!-- Case-sensitive variable names, 'Number' should be 'number' -->

- Corrected Code Snippet:

```python

number = 10

print(number)

```

3. \*\*Import Errors\*\*:

- Aim: The aim of this code snippet is to correctly import the pandas module.

- Wrong Code Snippet:

```python

import panda as pd

```

<!-- Wrong module name, should be 'pandas' instead of 'panda' -->

- Corrected Code Snippet:

```python

import pandas as pd

```

4. \*\*Data Type Misuse\*\*:

- Aim: The aim of this code snippet is to correctly concatenate a string and an integer.

- Wrong Code Snippet:

```python

x = "5" + 6

```

<!-- Trying to concatenate string with integer -->

- Corrected Code Snippet:

```python

x = "5" + str(6)

```

5. \*\*Print Debugging\*\*:

- Aim: The aim of this code snippet is to use a print statement to debug and display the result of a multiplication.

- Wrong Code Snippet:

```python

result = 10 \* 2

```

<!-- Missing print statement -->

- Corrected Code Snippet:

```python

result = 10 \* 2

print("Result:", result)

```

6. \*\*String Manipulation Errors\*\*:

- Aim: The aim of this code snippet is to access the last character of a string.

- Wrong Code Snippet:

```python

message = "Hello"

print(message[5])

```

<!-- Trying to access index out of range -->

- Corrected Code Snippet:

```python

message = "Hello"

print(message[4]) # Accessing the last character 'o'

```

7. \*\*List Manipulation Errors\*\*:

- Aim: The aim of this code snippet is to append an element to a list.

- Wrong Code Snippet:

```python

numbers = [1, 2, 3, 4, 5]

numbers.push(6)

```

<!-- Using the incorrect method 'push' instead of 'append' -->

- Corrected Code Snippet:

```python

numbers = [1, 2, 3, 4, 5]

numbers.append(6)

```

8. \*\*Syntax Errors\*\*:

- Aim: The aim of this code snippet is to correctly create a dictionary and print its values.

- Wrong Code Snippet:

```python

my\_dict = { "name": "Alice", "age": 25, }

print(my\_dict("name"))

```

<!-- Using parentheses instead of square brackets to access a dictionary value -->

- Corrected Code Snippet:

```python

my\_dict = { "name": "Alice", "age": 25 }

print(my\_dict["name"])

```

9. \*\*Import Errors\*\*:

- Aim: The aim of this code snippet is to correctly import the `math` module and use the `sqrt` function.

- Wrong Code Snippet:

```python

import maths

print(maths.sqrt(16))

```

<!-- Incorrect module name, should be 'math' instead of 'maths' -->

- Corrected Code Snippet:

```python

import math

print(math.sqrt(16))

```

10. \*\*Data Type Misuse\*\*:

- Aim: The aim of this code snippet is to add two integers and print the result.

- Wrong Code Snippet:

```python

a = "10"

b = 20

result = a + b

print(result)

```

<!-- Adding a string to an integer, causing a type error -->

- Corrected Code Snippet:

```python

a = "10"

b = 20

result = int(a) + b

print(result)

```

11. \*\*Print Debugging\*\*:

- Aim: The aim of this code snippet is to use a print statement to debug and display a list's contents.

- Wrong Code Snippet:

```python

my\_list = [1, 2, 3, 4]

print("List content: " + my\_list)

```

<!-- Trying to concatenate a string and a list, causing a type error -->

- Corrected Code Snippet:

```python

my\_list = [1, 2, 3, 4]

print("List content:", my\_list)

Certainly! Here are debugging questions for each of the specified topics, following the same format:

### 1. Syntax Errors

1. \*\*Syntax Errors\*\*:

- Aim: The aim of this code snippet is to correctly use colons and parentheses in control structures.

- Wrong Code Snippet:

```python

if True

print("This is a syntax error"

for i in range(5)

print(i

```

<!-- Missing colons and mismatched parentheses -->

- Corrected Code Snippet:

```python

if True:

print("This is a syntax error")

for i in range(5):

print(i)

```

### 2. Variable Naming

2. \*\*Variable Naming\*\*:

- Aim: The aim of this code snippet is to avoid using reserved keywords as variable names.

- Wrong Code Snippet:

```python

def = 10

print(def)

```

<!-- Using 'def', a reserved keyword, as a variable name -->

- Corrected Code Snippet:

```python

value = 10

print(value)

```

### 3. Import Errors

3. \*\*Import Errors\*\*:

- Aim: The aim of this code snippet is to correctly import a module using the correct module name.

- Wrong Code Snippet:

```python

import numppy as np

arr = np.array([1, 2, 3])

print(arr)

```

<!-- Typo in the module name 'numppy' instead of 'numpy' -->

- Corrected Code Snippet:

```python

import numpy as np

arr = np.array([1, 2, 3])

print(arr)

```

### 4. Data Type Misuse

4. \*\*Data Type Misuse\*\*:

- Aim: The aim of this code snippet is to correctly handle data types during concatenation.

- Wrong Code Snippet:

```python

num = 10

result = "The number is: " + num

print(result)

```

<!-- Trying to concatenate a string and an integer -->

- Corrected Code Snippet:

```python

num = 10

result = "The number is: " + str(num)

print(result)

```

### 5. Print Debugging

5. \*\*Print Debugging\*\*:

- Aim: The aim of this code snippet is to use print statements effectively for debugging.

- Wrong Code Snippet:

```python

def add(a, b):

return a + b

result = add(5, 10)

print("The result is ")

```

<!-- Missing variable in the print statement for debugging -->

- Corrected Code Snippet:

```python

def add(a, b):

return a + b

result = add(5, 10)

print("The result is", result)

```

### 6. String Manipulation Errors

6. \*\*String Manipulation Errors\*\*:

- Aim: The aim of this code snippet is to correctly concatenate and slice strings.

- Wrong Code Snippet:

```python

s = "Hello"

s[0] = "h"

greeting = s + " World

print(greeting)

```

<!-- Incorrect string manipulation and missing quote -->

- Corrected Code Snippet:

```python

s = "Hello"

greeting = "h" + s[1:] + " World"

print(greeting)

```

### 7. List Manipulation Errors

7. \*\*List Manipulation Errors\*\*:

- Aim: The aim of this code snippet is to correctly use list methods like append and extend.

- Wrong Code Snippet:

```python

my\_list = [1, 2, 3]

my\_list.append([4, 5])

print(my\_list)

```

<!-- Using append instead of extend -->

- Corrected Code Snippet:

```python

my\_list = [1, 2, 3]

my\_list.extend([4, 5])

print(my\_list)

```

### 8. Basic Control Structures

8. \*\*Basic Control Structures\*\*:

- Aim: The aim of this code snippet is to correctly use if-else statements and loops.

- Wrong Code Snippet:

```python

for i in range(5):

if i % 2 == 0

print(f"{i} is even")

else

print(f"{i} is odd")

```

<!-- Missing colons in if-else statements -->

- Corrected Code Snippet:

```python

for i in range(5):

if i % 2 == 0:

print(f"{i} is even")

else:

print(f"{i} is odd")

```

### 9. Function Definition and Invocation

9. \*\*Function Definition and Invocation\*\*:

- Aim: The aim of this code snippet is to correctly define and call a function.

- Wrong Code Snippet:

```python

def greet(name)

return f"Hello, {name}"

print(greet("Alice"))

```

<!-- Missing colon in function definition -->

- Corrected Code Snippet:

```python

def greet(name):

return f"Hello, {name}"

print(greet("Alice"))

```

### 10. Basic File I/O

10. \*\*Basic File I/O\*\*:

- Aim: The aim of this code snippet is to correctly read from a file.

- Wrong Code Snippet:

```python

file = open("example.txt", "r")

content = file.read

file.close()

print(content)

```

<!-- Incorrect method call 'read' without parentheses -->

- Corrected Code Snippet:

```python

file = open("example.txt", "r")

content = file.read()

file.close()

print(content)

```

These questions test different aspects within each topic, providing a range of debugging challenges.

```

***Medium Mistakes/Topics: (2 from chatgpt + 3 from gemini + 9\*2 from chatgpt/toptal + 10gpt)***

Certainly! Here are the questions for the medium-level topics with relevant aims:

### Medium Mistakes/Topics:

1. \*\*Mutable vs. Immutable Objects\*\*:

- Aim: The aim of this code snippet is to properly modify a list inside a function and return it.

- Wrong Code Snippet:

```python

def modify\_list(lst):

lst = lst.append(4)

return lst

numbers = [1, 2, 3]

print(modify\_list(numbers))

```

<!-- Modifying the list in place and returning 'None' because 'append' returns None -->

- Corrected Code Snippet:

```python

def modify\_list(lst):

lst.append(4)

return lst

numbers = [1, 2, 3]

print(modify\_list(numbers))

```

2. \*\*Object-Oriented Programming (OOP) Concepts\*\*:

- Aim: The aim of this code snippet is to properly initialize a subclass with attributes from the superclass.

- Wrong Code Snippet:

```python

class Animal:

def \_\_init\_\_(self, name):

self.name = name

class Dog(Animal):

def \_\_init\_\_(self, name, breed):

self.breed = breed

dog = Dog("Buddy", "Golden Retriever")

print(dog.name, dog.breed)

```

<!-- Missing call to 'super().\_\_init\_\_' in Dog class -->

- Corrected Code Snippet:

```python

class Animal:

def \_\_init\_\_(self, name):

self.name = name

class Dog(Animal):

def \_\_init\_\_(self, name, breed):

super().\_\_init\_\_(name)

self.breed = breed

dog = Dog("Buddy", "Golden Retriever")

print(dog.name, dog.breed)

```

Certainly, here are some improved debugging questions that delve deeper into Python's intricacies and produce incorrect or non-compiling code when a bug is present:

**1. Looping and List Mutability (Tests understanding of list modification within a loop):**

**Buggy Code:**

Python



def remove\_evens(data):

for num in data:

if num % 2 == 0:

data.remove(num) # Bug: Modifying list during iteration

return data

numbers = [1, 2, 3, 4, 5]

cleaned\_data = remove\_evens(numbers.copy()) # Create a copy to avoid unintended modification

print(cleaned\_data)

**Explanation:** The bug lies in removing elements from the data list within the loop itself. This can lead to unexpected behavior because the loop iterator might skip elements due to the list size changing during iteration.

**Correct Solution:**

Python



def remove\_evens(data):

cleaned\_data = []

for num in data:

if num % 2 != 0:

cleaned\_data.append(num) # Append to a new list

return cleaned\_data

numbers = [1, 2, 3, 4, 5]

cleaned\_data = remove\_evens(numbers)

print(cleaned\_data) # Output: [1, 3, 5]



**2. Recursion and Base Case Handling (Tests understanding of recursive function structure):**

**Buggy Code (Missing Base Case):**

Python



def factorial(num):

if num == 0:

return 1

else:

return num \* factorial(num) # Infinite recursion

result = factorial(5)

print(result)

**Explanation:** The bug is the recursion continues infinitely, leading to a runtime error.

**Correct Solution:**

Python



def factorial(num):

if num == 0:

return 1

else:

return num \* factorial(num - 1)

result = factorial(5)

print(result) # Output: 120

**3. Closures and Scope (Tests understanding of variable scope within functions):**

**Buggy Code (Incorrect Variable Capture):**

Python



def create\_multiplier(factor):

def multiplier(num):

return factor \* num # Captures 'factor' incorrectly

return multiplier

multiplier2 = create\_multiplier(2)

multiplier3 = create\_multiplier(3)

print(multiplier2(5)) # Output: 6 (should be 10)

print(multiplier3(5)) # Output: 6 (should be 15)

**Explanation:** The bug is in the create\_multiplier function. The inner function multiplier captures the value of factor at the time of creation, not when it's called. This leads to both multiplier2 and multiplier3 using the same value of factor (2 in this case).

**Correct Solution (Using Nonlocal Keyword):**

Python



def create\_multiplier(factor):

def multiplier(num):

nonlocal factor # Declare nonlocal to modify 'factor' within the inner function

factor \*= 2 # Double the factor value for each call

return factor \* num

return multiplier

multiplier2 = create\_multiplier(2)

multiplier3 = create\_multiplier(3)

print(multiplier2(

SET1

### Misusing expressions as defaults for function arguments

1. \*\*Misusing Expressions as Defaults for Function Arguments\*\*:

- Aim: The aim of this code snippet is to correctly use a default mutable argument in a function.

- Wrong Code Snippet:

```python

def add\_item(item, item\_list=[]):

item\_list.append(item)

return item\_list

list1 = add\_item(1)

list2 = add\_item(2)

print(list1)

print(list2)

```

<!-- The default mutable argument 'item\_list' causes shared references across calls -->

- Corrected Code Snippet:

```python

def add\_item(item, item\_list=None):

if item\_list is None:

item\_list = []

item\_list.append(item)

return item\_list

list1 = add\_item(1)

list2 = add\_item(2)

print(list1)

print(list2)

```

### Using class variables incorrectly

2. \*\*Using Class Variables Incorrectly\*\*:

- Aim: The aim of this code snippet is to correctly use class variables and instance variables.

- Wrong Code Snippet:

```python

class Dog:

tricks = [] # Class variable intended to store tricks

def add\_trick(self, trick):

self.tricks.append(trick)

dog1 = Dog()

dog2 = Dog()

dog1.add\_trick("roll over")

dog2.add\_trick("play dead")

print(dog1.tricks)

print(dog2.tricks)

```

<!-- Class variable 'tricks' shared among all instances -->

- Corrected Code Snippet:

```python

class Dog:

def \_\_init\_\_(self):

self.tricks = [] # Instance variable intended to store tricks

def add\_trick(self, trick):

self.tricks.append(trick)

dog1 = Dog()

dog2 = Dog()

dog1.add\_trick("roll over")

dog2.add\_trick("play dead")

print(dog1.tricks)

print(dog2.tricks)

```

### Specifying parameters incorrectly for an exception block

3. \*\*Specifying Parameters Incorrectly for an Exception Block\*\*:

- Aim: The aim of this code snippet is to correctly handle an exception with appropriate parameters.

- Wrong Code Snippet:

```python

try:

result = 10 / 0

except ZeroDivisionError, e:

print("Error: Division by zero.")

```

<!-- Incorrect syntax for specifying exception parameters -->

- Corrected Code Snippet:

```python

try:

result = 10 / 0

except ZeroDivisionError as e:

print(f"Error: Division by zero. Exception message: {e}")

```

### Misunderstanding Python scope rules

4. \*\*Misunderstanding Python Scope Rules\*\*:

- Aim: The aim of this code snippet is to correctly modify a global variable inside a function.

- Wrong Code Snippet:

```python

count = 0

def increment():

count += 1

increment()

print(count)

```

<!-- Local variable 'count' referenced before assignment -->

- Corrected Code Snippet:

```python

count = 0

def increment():

global count

count += 1

increment()

print(count)

```

### Modifying a list while iterating over it

5. \*\*Modifying a List While Iterating Over It\*\*:

- Aim: The aim of this code snippet is to correctly remove items from a list while iterating over it.

- Wrong Code Snippet:

```python

numbers = [1, 2, 3, 4, 5]

for number in numbers:

if number % 2 == 0:

numbers.remove(number)

print(numbers)

```

<!-- Modifying the list while iterating causes unexpected behavior -->

- Corrected Code Snippet:

```python

numbers = [1, 2, 3, 4, 5]

numbers\_to\_remove = [number for number in numbers if number % 2 == 0]

for number in numbers\_to\_remove:

numbers.remove(number)

print(numbers)

```

Sure! Here are the debugging questions for the specified concepts, following the provided format:

### 6. Confusing how Python binds variables in closures

1. \*\*Confusing how Python binds variables in closures\*\*:

- Aim: The aim of this code snippet is to correctly capture the loop variable in a closure.

- Wrong Code Snippet:

```python

def create\_multipliers():

multipliers = []

for i in range(5):

multipliers.append(lambda x: x \* i)

return multipliers

for multiplier in create\_multipliers():

print(multiplier(2))

```

<!-- The loop variable 'i' is captured incorrectly, leading to all lambdas using the last value of 'i' -->

- Corrected Code Snippet:

```python

def create\_multipliers():

multipliers = []

for i in range(5):

multipliers.append(lambda x, i=i: x \* i)

return multipliers

for multiplier in create\_multipliers():

print(multiplier(2))

```

### 7. Creating circular module dependencies

2. \*\*Creating circular module dependencies\*\*:

- Aim: The aim of this code snippet is to correctly import modules without creating a circular dependency.

- Wrong Code Snippet:

```python

# file a.py

import b

def func\_a():

print("Function A")

b.func\_b()

# file b.py

import a

def func\_b():

print("Function B")

a.func\_a()

```

<!-- Circular import causes an ImportError -->

- Corrected Code Snippet:

```python

# file a.py

def func\_a():

print("Function A")

# file b.py

import a

def func\_b():

print("Function B")

a.func\_a()

func\_b()

```

### 8. Name clashing with Python Standard Library modules

3. \*\*Name clashing with Python Standard Library modules\*\*:

- Aim: The aim of this code snippet is to correctly use a custom module without clashing with a Python Standard Library module.

- Wrong Code Snippet:

```python

# file random.py

def my\_function():

return "This is my custom function"

import random

print(random.my\_function())

```

<!-- The custom module 'random' clashes with the Python Standard Library 'random' module -->

- Corrected Code Snippet:

```python

# file my\_random.py

def my\_function():

return "This is my custom function"

import my\_random

print(my\_random.my\_function())

```

### 9. Misusing the \_\_del\_\_ method

4. \*\*Misusing the `\_\_del\_\_` method\*\*:

- Aim: The aim of this code snippet is to properly use the `\_\_del\_\_` method without causing issues.

- Wrong Code Snippet:

```python

class MyClass:

def \_\_del\_\_(self):

print("Cleaning up resources")

obj = MyClass()

del obj

print("Object deleted")

```

<!-- Misusing \_\_del\_\_ may cause unintended consequences, especially with circular references -->

- Corrected Code Snippet:

```python

class MyClass:

def close(self):

print("Cleaning up resources")

obj = MyClass()

obj.close()

del obj

print("Object deleted")

```

These questions cover a range of common Python mistakes, demonstrating their corrections with clear aims provided.

SET2   
  
Certainly! Here are revised questions for each topic, with slightly different errors to test other aspects within the same topic.

### 1. Misusing Expressions as Defaults for Function Arguments

1. \*\*Misusing Expressions as Defaults for Function Arguments\*\*:

- Aim: The aim of this code snippet is to correctly use a default mutable argument in a function.

- Wrong Code Snippet:

```python

def append\_to\_list(value, my\_list=[]):

my\_list += [value]

return my\_list

list\_a = append\_to\_list(1)

list\_b = append\_to\_list(2)

print(list\_a)

print(list\_b)

```

<!-- The default mutable argument 'my\_list' causes shared references across calls -->

- Corrected Code Snippet:

```python

def append\_to\_list(value, my\_list=None):

if my\_list is None:

my\_list = []

my\_list += [value]

return my\_list

list\_a = append\_to\_list(1)

list\_b = append\_to\_list(2)

print(list\_a)

print(list\_b)

```

### 2. Using Class Variables Incorrectly

2. \*\*Using Class Variables Incorrectly\*\*:

- Aim: The aim of this code snippet is to correctly use class variables and instance variables.

- Wrong Code Snippet:

```python

class Player:

score = 0 # Class variable intended to store score

def add\_score(self, points):

self.score += points

p1 = Player()

p2 = Player()

p1.add\_score(10)

p2.add\_score(20)

print(p1.score)

print(p2.score)

```

<!-- Class variable 'score' shared among all instances -->

- Corrected Code Snippet:

```python

class Player:

def \_\_init\_\_(self):

self.score = 0 # Instance variable intended to store score

def add\_score(self, points):

self.score += points

p1 = Player()

p2 = Player()

p1.add\_score(10)

p2.add\_score(20)

print(p1.score)

print(p2.score)

```

### 3. Specifying Parameters Incorrectly for an Exception Block

3. \*\*Specifying Parameters Incorrectly for an Exception Block\*\*:

- Aim: The aim of this code snippet is to correctly handle different exceptions using multiple except blocks.

- Wrong Code Snippet:

```python

try:

num = int("abc")

except ValueError as e:

print("Value Error:", e)

except ValueError, e:

print("Incorrect Value Error:", e)

```

<!-- Incorrect syntax and redundancy for specifying exception parameters -->

- Corrected Code Snippet:

```python

try:

num = int("abc")

except ValueError as e:

print("Value Error:", e)

```

### 4. Misunderstanding Python Scope Rules

4. \*\*Misunderstanding Python Scope Rules\*\*:

- Aim: The aim of this code snippet is to correctly modify a variable in an enclosing scope using nested functions.

- Wrong Code Snippet:

```python

def outer():

x = "hello"

def inner():

x = "world"

print(x)

inner()

print(x)

outer()

```

<!-- Inner function does not modify the variable in the outer scope -->

- Corrected Code Snippet:

```python

def outer():

x = "hello"

def inner():

nonlocal x

x = "world"

print(x)

inner()

print(x)

outer()

```

### 5. Modifying a List While Iterating Over It

5. \*\*Modifying a List While Iterating Over It\*\*:

- Aim: The aim of this code snippet is to correctly remove even numbers from a list while iterating over it.

- Wrong Code Snippet:

```python

nums = [1, 2, 3, 4, 5, 6]

for num in nums:

if num % 2 == 0:

nums.remove(num)

print(nums)

```

<!-- Modifying the list while iterating causes unexpected behavior -->

- Corrected Code Snippet:

```python

nums = [1, 2, 3, 4, 5, 6]

nums\_to\_remove = [num for num in nums if num % 2 == 0]

for num in nums\_to\_remove:

nums.remove(num)

print(nums)

```

### 6. Confusing how Python Binds Variables in Closures

6. \*\*Confusing how Python Binds Variables in Closures\*\*:

- Aim: The aim of this code snippet is to correctly capture the current value of the loop variable in a closure.

- Wrong Code Snippet:

```python

def create\_functions():

funcs = []

for i in range(3):

def func():

return i

funcs.append(func)

return funcs

for func in create\_functions():

print(func())

```

<!-- The loop variable 'i' is captured incorrectly, leading to all functions using the last value of 'i' -->

- Corrected Code Snippet:

```python

def create\_functions():

funcs = []

for i in range(3):

def func(i=i):

return i

funcs.append(func)

return funcs

for func in create\_functions():

print(func())

```

### 7. Creating Circular Module Dependencies

7. \*\*Creating Circular Module Dependencies\*\*:

- Aim: The aim of this code snippet is to correctly structure module imports to avoid circular dependencies.

- Wrong Code Snippet:

```python

# file module1.py

import module2

def function1():

print("Function 1")

module2.function2()

# file module2.py

import module1

def function2():

print("Function 2")

module1.function1()

```

<!-- Circular import causes an ImportError -->

- Corrected Code Snippet:

```python

# file module1.py

def function1():

print("Function 1")

# file module2.py

from module1 import function1

def function2():

print("Function 2")

function1()

function2()

```

### 8. Name Clashing with Python Standard Library Modules

8. \*\*Name Clashing with Python Standard Library Modules\*\*:

- Aim: The aim of this code snippet is to correctly use a custom module without clashing with a Python Standard Library module.

- Wrong Code Snippet:

```python

# file collections.py

def my\_function():

return "Custom collections function"

import collections

print(collections.my\_function())

```

<!-- The custom module 'collections' clashes with the Python Standard Library 'collections' module -->

- Corrected Code Snippet:

```python

# file my\_collections.py

def my\_function():

return "Custom collections function"

import my\_collections

print(my\_collections.my\_function())

```

### 9. Misusing the `\_\_del\_\_` Method

9. \*\*Misusing the `\_\_del\_\_` Method\*\*:

- Aim: The aim of this code snippet is to properly use the `\_\_del\_\_` method without causing issues.

- Wrong Code Snippet:

```python

class Resource:

def \_\_del\_\_(self):

print("Resource cleanup")

def create\_resource():

res = Resource()

print("Resource created")

create\_resource()

```

<!-- The \_\_del\_\_ method may not be called immediately due to delayed garbage collection -->

- Corrected Code Snippet:

```python

class Resource:

def cleanup(self):

print("Resource cleanup")

def create\_resource():

res = Resource()

print("Resource created")

res.cleanup()

create\_resource()

```

These questions test different aspects within the same topics, providing a broader range of debugging challenges.

New set  
  
Certainly! Here are debugging questions for each of the specified topics:

### 1. Exception Handling

1. \*\*Exception Handling\*\*:

- Aim: The aim of this code snippet is to handle a specific exception and provide a custom error message.

- Wrong Code Snippet:

```python

try:

result = 10 / 0

except ZeroDivisionError:

print("An error occurred")

```

<!-- Custom error message missing -->

- Corrected Code Snippet:

```python

try:

result = 10 / 0

except ZeroDivisionError:

print("Cannot divide by zero")

```

### 2. Scope and Namespace

2. \*\*Scope and Namespace\*\*:

- Aim: The aim of this code snippet is to correctly modify a variable in a nested function.

- Wrong Code Snippet:

```python

def outer\_function():

x = 10

def inner\_function():

x += 5

print(x)

inner\_function()

outer\_function()

```

<!-- Attempting to modify a variable from the outer scope without using 'nonlocal' -->

- Corrected Code Snippet:

```python

def outer\_function():

x = 10

def inner\_function():

nonlocal x

x += 5

print(x)

inner\_function()

outer\_function()

```

### 3. Mutable vs. Immutable Objects

3. \*\*Mutable vs. Immutable Objects\*\*:

- Aim: The aim of this code snippet is to correctly modify a mutable object within a function.

- Wrong Code Snippet:

```python

def add\_value(my\_list):

my\_list = my\_list + [4, 5, 6]

return my\_list

original\_list = [1, 2, 3]

modified\_list = add\_value(original\_list)

print(original\_list)

print(modified\_list)

```

<!-- Using assignment operator creates a new object instead of modifying the original list -->

- Corrected Code Snippet:

```python

def add\_value(my\_list):

my\_list.extend([4, 5, 6])

return my\_list

original\_list = [1, 2, 3]

modified\_list = add\_value(original\_list)

print(original\_list)

print(modified\_list)

```

### 4. List Comprehensions and Generators

4. \*\*List Comprehensions and Generators\*\*:

- Aim: The aim of this code snippet is to correctly use a list comprehension to generate a list of squares.

- Wrong Code Snippet:

```python

squares = [x \* x for x in range(5)]

print(squares)

```

<!-- Incorrect use of list comprehension syntax -->

- Corrected Code Snippet:

```python

squares = [x \*\* 2 for x in range(5)]

print(squares)

```

### 5. Object-Oriented Programming (OOP) Concepts

5. \*\*Object-Oriented Programming (OOP) Concepts\*\*:

- Aim: The aim of this code snippet is to correctly initialize a subclass with attributes from the superclass.

- Wrong Code Snippet:

```python

class Animal:

def \_\_init\_\_(self, name):

self.name = name

class Dog(Animal):

def \_\_init\_\_(self, name, breed):

self.breed = breed

dog = Dog("Buddy", "Golden Retriever")

print(dog.name, dog.breed)

```

<!-- Missing call to 'super().\_\_init\_\_' in Dog class -->

- Corrected Code Snippet:

```python

class Animal:

def \_\_init\_\_(self, name):

self.name = name

class Dog(Animal):

def \_\_init\_\_(self, name, breed):

super().\_\_init\_\_(name)

self.breed = breed

dog = Dog("Buddy", "Golden Retriever")

print(dog.name, dog.breed)

```

### 6. Module Structure and Packaging

6. \*\*Module Structure and Packaging\*\*:

- Aim: The aim of this code snippet is to correctly import a function from a custom module.

- Wrong Code Snippet:

```python

import my\_module

print(my\_module.my\_function())

```

<!-- Incorrect module name or file path -->

- Corrected Code Snippet:

```python

from my\_package import my\_module

print(my\_module.my\_function())

```

### 7. Regular Expressions

7. \*\*Regular Expressions\*\*:

- Aim: The aim of this code snippet is to correctly use a regular expression to find patterns in a string.

- Wrong Code Snippet:

```python

import re

text = "Hello World"

pattern = "\d+"

result = re.findall(pattern, text)

print(result)

```

<!-- Incorrect regular expression pattern to find digits -->

- Corrected Code Snippet:

```python

import re

text = "Hello 123 World"

pattern = "\d+"

result = re.findall(pattern, text)

print(result)

```

### 8. Debugging Techniques

8. \*\*Debugging Techniques\*\*:

- Aim: The aim of this code snippet is to use the debugger to identify and fix a logical error.

- Wrong Code Snippet:

```python

def factorial(n):

if n == 1:

return 1

else:

return n \* factorial(n-1)

result = factorial(5)

print(result)

```

<!-- Logical error in the factorial function -->

- Corrected Code Snippet:

```python

def factorial(n):

if n == 0:

return 1

else:

return n \* factorial(n-1)

result = factorial(5)

print(result)

```

### 9. Virtual Environments and Dependency Management

9. \*\*Virtual Environments and Dependency Management\*\*:

- Aim: The aim of this code snippet is to correctly create and activate a virtual environment.

- Wrong Code Snippet:

```bash

python -m venv my\_env

```

<!-- Missing command to activate the virtual environment -->

- Corrected Code Snippet:

```bash

python -m venv my\_env

source my\_env/bin/activate

```

### 10. Concurrency and Parallelism

10. \*\*Concurrency and Parallelism\*\*:

- Aim: The aim of this code snippet is to correctly use threading to execute multiple tasks concurrently.

- Wrong Code Snippet:

```python

import threading

import time

def print\_numbers():

for i in range(5):

print(i)

time.sleep(1)

t1 = threading.Thread(target=print\_numbers)

t2 = threading.Thread(target=print\_numbers)

t1.start()

t2.start()

```

<!-- Missing join() method to wait for threads to finish -->

- Corrected Code Snippet:

```python

import threading

import time

def print\_numbers():

for i in range(

***Hard Mistakes/Topics - 16 questions from GPT :***

SET1

Certainly! Here are debugging questions for each of the specified topics:

### 1. Algorithm Efficiency

1. \*\*Algorithm Efficiency\*\*:

- Aim: The aim of this code snippet is to optimize a function to find the sum of all even numbers in a given range.

- Wrong Code Snippet:

```python

def sum\_even\_numbers(n):

result = 0

for i in range(n):

if i % 2 == 0:

result += i

return result

print(sum\_even\_numbers(10))

```

<!-- Inefficient algorithm to find the sum of even numbers -->

- Corrected Code Snippet:

```python

def sum\_even\_numbers(n):

return sum(i for i in range(n) if i % 2 == 0)

print(sum\_even\_numbers(10))

```

### 2. Memory Management

2. \*\*Memory Management\*\*:

- Aim: The aim of this code snippet is to manage memory efficiently when working with large data structures.

- Wrong Code Snippet:

```python

data = [i for i in range(10\*\*6)]

```

<!-- Creating a large list in memory unnecessarily -->

- Corrected Code Snippet:

```python

import numpy as np

data = np.arange(10\*\*6)

```

### 3. Closures and Decorators

3. \*\*Closures and Decorators\*\*:

- Aim: The aim of this code snippet is to correctly implement a decorator to measure the execution time of a function.

- Wrong Code Snippet:

```python

import time

def measure\_time(func):

start\_time = time.time()

result = func()

end\_time = time.time()

print(f"Execution time: {end\_time - start\_time} seconds")

return result

@measure\_time

def example\_function():

time.sleep(2)

return "Example"

print(example\_function())

```

<!-- Incorrect decorator implementation -->

- Corrected Code Snippet:

```python

import time

from functools import wraps

def measure\_time(func):

@wraps(func)

def wrapper(\*args, \*\*kwargs):

start\_time = time.time()

result = func(\*args, \*\*kwargs)

end\_time = time.time()

print(f"Execution time: {end\_time - start\_time} seconds")

return result

return wrapper

@measure\_time

def example\_function():

time.sleep(2)

return "Example"

print(example\_function())

```

### 4. Metaprogramming

4. \*\*Metaprogramming\*\*:

- Aim: The aim of this code snippet is to dynamically create a class with a specified number of attributes.

- Wrong Code Snippet:

```python

class MyClass:

pass

def add\_attributes(cls, n):

for i in range(n):

setattr(cls, f'attr{i}', i)

add\_attributes(MyClass, 3)

obj = MyClass()

print(obj.attr0, obj.attr1, obj.attr2)

```

<!-- Incorrect usage of setattr to add attributes dynamically -->

- Corrected Code Snippet:

```python

def create\_class(n):

attrs = {f'attr{i}': i for i in range(n)}

return type('MyClass', (), attrs)

MyClass = create\_class(3)

obj = MyClass()

print(obj.attr0, obj.attr1, obj.attr2)

```

### 5. Concurrency Control

5. \*\*Concurrency Control\*\*:

- Aim: The aim of this code snippet is to handle a race condition when accessing a shared resource in a multithreaded environment.

- Wrong Code Snippet:

```python

import threading

counter = 0

def increment\_counter():

global counter

for \_ in range(100000):

counter += 1

t1 = threading.Thread(target=increment\_counter)

t2 = threading.Thread(target=increment\_counter)

t1.start()

t2.start()

t1.join()

t2.join()

print(counter)

```

<!-- Race condition when accessing the shared 'counter' variable -->

- Corrected Code Snippet:

```python

import threading

counter = 0

lock = threading.Lock()

def increment\_counter():

global counter

for \_ in range(100000):

with lock:

counter += 1

t1 = threading.Thread(target=increment\_counter)

t2 = threading.Thread(target=increment\_counter)

t1.start()

t2.start()

t1.join()

t2.join()

print(counter)

```

### 6. Advanced File I/O

6. \*\*Advanced File I/O\*\*:

- Aim: The aim of this code snippet is to correctly read and write binary data to a file.

- Wrong Code Snippet:

```python

with open('data.bin', 'w') as file:

file.write(b'Hello')

with open('data.bin', 'r') as file:

data = file.read()

print(data)

```

<!-- Incorrect file mode and encoding used for binary data -->

- Corrected Code Snippet:

```python

with open('data.bin', 'wb') as file:

file.write(b'Hello')

with open('data.bin', 'rb') as file:

data = file.read()

print(data)

```

### 7. Networking

7. \*\*Networking\*\*:

- Aim: The aim of this code snippet is to correctly implement a simple client-server communication using sockets.

- Wrong Code Snippet:

```python

import socket

server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

server\_socket.bind(('localhost', 12345))

server\_socket.listen(1)

client\_socket, \_ = server\_socket.accept()

data = client\_socket.recv(1024)

print(data.decode())

client\_socket.close()

server\_socket.close()

```

<!-- Incomplete server implementation -->

- Corrected Code Snippet:

```python

import socket

server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

server\_socket.bind(('localhost', 12345))

server\_socket.listen(1)

client\_socket, \_ = server\_socket.accept()

client\_socket.send(b'Hello from server')

client\_socket.close()

server\_socket.close()

```

### 8. Database Interaction

8. \*\*Database Interaction\*\*:

- Aim: The aim of this code snippet is to correctly connect to a SQLite database and execute a query.

- Wrong Code Snippet:

```python

import sqlite3

conn = sqlite3.connect('example.db')

cursor = conn.cursor()

cursor.execute('SELECT \* FROM users')

rows = cursor.fetchall()

for row in rows:

print(row)

cursor.close()

conn.close()

```

<!-- Missing table creation and data insertion -->

- Corrected Code Snippet:

```python

import sqlite3

conn = sqlite3.connect('example.db')

cursor = conn.cursor()

cursor.execute('''CREATE TABLE IF NOT EXISTS users

(id INTEGER PRIMARY KEY, name TEXT,

SET2

Certainly! Here's another set of debugging questions for the specified topics:

### 1. Algorithm Efficiency

1. \*\*Algorithm Efficiency\*\*:

- Aim: The aim of this code snippet is to optimize a function to find the sum of all prime numbers in a given range.

- Wrong Code Snippet:

```python

def sum\_prime\_numbers(n):

result = 0

for i in range(2, n):

prime = True

for j in range(2, i):

if i % j == 0:

prime = False

break

if prime:

result += i

return result

print(sum\_prime\_numbers(10))

```

<!-- Inefficient algorithm to find the sum of prime numbers -->

- Corrected Code Snippet:

```python

def is\_prime(num):

if num < 2:

return False

for i in range(2, int(num\*\*0.5) + 1):

if num % i == 0:

return False

return True

def sum\_prime\_numbers(n):

return sum(i for i in range(2, n) if is\_prime(i))

print(sum\_prime\_numbers(10))

```

### 2. Memory Management

2. \*\*Memory Management\*\*:

- Aim: The aim of this code snippet is to minimize memory usage when processing a large dataset.

- Wrong Code Snippet:

```python

data = [i \*\* 2 for i in range(10\*\*6)]

```

<!-- Creating a large list in memory -->

- Corrected Code Snippet:

```python

import numpy as np

data = np.arange(10\*\*6) \*\* 2

```

### 3. Closures and Decorators

3. \*\*Closures and Decorators\*\*:

- Aim: The aim of this code snippet is to correctly implement a decorator to log function calls.

- Wrong Code Snippet:

```python

def log(func):

print(f'Calling function {func.\_\_name\_\_}')

return func

@log

def example\_function():

return "Example"

print(example\_function())

```

<!-- Missing wrapper function in decorator -->

- Corrected Code Snippet:

```python

def log(func):

def wrapper(\*args, \*\*kwargs):

print(f'Calling function {func.\_\_name\_\_}')

return func(\*args, \*\*kwargs)

return wrapper

@log

def example\_function():

return "Example"

print(example\_function())

```

### 4. Metaprogramming

4. \*\*Metaprogramming\*\*:

- Aim: The aim of this code snippet is to dynamically create a class with custom methods.

- Wrong Code Snippet:

```python

def add\_method(cls, func):

setattr(cls, func.\_\_name\_\_, func)

class MyClass:

pass

def custom\_method(self):

return "Custom method"

add\_method(MyClass, custom\_method)

obj = MyClass()

print(obj.custom\_method())

```

<!-- Incorrect usage of setattr to add a method dynamically -->

- Corrected Code Snippet:

```python

def add\_method(cls, func):

setattr(cls, func.\_\_name\_\_, func)

class MyClass:

pass

def custom\_method(self):

return "Custom method"

add\_method(MyClass, custom\_method)

obj = MyClass()

print(obj.custom\_method()) # Outputs: Custom method

```

### 5. Concurrency Control

5. \*\*Concurrency Control\*\*:

- Aim: The aim of this code snippet is to prevent a race condition when updating a shared resource.

- Wrong Code Snippet:

```python

import threading

counter = 0

def increment\_counter():

global counter

counter += 1

threads = []

for \_ in range(100):

t = threading.Thread(target=increment\_counter)

t.start()

threads.append(t)

for t in threads:

t.join()

print(counter)

```

<!-- Race condition when updating the shared 'counter' variable -->

- Corrected Code Snippet:

```python

import threading

counter = 0

lock = threading.Lock()

def increment\_counter():

global counter

with lock:

counter += 1

threads = []

for \_ in range(100):

t = threading.Thread(target=increment\_counter)

t.start()

threads.append(t)

for t in threads:

t.join()

print(counter)

```

### 6. Advanced File I/O

6. \*\*Advanced File I/O\*\*:

- Aim: The aim of this code snippet is to correctly read and write data to a binary file.

- Wrong Code Snippet:

```python

with open('data.bin', 'w') as file:

file.write(b'Hello')

with open('data.bin', 'r') as file:

data = file.read()

print(data)

```

<!-- Incorrect file mode and encoding used for binary data -->

- Corrected Code Snippet:

```python

with open('data.bin', 'wb') as file:

file.write(b'Hello')

with open('data.bin', 'rb') as file:

data = file.read()

print(data)

```

### 7. Networking

7. \*\*Networking\*\*:

- Aim: The aim of this code snippet is to correctly implement a server that handles multiple client connections.

- Wrong Code Snippet:

```python

import socket

server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

server\_socket.bind(('localhost', 12345))

server\_socket.listen(1)

while True:

client\_socket, \_ = server\_socket.accept()

data = client\_socket.recv(1024)

print(data.decode())

client\_socket.close()

```

<!-- Server does not handle multiple client connections -->

- Corrected Code Snippet:

```python

import socket

import threading

def handle\_client(client\_socket):

data = client\_socket.recv(1024)

print(data.decode())

client\_socket.close()

server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

server\_socket.bind(('localhost', 12345))

server\_socket.listen(5)

while True:

client\_socket, \_ = server\_socket.accept()

t = threading.Thread(target=handle\_client, args=(client\_socket,))

t.start()

```

### 8. Database Interaction

8. \*\*Database Interaction\*\*:

- Aim: The aim of this code snippet is to correctly execute a SQL query to insert data into a SQLite database.

- Wrong Code Snippet:

```python

import sqlite3

conn = sqlite3.connect('example.db')

cursor = conn.cursor()

cursor.execute('INSERT INTO users (name) VALUES ("Alice")')

conn.commit()

cursor.close()

conn.close()

```

<!-- Missing table creation and incorrect data insertion -->

- Corrected Code Snippet:

```python

import sqlite3

conn = sqlite3.connect('example.db')

cursor = conn.cursor()

cursor.execute('''CREATE TABLE IF NOT EXISTS users

(id INTEGER