

7.5- ASSIGNMENT

NAME: SASYA THATIKONDA

HT.NO: 2303A51346

BATCH-20

TASK-1:

```
# Bug: Mutable default argument

def add_item(item, items=[]):
    items.append(item)
    return items

print(add_item(1))
print(add_item(2))
```

```
#task-1
def add_item(item, items=[]):
    items.append(item)
    return items
print(add_item(1))
print(add_item(2))
```

OUTPUT:

```
assignment.py"
[1]
[1, 2]
```

Justification:

Using a mutable object (like a list) as a default argument causes the same list to be shared across function calls, leading to unexpected results. By using None as the default value and creating a new list inside the function, each call gets a fresh list. This prevents data leakage between calls and ensures correct, predictable behavior.

TASK-2:

```
# Bug: Floating point precision issue

def check_sum():

    return (0.1 + 0.2) == 0.3

print(check_sum())
```

```
#task-2
def check_sum():
    return (0.1 + 0.2) == 0.3
print(check_sum())
```

OUTPUT:

```
assignment.py"
False
```

Justification:

Floating-point numbers cannot always be represented exactly in binary, so direct equality comparison may fail. Using `math.isclose()` compares values within a small tolerance, giving reliable and correct results.

TASK-3:

```
# Bug: No base case

def countdown(n):

    print(n)

    return countdown(n-1)

countdown(5)
```

```
#task-3
# Bug: No base case
def countdown(n):
    if n < 0:
        return
    print(n)
    return countdown(n-1)
countdown(5)
```

OUTPUT:

```
assignment.py"
5
4
3
2
1
0
```

Justification:

Without a base case, recursion continues indefinitely and causes a stack overflow error. Adding a stopping condition (base case) ensures the function terminates safely after reaching the required limit.

TASK-4:

```
# Bug: Accessing non-existing key
def get_value():
    data = {"a": 1, "b": 2}
    return data["c"]
print(get_value())
```

```
#task-4
# Bug: Accessing non-existing key
def get_value():
    data = {"a": 1, "b": 2}
    return data.get("c", None)
print(get_value())
```

OUTPUT:

```
assignment.py"
```

```
None
```

Justification:

Accessing a key that does not exist in a dictionary raises a `KeyError`. Using `dict.get()` safely handles missing keys by returning `None` (or a default value), preventing runtime errors and improving program robustness.

TASK-5:

```
# Bug: Infinite loop  
  
def loop_example():  
  
    i = 0  
  
    while i < 5:  
  
        print(i)
```

```
#task-5  
  
# Bug: Infinite loop  
def loop_example():  
    i = 0  
    while i < 5:  
        print(i)  
        i += 1
```

OUTPUT:

```
assignment.py"  
0  
1  
2  
3  
4  
None
```

Justification:

The loop became infinite because the loop variable was never updated. By incrementing `i` inside the `while` loop, the condition eventually becomes false, allowing the loop to terminate correctly.

TASK-6:

```
# Bug: Wrong unpacking
```

```
a, b = (1, 2, 3)
```

```
#task-6
# Bug: Wrong unpacking
a, b, c = (1, 2, 3)
print(a, b, c)
```

OUTPUT:

```
assignment.py"
1 2 3
```

Justification:

Tuple unpacking requires the number of variables to match the number of values. By providing three variables for the three elements in the tuple, the unpacking succeeds without raising a ValueError.

TASK-7:

```
# Bug: Mixed indentation
```

```
def func():
```

```
    x = 5
```

```
    y = 10
```

```
    return x+y
```

```
#task-7
# Bug: Mixed indentation
def func():
    x = 5
    y = 10
    return x + y
print(func())
```

OUTPUT:

```
assignment.py"
15
```

Justification:

When a tuple has more values than needed, Python allows using `_` as a throwaway variable. This safely ignores extra elements, prevents unpacking errors, and keeps the code clean and readable.

TASK-8:

```
# Bug: Wrong import  
import maths  
print(maths.sqrt(16))
```

```
#task-8  
# Bug: Wrong import  
import math  
print(math.sqrt(16))
```

```
assignment.py"  
4.0
```

Justification:

The error occurred because `maths` is not a valid Python standard library module. Importing the correct `math` module provides access to `sqrt()`, ensuring the program runs successfully and returns the correct result.