

Task 7 Debugging Practice Bank

Task 7: Programming Practical Test - Unit 4 - 7.5%

Task 7 Debugging Practice Bank

Use these short tasks to practise before the programming practical test. For each task, identify the error type, explain the problem and write a correction.

Practice 1: Syntax Error

```
1 IF number0fSessions > 0  
2 THEN  
3     average = (number0fSessions + 1) / 2  
4
```

Model Answer

The error is a syntax error because the `IF` line is missing `THEN`.

```
1 IF number0fSessions > 0  
2 THEN  
3     average = (number0fSessions + 1) / 2  
4
```

Practice 2: Logical Error

```
1 totalStudyMinutes = 0  
2
```

The program is meant to calculate total study minutes.

Model Answer

The error is a logical error because the program runs but calculates the wrong result. The statement should add the two values.

```
1 totalStudyMinutes = 0  
2
```

Practice 3: Run-Time Error

```
1 average = (number0fSessions + 1) / 2  
2
```

Model Answer

A run-time error could occur if `number0fSessions` is 0 because division by zero is not valid. The program should validate that `number0fSessions` is greater than 0 before calculating the average.

Practice 4: Boundary Error

```
def count_balanced_sessions(start, end):  
    count = 0  
    for minutes in range(start, end):  
        if is_balanced(minutes):  
            count += 1  
    return count
```

The program should count totals from 30 to 60 minutes as balanced.

Model Answer

The error is a logical error. The boundary values 30 and 60 are incorrectly excluded.

```
def count_balanced_sessions(start, end):  
    count = 0  
    for minutes in range(start, end + 1):  
        if is_balanced(minutes):  
            count += 1  
    return count
```

Practice 5: Counter Error

```
def count_balanced_sessions(start, end):  
    count = 1  
    for minutes in range(start, end):  
        if is_balanced(minutes):  
            count = 1  
        else:  
            count += 1  
    return count
```

Model Answer

The error is a logical error. The counter is reset to 1 each time a balanced session is found instead of increasing by 1.

```
def count_balanced_sessions(start, end):  
    count = 1  
    for minutes in range(start, end):  
        if is_balanced(minutes):  
            count += 1  
        else:  
            count = 1  
    return count
```

Practice 6: Validation Gap

```
def count_balanced_sessions(start, end):  
    count = 0  
    for minutes in range(start, end):  
        count += 1  
    return count
```

Model Answer

The code accepts any value. If the user enters a negative number, the total becomes inaccurate. The program should check that minutes is not negative before adding it.

```
def count_balanced_sessions(start, end):  
    count = 0  
    for minutes in range(start, end):  
        if minutes >= 0:  
            count += 1  
    return count
```

Practice 7: Trace To Find The Error

```
def count_balanced_sessions(start, end):  
    count = 0  
    for minutes in range(start, end):  
        if is_balanced(minutes):  
            count += 1  
    return count
```

```
total <- 0
for (i in 1:length(minutes)) {
  total <- minutes[i]
}
```

Test data: 10, 15, 20.

	count	minutes	total
	start	-	0
	1	10	10
	2	15	15
	3	20	20

Model Answer

The program only outputs the final input value. The error is in `total <- minutes` . It should accumulate:

```
total <- 0
for (i in 1:length(minutes)) {
  total <- total + minutes[i]
}
```

Practice 8: Documentation Fix

Weak comment:

```
# Loop through minutes and calculate total
```

Stronger comment:

```
# Calculate total minutes for each count
```

Good comments explain why the code exists or what important logic does. They do not repeat obvious code.