#14_Steps_for_Debugging

23/05/2025

Reproduce the error

- 1st step is to reproduce the problem
  - half the battle in many situations
  - becomes very critical in sophisticated applications

Determine the Error Boundaries

24/05/2025

- After being able to consistently reproduce the error, you should tweak the code/data that caused the error.

e.g.,

```
Traceback (most recent call last)
  File "/Users/xyzzy/python-project/test.py", line 5, in <module>
    result = filter_even_numbers(numbers)
             ^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/Users/xyzzy/python-project/test.py", line 2, in filter_even_numbers
    return list(filter(lambda x: x%2==0, numbers))
                ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
TypeError: 'NoneType' object is not iterable
```

NB: in the file we are executing

- error occurred in the `filter_even_numbers` on line 2 when attempting to iterate over the `numbers` collection
- `'NoneType' object is not iterable` indicates the collection is of type `NoneType`. i.e., `numbers` initialised from something that returned `None`

Trace the code

- once the boundaries of problem are understood, trace the code.

THIS IS THE STACK TRACK FROM #13_Debugging

get retrieves the value associated with key, 'name'. i.e. 'name': 'Alice' (04/05/2025)
'name' = key, 'Alice' = value

```
def process_student(student_data):
    name = student_data.get('name')
    grade = student_data.get('grade')
    return (name, grade)
```

tuple

if argument passed to get doesn't exist, get returns None

```
def average_grade(grades):
    total = sum(grades)
    average = total / len(grades)
    return average

students = [
    {'name': 'Alice', 'grade': 85},
    {'name': 'Bob'}, {'name': 'Jack', 'grade': 72},
    {'name': 'Jane', 'grade': 75},
]

def collect_grades(students):
    grades = []
    for student in students:
        name, grade = process_student(student)
        grades.append(grade)
    return grades

grades = collect_grades(students)
print(average_grade(grades))
# TypeError: unsupported operand type(s) for +: 'int'
# and 'NoneType'
```

line 25 error occurs at this line

FIRST STEP

Try to reproduce the error consistently.

☑ error is reproduced everytime program is run.

24/05/2025

SECOND STEP

- Determine the error boundaries by reducing input size to a single student

```
# Insert this next line below the current 'students' assignment.
students = [{'name': 'Alice', 'grade': 85}]

# rest of the code omitted

print(average_grade(grades)) # 85
```

For the first student, everything works.

```
students = [{'name': `Bob'}]

# rest of the code omitted

print(average_grade(grades))  # TypeError: unsupported
                                operand type(s) for +: `int'
                                and 'NoneType'
```

- For the second student that does NOT have a grade, we get a repeat of the error.
- So try with one student with a grade and one without a grade.

```
students = [{'name': 'Alice', `grade': 85}, {'name': `Bob'}]
# rest of code omitted

print(average_grade(grades))
# TypeError: unsupported operand type(s) for +: 'int' and 'NoneType'
```

THIRD STEP

24/05/2025

- Considering the consistency in the error message, now we know exactly what inputs are causing trouble and trace code back to where it originates.

- When reviewing the stack trace
  - program makes it to invoking `average_grade` without error. i.e, `collect_grades` does not produce errors, although it could be giving us something unexpected in our return value

- The line that throws an error is where invoking `sum` and passing in `grades`

- Trapping the error:
  the identification of identifying where the error originates.

Understand the Problem Well

- Analyse the code which is subject to the produced error.

```
def average_grade(grades):
    print(grades)
    total = sum(grades)
    average = total / len(grades)
    return average
```

`print` to determine the input to function i.e., [85, None, 72, 75]

- When passing the original students list, [85, None, 72, 75] is the output.

THIS IS THE PROBLEM.

24/05/2025

Implement a Fix

- Could suppress the error from being thrown.

```
try:
    print(average_grade(grades))
except Exception:
    print("Something went wrong.")
```

- original error in `average_grade` still occurs but this may be the only solution, i.e., dealing with edge cases when library or code that cannot be modified.

- could fix the `average_grade` function or even better fix `collect_grades` function.
  i.e., if the student does not have a grade like Bob, don't add anything to the list.

```
def collect_grades(students):
    grades = []
    for student in students:
        name, grade = process_student(student)
        if grade: # i.e., if grade is True
            grades.append(grade)
    return grades.
```

FIX ONE PROBLEM AT A TIME!!!
RESIST TRYING TO FIX MULTIPLE ERRORS AT ONCE!

24/05/2025

Test the Fix

- Test with similar sets of data
- This is manual testing.
- Later on you'll learn about automatic testing which prevents regression

25/05/2025

*hen testing only one graded student with a grade of 0, an error is returned.

```
students = [{'name': 'Bob', 'grade': 100}]
```

```
Traceback (most recent call last):
  File "/....."
    print(average_grade(grades))
  File "/....."
    average = total / len(grades)
ZeroDivisionError: division by zero
```

→ because when grade = 0
if grade: is False
∴ grades = [] which is an empty list with a length of zero.

Hence:

average = total / len(grades)
= 0 / 0
= ZeroDivisionError: division by zero.

26/05/2025

→ To allow for a grade of 0, we can include an if statement within collect_grades function. to create a test with a logical operator that includes '0's

26/05/2025

```
def collect_grades(students):
    grades = []
    for student in students:
        name, grade = process_student(student)
        if grade != None:
            grades.append(grade)
    return grades.
```

If only one grade is included and it is 0, the list grades = [0] of length 1. i.e., the denominator != 0.
for average = total / len(grades)

For now we are only interested in whether a number is assigned to 'grade' or there is no 'grade': value pair. To be complete, you would also want to make sure to handle other values assigned, such as alphabetical characters.
e.g., 'grade': 'A'