Jeremy Evert

Computer Science 1

Problem 1: 2.14 LAB Driving costs

Code (the main thing)

```
# 2.14 LAB: Driving costs
# Read gas mileage (miles per gallon) and gas cost (dollars per gallon),
# then compute gas cost for driving 20, 75, and 500 miles.

gas_mileage_mpg = float(input())
gas_price_per_gallon = float(input())

miles_20 = 20.0
miles_75 = 75.0
miles_500 = 500.0

gallons_for_20_miles = miles_20 / gas_mileage_mpg
gallons_for_75_miles = miles_75 / gas_mileage_mpg
gallons_for_500_miles = miles_500 / gas_mileage_mpg

cost_for_20_miles = gallons_for_20_miles * gas_price_per_gallon
cost_for_75_miles = gallons_for_75_miles * gas_price_per_gallon
cost_for_500_miles = gallons_for_500_miles * gas_price_per_gallon

print(f'{cost_for_20_miles:.2f} {cost_for_75_miles:.2f} {cost_for_500_miles:.2f}')
```

Run Results (proof it runs)

Input:

```
25.0

3.1599
```

Output:

```
2.53 9.48 63.20
```

Quick Reflection (what you learned)

I learned how to convert a “cost per gallon” and “miles per gallon” into a per-trip cost using unit conversions (miles → gallons → dollars). I also practiced formatting floating-point output to exactly two decimals so it matches the autograder.

Problem 2: 19.1.1 LAB Convert to seconds

Code (the main thing)

```
# 19.1.1 LAB: Convert to seconds

# Read seconds, minutes, and hours, then convert to total seconds.


input_seconds = int(input())

input_minutes = int(input())

input_hours = int(input())


seconds_from_minutes = input_minutes * 60

seconds_from_hours = input_hours * 3600


total_seconds = input_seconds + seconds_from_minutes + seconds_from_hours


print(f"{total_seconds} seconds")
```

Run Results (proof it runs)

Input:

```
40

6

1
```

Output:

```
4000 seconds
```

Quick Reflection (what you learned)

This reinforced how important it is to do clean unit conversions (minutes to seconds, hours to seconds) before combining values. I also practiced building readable variable names so the math is easy to follow later.

Problem 3: 2.13 LAB Divide input integers (Extra-credit “verbose + help mode” version)

Code (the main thing)

```
"""
Divide Input Integers (Verbose + Function Explorer Edition)


What this program does:
- Reads a numerator (top number) and a divisor (bottom number).
- Divides three times.
- After each division, it floors the result and reports the "lost" fractional amount.
- Prints every step verbosely so you can see exactly what happened and when printing happened.


Interactive Help Mode:
- At the main menu, press:
  h  -> open function explorer (shows a numbered function list)
  r  -> run the divider
  q  -> quit
```

```
Inside Help Mode:
- Pick a number to view a function’s docstring.
- Then choose:
   b  -> back to function list
   c  -> return to calculator menu
   q  -> quit program

Command-line examples (Windows CMD):
1) List functions in this file:
  py -c "import inspect, divide_input_integers_verbose_function_heavy as m; print('\n'.join(sorted([n for n,v in m.__dict__.items() if inspect.isfunction(v) and v.__module__==m.__name__])))"

2) Pretty list (function + first docstring line):
  py -c "import inspect, divide_input_integers_verbose_function_heavy as m; funcs=[(n,v) for n,v in m.__dict__.items() if inspect.isfunction(v) and v.__module__==m.__name__]; funcs=sorted(funcs); print('\n'.join([f'{n}: {(inspect.getdoc(v).splitlines()[0] if inspect.getdoc(v) else \"(no docstring)\")}' for n,v in funcs]))"

3) Show help for one function:
  py -c "import divide_input_integers_verbose_function_heavy as m; help(m.get_top_number)"
"""

import math
import inspect
```

```
def verbose_print(message: str) -> None:
    """Print a message with extra visibility around printing."""
    print("\n[PRINT ABOUT TO HAPPEN]")
    print(message)
    print("[PRINT COMPLETED]")


def get_top_number() -> int:
    """Prompt for the numerator (top number) and return it as an integer."""
    while True:
        raw_input_text = input("Enter the TOP number (numerator): ")
        try:
            numerator = int(raw_input_text)
            verbose_print(f"Captured numerator = {numerator}")
            return numerator
        except ValueError:
            verbose_print(f"'{raw_input_text}' is not a valid integer. Try again.")


def get_bottom_number() -> int:
    """Prompt for the divisor (bottom number) and return it as an integer (non-zero)."""
    while True:
        raw_input_text = input("Enter the BOTTOM number (divisor): ")
        try:
            divisor = int(raw_input_text)
            if divisor == 0:
                verbose_print("Divisor cannot be 0. Please enter a non-zero integer.")
                continue
            verbose_print(f"Captured divisor = {divisor}")
            return divisor
        except ValueError:
            verbose_print(f"'{raw_input_text}' is not a valid integer. Try again.")


def calculate_exact_division(numerator: int, divisor: int) -> float:
    """Perform exact division and return the full float result."""
    verbose_print(f"About to compute exact division: {numerator} / {divisor}")
    exact_result = numerator / divisor
    verbose_print(f"Exact division result = {exact_result}")
    return exact_result


def floor_and_report_loss(exact_value: float) -> tuple[int, float]:
    """Floor a float to an integer and report what was removed."""
    verbose_print(f"About to floor the value: {exact_value}")
    floored_value = math.floor(exact_value)
    lost_fraction = exact_value - floored_value
    verbose_print(f"Floored value = {floored_value}")
    verbose_print(f"Amount lost due to flooring = {lost_fraction}")
    return floored_value, lost_fraction
```

```
def first_division_step(current_value: int, divisor: int) -> float:

    """Perform the first exact division step."""

    verbose_print("=== FIRST DIVISION STEP (Exact) ===")

    return calculate_exact_division(current_value, divisor)



def first_rounding_step(exact_value: float) -> int:

    """Perform the first rounding step: floor the value and report loss."""

    verbose_print("=== FIRST ROUNDING STEP (Floor + Loss) ===")

    floored_value, _ = floor_and_report_loss(exact_value)

    return floored_value



def second_division_step(current_value: int, divisor: int) -> float:

    """Perform the second exact division step."""

    verbose_print("=== SECOND DIVISION STEP (Exact) ===")

    return calculate_exact_division(current_value, divisor)



def second_rounding_step(exact_value: float) -> int:

    """Perform the second rounding step: floor the value and report loss."""

    verbose_print("=== SECOND ROUNDING STEP (Floor + Loss) ===")

    floored_value, _ = floor_and_report_loss(exact_value)

    return floored_value
```

```
def third_division_step(current_value: int, divisor: int) -> float:
    """Perform the third exact division step."""
    verbose_print("=== THIRD DIVISION STEP (Exact) ===")
    return calculate_exact_division(current_value, divisor)


def third_rounding_step(exact_value: float) -> int:
    """Perform the third rounding step: floor the value and report loss."""
    verbose_print("=== THIRD ROUNDING STEP (Floor + Loss) ===")
    floored_value, _ = floor_and_report_loss(exact_value)
    return floored_value


def ask_run_again() -> bool:
    """Ask if the user wants to run the divider again."""
    while True:
        raw_choice = input("\nRun again? (y/n): ").strip().lower()
        if raw_choice in ("y", "yes"):
            verbose_print("User chose to run again.")
            return True
        if raw_choice in ("n", "no"):
            verbose_print("User chose to stop.")
            return False
        verbose_print("Please type 'y' or 'n'.")
```

```python
def get_local_functions() -> list[tuple[str, object]]:
    """Return a sorted list of (function_name, function_object) defined in this module."""
    current_module = inspect.getmodule(get_local_functions)
    items = []
    for name, value in current_module.__dict__.items():
        if inspect.isfunction(value) and value.__module__ == current_module.__name__:
            if not name.startswith("__"):
                items.append((name, value))
    return sorted(items, key=lambda pair: pair[0])


def show_help_mode() -> None:
    """Interactive function explorer: list functions, view docstrings, return to menu."""
    functions = get_local_functions()

    while True:
        verbose_print("=== HELP MODE: Function Explorer ===")
        print("Available functions:\n")
        for index, (name, _) in enumerate(functions, start=1):
            print(f"  {index:2d}) {name}")

        print("\nChoose:")
        print("  - Type a number to view that function's docstring")
        print("  - Type 'c' to return to calculator menu")
        print("  - Type 'q' to quit program")
```

```
choice = input("\nHelp> ").strip().lower()

if choice == "c":
    verbose_print("Leaving Help Mode. Returning to calculator menu.")
    return
if choice == "q":
    verbose_print("Quitting program from Help Mode.")
    raise SystemExit

if not choice.isdigit():
    verbose_print("Not a number. Try again.")
    continue

selected_index = int(choice)
if not (1 <= selected_index <= len(functions)):
    verbose_print("Number out of range. Try again.")
    continue

function_name, function_object = functions[selected_index - 1]
doc = inspect.getdoc(function_object) or "(No docstring available.)"

verbose_print(f"=== DOCSTRING: {function_name} ===")
print(doc)

while True:
```

```
            print("\nChoose next:")
            print("  b = back to function list")
            print("  c = return to calculator menu")
            print("  q = quit program")

            sub_choice = input("Doc> ").strip().lower()
            if sub_choice == "b":
                break
            if sub_choice == "c":
                return
            if sub_choice == "q":
                raise SystemExit
            verbose_print("Please type b, c, or q.")


def run_one_verbose_cycle() -> None:
    """Run one full cycle of 3 divisions + floor reporting and then summarize."""
    verbose_print("Starting a new verbose division cycle...")

    numerator = get_top_number()
    divisor = get_bottom_number()

    verbose_print(f"Verification: numerator = {numerator}")
    verbose_print(f"Verification: divisor   = {divisor}")

    first_exact = first_division_step(numerator, divisor)
    first_floored = first_rounding_step(first_exact)

    second_exact = second_division_step(first_floored, divisor)
    second_floored = second_rounding_step(second_exact)

    third_exact = third_division_step(second_floored, divisor)
    third_floored = third_rounding_step(third_exact)

    verbose_print("=== SUMMARY ===")
    verbose_print(f"Final floored results (space-separated): {first_floored} {second_floored} {third_floored}")


def main() -> None:
    """Main menu loop: r=run divider, h=help mode, q=quit."""
    verbose_print("Welcome to the ultra-verbose divider + function explorer.")

    while True:
        print("\nMain Menu:")
        print("  r = run divider")
        print("  h = help (function explorer)")
        print("  q = quit")

        choice = input("\nSelect> ").strip().lower()

        if choice == "q":
            verbose_print("All done. Goodbye!")
            return
        if choice == "h":
            show_help_mode()
            continue
        if choice == "r":
            run_one_verbose_cycle()
            if not ask_run_again():
                verbose_print("Returning to main menu.")
            continue

        verbose_print("Unknown choice. Please type r, h, or q.")


if __name__ == "__main__":
    main()
```

Run Results (proof it runs)

```
J:\git\SwosuCsPythonExamples\CS1_CS2\Ch02\ChatGPT_Assisted>dir
 Volume in drive J is cave
 Volume Serial Number is D650-96C8

 Directory of J:\git\SwosuCsPythonExamples\CS1_CS2\Ch02\ChatGPT_Assisted

01/26/2026  10:49 AM    <DIR>          .
```

```
01/26/2026  10:14 AM    <DIR>          ..
01/26/2026  10:49 AM            13,888 Chapter_2_report_Jeremy_Evert.docx
01/26/2026  10:29 AM            11,263 divide_input_integers_verbose_function_heavy.py
01/26/2026  10:20 AM               183 divide_three_times_easy.py
01/26/2026  10:43 AM               813 driving_cost_calculator.py
01/26/2026  10:37 AM             1,862 driving_cost_calculator_readme.md
01/26/2026  10:24 AM    <DIR>          __pycache__
               5 File(s)         28,009 bytes
               3 Dir(s)  7,550,637,432,832 bytes free


J:\git\SwosuCsPythonExamples\CS1_CS2\Ch02\ChatGPT_Assisted>python divide_three_times_easy.py
250


J:\git\SwosuCsPythonExamples\CS1_CS2\Ch02\ChatGPT_Assisted>
```

Example “calculator” run (simple test):

- Numerator input: 2000
- Divisor input: 2
- Final summary line should include: 1000 500 250

Example Help Mode evidence:

- From main menu, press h
- Show the numbered function list
- Pick a function number
- Show the docstring text

(Paste screenshot or paste output here.)

Quick Reflection (what you learned)

I learned how to organize a program using small functions with clear docstrings so the code is readable and self-documenting. I also learned how to use Python introspection (inspect) to build a help menu that lists functions and displays their docstrings interactively.

Extra Credit Notes (fill in what you did)

- Improvements beyond “it works”:
  - ☐ Improved 1 problem (+10)
  - ☐ Improved 2 problems (+20)
  - ☐ Improved 3 problems (+30)
  - Notes: ____
- ☐ Recorded a short video (+25)
  Link/filename: _______________________________________
- ☐ Pushed code to the shared repo (+50)
  Link: ____
- ☐ Presented during class (+25)
