**Task No. 1:** Refactor this code to remove duplicated logic & make it a working code.

def print\_report(data):

last\_value = None

diff\_list = []

for val in data:

if last\_value is not None:

diff = abs(val - last\_value)

diff\_list.append(diff)

last\_value = val

print(f"Max difference: {max(diff\_list)}")

print(f"Min difference: {min(diff\_list)}")

print(f"Average difference: {sum(diff\_list)/len(diff\_list)}")

**Solution:**

def calculate\_differences(data):

diff\_list = [abs(data[i] - data[i - 1]) for i in range(1, len(data))]

return diff\_list

def print\_report(data):

diff\_list = calculate\_differences(data)

if diff\_list:

print(f"Max difference: {max(diff\_list)}")

print(f"Min difference: {min(diff\_list)}")

print(f"Average difference: {sum(diff\_list) / len(diff\_list)}")

else:

print("No differences to calculate.")

data = [10, 15, 20, 25, 30]

calculate\_differences(data)

print\_report(data)

A black text on a white background

Description automatically generated**Output:**

**Task No. 2:** Refactor this code to simplify the conditional logic & make it a working code.

def calculate\_shipping(region, weight):

if region == "US":

if weight <= 2:

return 3

elif weight <= 6:

return 5

elif weight <= 10:

return 8

else:

return 10

elif region == "Canada":

if weight <= 2:

return 4

**Solution:**

def calculate\_shipping(region, weight):

shipping\_rates = {

"US": [(2, 3), (6, 5), (10, 8)],

"Canada": [(2, 4)]

}

for target\_weight, rate in shipping\_rates.get(region, []):

if weight <= target\_weight:

return rate

return None

region = "US"

weight = 3

result = calculate\_shipping(region, weight)

if result is not None:

print(f"The shipping cost for {region} and {weight} lbs is ${result}.")

else:

print("Shipping information not found.")

**Output:**



**Task No. 3:** Identify code smells in the following method and refactor the code to address them. Explain your changes & make it a working code.

def process\_data(filename):

data = []

with open(filename, 'r') as file:

for line in file:

parts = line.split('\t')

name = parts[0]

date\_str = parts[1].strip

date\_parts = date\_str.split('-')

month = int(date\_parts[0])

day = int(date\_parts[1])

year = int(date\_parts[2])

entry = {'name': name, 'month': month, 'day': day, 'year': year}

data.append(entry)

file.close()

return data

**Solution:**

def process\_data(filename):

processed\_data = []

with open(filename, 'r') as file:

for line in file:

name, date\_str = map(str.strip, line.split('\t'))

month, day, year = map(int, date\_str.split('-'))

entry = {'name': name, 'month': month, 'day': day, 'year': year}

processed\_data.append(entry)

return processed\_data

filename = "data.txt"

result = process\_data(filename)

for entry in result:

print(entry)

**Output:**

A number with black text

Description automatically generated with medium confidence

Explaination

**Inefficient File Closure:** The file.close() statement is unnecessary because the open construct automatically closes the file when the block is exited.

**Incorrect Function Call:** date\_str.strip is a function reference, but it should be date\_str.strip() to actually call the strip method.

**Inconsistent Indentation:** The indentation of the file.close() line is not consistent with the rest of the code.

**Inconsistent Naming:** The variable names are clear but could be made more descriptive for better readability.

**Task No. 4:** Refactor this code to remove duplicate code smell & make it working code.

class SaveFile:

def save\_txt(self, data, filename):

with open(filename + ".txt", "w") as file:

file.write(data)

file.close()

def save\_csv(self, data, filename):

with open(filename + ".csv", "w") as file:

file.write(data)

file.close()

**Solution:**

class SaveFile:

def \_save\_data(self, data, filename, file\_extension):

with open(filename + file\_extension, "w") as file:

file.write(data)

def save\_txt(self, data, filename):

self.\_save\_data(data, filename, ".txt")

def save\_csv(self, data, filename):

self.\_save\_data(data, filename, ".csv")

save\_file\_instance = SaveFile()

txt\_data = "This is text data."

csv\_data = "Name, Age\nJohn, 25\nAlice, 30"

save\_file\_instance.save\_txt(txt\_data, "text\_data")

save\_file\_instance.save\_csv(csv\_data, "csv\_data")

**Task No. 5:** Refactor this code to simplify method calls & make it working code.

import util

import datetime

def validate\_data(data):

if util.is\_empty(data):

raise Exception("Data is empty")

if not util.is\_valid\_format(data):

raise Exception("Invalid format")

if not util.is\_valid\_format(datatime.str\_to\_date(data[“date”])):

raise Exception("Invalid date")

**Solution:**

import util

import datetime

def validate\_data(data):

if util.is\_empty(data):

raise ValueError("Data is empty")

if not util.is\_valid\_format(data):

raise ValueError("Invalid format")

date\_str = data.get("date")

if not date\_str or not util.is\_valid\_format(datetime.str\_to\_date(date\_str)):

raise ValueError("Invalid date")

data = {"date": "2023-12-17", "other\_data": "some\_value"}

try:

validate\_data(data)

print("Data is valid.")

except ValueError as e:

print(f"Validation failed: {e}")

**Output:**

**Task No. 6:** Identify code smells in this method and refactor to improve readability and reduce complexity & make it working code.

def process\_order(order):

tax = 0

shipping = 0

subtotal = 0

discount = 0

for item in order:

qty = item["qty"]

price = item["price"]

subtotal += qty \* price

if item["type"] == 1:

tax += qty \* price \* 0.075

if qty > 10:

discount += qty \* price \* 0.1

if item["ship"] == "Priority":

shipping += 5

**Solution:**

TAX\_RATE = 0.075

DISCOUNT\_RATE = 0.1

DISCOUNT\_THRESHOLD = 10

PRIORITY\_SHIPPING\_COST = 5

def calculate\_subtotal(order):

subtotal = 0

for item in order:

qty = item["qty"]

price = item["price"]

subtotal += qty \* price

return subtotal

def calculate\_tax(order):

tax = 0

for item in order:

qty = item["qty"]

price = item["price"]

if item["type"] == 1:

tax += qty \* price \* TAX\_RATE

return tax

def calculate\_discount(order):

discount = 0

for item in order:

qty = item["qty"]

price = item["price"]

if qty > DISCOUNT\_THRESHOLD:

discount += qty \* price \* DISCOUNT\_RATE

return discount

def calculate\_shipping(order):

shipping = 0

for item in order:

if item["ship"] == "Priority":

shipping += PRIORITY\_SHIPPING\_COST

return shipping

def process\_order(order):

subtotal = calculate\_subtotal(order)

tax = calculate\_tax(order)

discount = calculate\_discount(order)

shipping = calculate\_shipping(order)

total = subtotal + tax - discount + shipping

# Return the results or perform additional actions

return {

"subtotal": subtotal,

"tax": tax,

"discount": discount,

"shipping": shipping,

"total": total

}

order = [

{"qty": 5, "price": 10, "type": 1, "ship": "Priority"},

{"qty": 12, "price": 8, "type": 2, "ship": "Regular"},

]

result = process\_order(order)

print(result)

**Output:**

