Code Analysis Report

Analyzed Code:

```
"""
module name: purchase
function name: purchase
overview of this function:
1) Customer interaction for what and how much the want buy.
2) check the user interaction valid or not with exception handelling.
3) calculating the customer purchase product with discount(if discountable)
4) show the last update of the product
5) write the invoice for customer with unique naming
"""

def purchase(List):
L = List # assign list with variable name 'L'.
a_name = input("Please enter your name: ")
print("\nHello " + a_name + "! Welcome to our Electronic Store.\nLook at
above and select product as your choice.")
q = {} # assign empty dictionary with variable name 'q'.
flag = "Y"
while flag.upper() == "Y": # check and go if flag is 'Y' or 'y'.
product = input("\nWhat product do you want to buy? ")
product_name = product.upper() # change the string value in the upper case.
if product_name == L[0][0].upper() \
or product_name == L[1][0].upper() \
or product_name == L[2][0].upper(): # check the user entered product name
with stock of store
p = True
while p == True:
try:
p_quantity = int(input("How many " + product + " do you want to buy: "))
p = False
except: # executes, if customer entered unexpected value.
print("\t\tError!!!\nPlease enter integer value!! ")
if product_name == L[0][0].upper() and p_quantity <= int(L[0][2]):
```

```
q[product_name] = p_quantity
elif product_name == L[1][0].upper() and p_quantity <= int(L[1][2]):
q[product_name] = p_quantity
elif product_name == L[2][0].upper() and p_quantity <= int(L[2][2]):
q[product_name] = p_quantity
else:
print(
"\nSorry!! " + a_name + "!, " + product + " is out of stock.\nWe will add stock of " + product + " later. \nLets hope, you will get this product after next shopping.\n")
flag = (input(a_name + " do you want buy more products?(Y/N)"))
else:
print("sorry!! " + product + " is not available in our store.")
print("\nChoose from following products please!")
print("-----------------------------------------")
print("PRODUCT\t\tPRICE\t\tQUANTITY")
print("-----------------------------------------")
for i in range(len(L)):
print(L[i][0], "\t\t", L[i][1], "\t\t",
L[i][2]) # print, last updated product name, quantity and price.
print("-----------------------------------------")
print("\nYou Choosed Items and it's Quantity respectively:\n", q, "\n")
'''
In the following operation:
1) change every string value in the upper case latter.
2) check what is the product entered by customer.
3) executes respective condition if product is phone or laptop or hdd entered by customer.
'''
f_amount = 0 # final amount
for keys in q.keys():
if keys == L[0][0].upper(): # executes this operation if product is phone entered by customer.
p_price = int(L[0][1])
p_num = int(q[keys])
p_amount = (p_price * p_num)
f_amount += (p_price * p_num)
print("\nTotal cost for phone: ", p_amount)
```

```
elif keys == L[1][0].upper(): # executes this operation if product is laptop entered by customer.
l_price = int(L[1][1])
l_num = int(q[keys])
l_amount = (l_price * l_num)
f_amount += (l_price * l_num)
print("Total cost for laptop: ", l_amount)
else: # executes this operation if product is hdd entered by customer.
h_price = int(L[2][1])
h_num = int(q[keys])
h_amount = (h_price * h_num)
f_amount += (h_price * h_num)
print("Total cost for HDD: ", h_amount)
print("\nYour discountable total amount is: ", f_amount)
'''
In the following operation:
1) ask to customer for expected discount % in total purchase amount.
2) check the total purchase amount which is dcustomer expected discountable or not.
3) check the total purchase amount which is basic discountable or not.
'''
disc = float(input("Please enter your expected discount (%): "))
dis = 0.0
if (f_amount >= 5000) and (f_amount < 10000):
discount = disc
if discount <= 5.0:
dis = (discount * f_amount) / 100
total = float(f_amount - dis)
print("You got your expected " + str(disc) + "% discount and amount is: ", dis)
else:
discount = 5.0
dis = (discount * f_amount) / 100
total = float(f_amount - dis)
print("You did not got your expected " + str(
disc) + "% discount\nBecause, your totel purchase is not meet the minimum criteria for " + str(
disc) + "% discount.")
```

```
print("You got basic 5% discount and discounted amount is:", dis)
elif f_amount >= 10000:
discount = disc
if discount <= 10.0:
dis = (discount * f_amount) / 100
total = float(f_amount - dis)
print("You got your expected " + str(disc) + "% discount and amount is: ",
dis)
else:
discount = 10.0
dis = (discount * f_amount) / 100
total = float(f_amount - dis)
print("You did not got your expected " + str(
disc) + "% discount\nBecause, your totel purchase is not meet the minimum
criteria for " + str(
disc) + "% discount.")
print("You got basic 10% discount and discounted amount is:", dis)
else:
discount = 0.0
total = float(f_amount)
print("You did not got your expected " + str(
disc) + "% discount\nBecause, your totel purchase is not meet the minimum
criteria for " + str(
disc) + "% discount.")
print("Your payable amount is: ", total)
'''
In the following operation:
1) write a each unique involve name which includes current date and time
with customer name as well.
2) write a purchase product name and details in file (invoice).
3) write a discounted amount and final payable amount in file (invoice).
'''

import datetime # import system date and time for create a unique invoive
name.
dt = str(datetime.datetime.now().year) + "-" +
str(datetime.datetime.now().month) + "-" + str(
datetime.datetime.now().day) + "-" + str(datetime.datetime.now().hour) + "-"
+ str(
datetime.datetime.now().minute) + "-" + str(datetime.datetime.now().second)
```

```
invoice = str(dt) # unique invoice
t = str(datetime.datetime.now().year) + "-" +
str(datetime.datetime.now().month) + "-" + str(
datetime.datetime.now().day) # date
d = str(t) # date
u = str(datetime.datetime.now().hour) + ":" +
str(datetime.datetime.now().minute) + ":" + str(
datetime.datetime.now().second) # time
e = str(u) # time
file = open(invoice + " (" + a_name + ").txt", "w") # generate a unique
invoive name and open it in write mode.
file.write("============================================================")
file.write("\nELECTRONIC STORE\t\t\t\tINVOICE")
file.write("\n\nInvoice: " + invoice + "\t\tDate: " + d + "\n\t\t\t\t\tTime:
" + e + "")
file.write("\nName of Customer: " + str(a_name) + "")
file.write("\n============================================================"
)
file.write("\nPARTICULAR\tQUANTITY\tUNIT PRICE\tTOTAL")
file.write("\n------------------------------------------------------------"
)
for keys in q.keys(): # In this loop, write in a file only those product
which is purchase by user.
if keys == "PHONE":
file.write(
str("\n" + str(keys) + " \t\t " + str(q['PHONE']) + " \t\t " + str(L[0][1])
+ " \t\t " + str(p_amount)))
elif keys == "LAPTOP":
file.write(str(
"\n" + str(keys) + " \t\t " + str(q['LAPTOP']) + " \t\t " + str(L[1][1]) + "
\t\t " + str(l_amount)))
else:
file.write(
str("\n" + str(keys) + " \t\t " + str(q['HDD']) + " \t\t " + str(L[2][1]) +
" \t\t " + str(h_amount)))
file.write("\n\n------------------------------------------------------------
-")
file.write("\n\t\t\tYour discountable amount: " + str(f_amount))
file.write("\n------------------------------------------------------------"
)
```

```
file.write("\n\t\t Your " + str(discount) + "% discounted amount is: " +
str(dis))
file.write("\n----------------------------------------------------------------"
)
file.write("\n\t\t\t Your payable amount is: " + str(total))
file.write("\n----------------------------------------------------------------"
)
file.write("\n\n\tThank You " + a_name + " for your shopping.\n\t\tSee you
again!")
file.write("\n================================================================"
)
file.close()
return q
```

Static Code Metrics:

Metric	Value
Lines of Code (LOC)	166
Comment Density	0.0
Cyclomatic Complexity	25
Maintainability Index	46.84452379757548
Halstead Metrics	
Unique Operators	10
Unique Operands	190
Total Operators	132
Total Operands	265
Volume	3034.6109073405655
Effort	21162.41816961184
Depth of Inheritance Tree (DIT)	0
Coupling Between Object classes (CBO)	0
Lack of Cohesion of Methods (LCOM)	0
Fan-in	90
Fan-out	90
Number of Methods (NOM)	0

McCall's Intermediate Quality Metrics:

Metric	Value
Modifiability	0.34
Testability	0.48
Reliability	0.5
Understandability	-0.27
Self-Descriptiveness	0.08
Reusability	1.0
Portability	0.73
Efficiency	0

GPT Analysis Report:

Code Improvement Suggestions

1. Function Structure

The `purchase` function is lengthy and complex, exhibiting a high cyclomatic complexity (25). It would benefit from being divided into smaller, more manageable functions that each handle a specific aspect of the purchase process. This encourages single responsibility and improves readability.

Refactoring Suggestion:

- Break down the function into several smaller functions:

- `get_customer_info()`: to handle customer interactions.

- `validate_product()`: to check if the product is available.

- `get_quantity()`: to manage quantity input from the user.

- `calculate_total()`: for calculating the total cost.

- `apply_discount()`: for processing discounts.

- `generate_invoice()`: to handle invoice creation.

2. Exception Handling

The current exception handling is broadly defined and does not specify the type of exception being caught. This can lead to masking other potential issues.

Improvement Suggestion:

- Instead of using a bare `except`, catch specific exceptions (e.g., `ValueError` when converting user input to an integer).

```
python
try:
p_quantity = int(input("How many " + product + " do you want to buy: "))
except ValueError:
```

```
print("\t\tError!!!\nPlease enter an integer value!!")
```

3. Magic Numbers and Strings

The code contains "magic numbers" like `5000`, `10000`, and hardcoded product names such as "PHONE". These should be replaced with constants for better maintainability and readability.

Refactoring Suggestion:

```
python
MIN_DISCOUNT_AMOUNT_1 = 5000
MIN_DISCOUNT_AMOUNT_2 = 10000
BASIC_DISCOUNT_1 = 5.0
BASIC_DISCOUNT_2 = 10.0
```

4. Improving User Input Handling

The existing user input does not provide clear feedback or validation upon incorrect entries, which could confuse users.

Improvement Suggestion:

- Create a loop that continues to prompt the user until valid input is received.

5. Use of Dictionaries

Instead of checking product availability using a series of `if-elif` statements, consider using a dictionary to map product names to their respective data.

Example:

```
python
products = {
"PHONE": L[0],
"LAPTOP": L[1],
"HDD": L[2]
}
```

This makes checking product information more concise and easier to extend in the future.

6. Comment Quality

The comments in the code lack clarity and are not helpful in explaining the logic of the code.

Improvement Suggestion:

- Replace comments with meaningful function and variable names.
- Provide docstrings for functions to clearly explain their purpose and usage.

7. File Handling

The file handling could be improved by using a context manager (`with` statement) to automatically handle file closing and prevent potential resource leaks.

Improvement Suggestion:

```
python
with open(invoice + " (" + a_name + ").txt", "w") as file:
# write to file
```

8. Inconsistent Use of Data Structures

The use of lists to represent products could lead to confusion and errors due to the order of elements. A better representation would be a list of dictionaries.

Refactoring Suggestion:

```
python
products = [
{"name": "PHONE", "price": 1000, "quantity": 50},
{"name": "LAPTOP", "price": 2000, "quantity": 30},
{"name": "HDD", "price": 300, "quantity": 100}
]
```

9. Testing and Validation

The current code lacks modular testing capabilities because it consists of a single monolithic function.

Improvement Suggestion:

- Implement unit tests for the individual functions once the code has been refactored. This will ensure each part works as expected and improve reliability.

10. Use of Enumerations

For products or categories, consider using Python's built-in `enum` module for better clarity and error prevention.

Example:

```
python
from enum import Enum

class Product(Enum):
PHONE = "PHONE"
LAPTOP = "LAPTOP"
HDD = "HDD"
```

Conclusion

By restructuring the `purchase` function into smaller, dedicated functions, employing clear error handling, improving input handling, and making use of data structures and constants, the code will become much more maintainable, readable, and testable. Each of these improvements addresses specific metrics, such as cyclomatic complexity and maintainability index, which were notably lacking in the original code.