

AI Assisted Coding

Assignment-8.1

Name:P Samanvith

HTNO:2303A52090

Batch:40

Task Description #1

(Password Strength Validator – Apply AI in Security Context)

- Task: Apply AI to generate at least 3 assert test cases for `is_strong_password(password)` and implement the validator function.
- Requirements:
 - o Password must have at least 8 characters.
 - o Must include uppercase, lowercase, digit, and special character.
 - o Must not contain spaces.

Example Assert Test Cases:

```
assert is_strong_password("Abcd@123") == True  
assert is_strong_password("abcd123") == False  
assert is_strong_password("ABCD@1234") == True
```

Code:

Task Description #2

(Number Classification with Loops – Apply AI for Edge Case Handling)

- Task: Use AI to generate at least 3 assert test cases for a `classify_number(n)` function. Implement using loops.

- Requirements:

- o Classify numbers as Positive, Negative, or Zero.
 - o Handle invalid inputs like strings and None.
 - o Include boundary conditions (-1, 0, 1).

Example Assert Test Cases:

```
assert classify_number(10) == "Positive"
```

```
assert classify_number(-5) == "Negative"
```

```
assert classify_number(0) == "Zero"
```

Code:

```

assignment_8.py > classify_number
7     return isinstance(p,str) and len(p)>=8 and ' ' not in p and any(c.isupper() for c in p)
8
9 # TASK 2: Number Classification | Req: Classify as Positive/Negative/Zero, handle invalid
10 # assert classify_number(10)=="Positive"; assert classify_number(-5)=="Negative"; assert classify_number(0)=="Zero"
11 def classify_number(n)->str:
12     if n is None or isinstance(n,(str,bool)):return "Invalid"
13     try:num=float(n) if not isinstance(n,(int,float)) else n
14     except:return "Invalid"
15     return "Positive" if num>0 else ("Negative" if num<0 else "Zero")
16
17 # TASK 3: Anagram Checker | Req: Ignore case/spaces/punctuation, handle edge cases
18 # assert is_anagram("listen","silent")==True; assert is_anagram("hello","world")==False
19 def is_anagram(s1:str,s2:str)->bool:
20     if not isinstance(s1,str) or not isinstance(s2,str):return False
21     c1=''.join(x.lower() for x in s1 if x.isalnum())
22     c2=''.join(x.lower() for x in s2 if x.isalnum())
23     return (len(c1)==0 and len(c2)==0) or (len(c1)>0 and len(c2)>0 and sorted(c1)==sorted(c2))

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

Enter your choice (1-7): 2
Enter a number: 3456789
Number classification: Positive

Enter your choice (1-7): 2
Enter a number: -23456
Number classification: Negative

Enter your choice (1-7): 2
Enter a number: 0000000000
Number classification: Zero

```

Task Description #3 (Anagram Checker – Apply AI for String Analysis)

- Task: Use AI to generate at least 3 assert test cases for `is_anagram(str1, str2)` and implement the function.

- Requirements:

- Ignore case, spaces, and punctuation.
- Handle edge cases (empty strings, identical words).

Example Assert Test Cases:

```
assert is_anagram("listen", "silent") == True
```

```
assert is_anagram("hello", "world") == False
```

```
assert is_anagram("Dormitory", "Dirty Room") == True
```

Code:

```
assignment_6.py > classify_number
17 # TASK 3: Anagram Checker | Req: Ignore case/spaces/punctuation, handle edge cases
18 # assert is_anagram("listen", "silent") == True; assert is_anagram("hello", "world") == False
19 def is_anagram(s1:str,s2:str)->bool:
20     if not isinstance(s1,str) or not isinstance(s2,str):return False
21     c1=''.join(x.lower() for x in s1 if x.isalnum())
22     c2=''.join(x.lower() for x in s2 if x.isalnum())
23     return (len(c1)==0 and len(c2)==0) or (len(c1)>0 and len(c2)>0 and sorted(c1)==sorted(c2))
24
25 # TASK 4: Inventory Class | Methods: add_item(name, qty), remove_item(name, qty), get_stock()
26 # inv.add_item("Pen",10); assert inv.get_stock("Pen")==10; inv.remove_item("Pen",5); assert
27 class Inventory:
28     def __init__(self):self.stock:Dict[str,int]={}
29     def add_item(self,n:str,q:int)->None:
30         if not isinstance(n,str) or not isinstance(q,int) or isinstance(q,bool):raise TypeError
31         if q<0:raise ValueError("Quantity cannot be negative")
32         self.stock[n]=self.stock.get(n,0)+q
33     def remove_item(self,n:str,q:int)->None:
34         if not n in self.stock:raise ValueError(f"Item {n} not found in stock")
35         if self.stock[n]<=q:raise ValueError(f"Cannot remove more than {self.stock[n]} units of item {n}")
36         self.stock[n]-=q
37
38     def get_stock(self,n:str)->int:
39         if not n in self.stock:raise ValueError(f"Item {n} not found in stock")
40         return self.stock[n]
41
42     def __str__(self):
43         return f"Inventory Stock: {self.stock}"
44
45     def __repr__(self):
46         return f"Inventory Stock: {self.stock}"
47
48     def __len__(self):
49         return len(self.stock)
50
51     def __iter__(self):
52         return iter(self.stock.items())
53
54     def __contains__(self, item):
55         return item in self.stock
56
57     def __eq__(self, other):
58         if not isinstance(other, Inventory):return False
59         return self.stock==other.stock
60
61     def __ne__(self, other):
62         if not isinstance(other, Inventory):return True
63         return self.stock!=other.stock
64
65     def __gt__(self, other):
66         if not isinstance(other, Inventory):return False
67         return self.stock>other.stock
68
69     def __ge__(self, other):
70         if not isinstance(other, Inventory):return False
71         return self.stock>=other.stock
72
73     def __lt__(self, other):
74         if not isinstance(other, Inventory):return True
75         return self.stock<other.stock
76
77     def __le__(self, other):
78         if not isinstance(other, Inventory):return True
79         return self.stock<=other.stock
80
81     def __add__(self, other):
82         if not isinstance(other, Inventory):return self
83         new_stock={**self.stock,**other.stock}
84         return Inventory(new_stock)
85
86     def __sub__(self, other):
87         if not isinstance(other, Inventory):return self
88         new_stock={k:v for k,v in self.stock.items() if k not in other.stock or v>other.stock[k]}
89         return Inventory(new_stock)
90
91     def __iadd__(self, other):
92         if not isinstance(other, Inventory):return self
93         self.stock={**self.stock,**other.stock}
94         return self
95
96     def __isub__(self, other):
97         if not isinstance(other, Inventory):return self
98         self.stock={k:v for k,v in self.stock.items() if k not in other.stock or v>other.stock[k]}
99         return self
100
101     def __copy__(self):
102         new_stock={k:v for k,v in self.stock.items()}
103         return Inventory(new_stock)
104
105     def __deepcopy__(self, memo):
106         new_stock={k:v for k,v in self.stock.items()}
107         return Inventory(new_stock)
108
109     def __hash__(self):
110         return hash(self.stock)
111
112     def __getstate__(self):
113         return self.stock
114
115     def __setstate__(self, state):
116         self.stock=state
117
118     def __getattribute__(self, name):
119         if name in ["__dict__", "__weakref__"]:
120             return super().__getattribute__(name)
121         return self.__dict__.get(name, None)
122
123     def __setattr__(self, name, value):
124         if name in ["__dict__", "__weakref__"]:
125             return super().__setattr__(name, value)
126         self.__dict__[name]=value
127
128     def __delattr__(self, name):
129         if name in ["__dict__", "__weakref__"]:
130             return super().__delattr__(name)
131         del self.__dict__[name]
132
133     def __dir__(self):
134         return dir(self.__dict__)
135
136     def __str__(self):
137         return f"Inventory Stock: {self.stock}"
138
139     def __repr__(self):
140         return f"Inventory Stock: {self.stock}"
141
142     def __len__(self):
143         return len(self.stock)
144
145     def __iter__(self):
146         return iter(self.stock.items())
147
148     def __contains__(self, item):
149         return item in self.stock
150
151     def __eq__(self, other):
152         if not isinstance(other, Inventory):return False
153         return self.stock==other.stock
154
155     def __ne__(self, other):
156         if not isinstance(other, Inventory):return True
157         return self.stock!=other.stock
158
159     def __gt__(self, other):
160         if not isinstance(other, Inventory):return False
161         return self.stock>other.stock
162
163     def __ge__(self, other):
164         if not isinstance(other, Inventory):return False
165         return self.stock>=other.stock
166
167     def __lt__(self, other):
168         if not isinstance(other, Inventory):return True
169         return self.stock<other.stock
170
171     def __le__(self, other):
172         if not isinstance(other, Inventory):return True
173         return self.stock<=other.stock
174
175     def __add__(self, other):
176         if not isinstance(other, Inventory):return self
177         new_stock={**self.stock,**other.stock}
178         return Inventory(new_stock)
179
180     def __sub__(self, other):
181         if not isinstance(other, Inventory):return self
182         new_stock={k:v for k,v in self.stock.items() if k not in other.stock or v>other.stock[k]}
183         return Inventory(new_stock)
184
185     def __iadd__(self, other):
186         if not isinstance(other, Inventory):return self
187         self.stock={**self.stock,**other.stock}
188         return self
189
190     def __isub__(self, other):
191         if not isinstance(other, Inventory):return self
192         self.stock={k:v for k,v in self.stock.items() if k not in other.stock or v>other.stock[k]}
193         return self
194
195     def __copy__(self):
196         new_stock={k:v for k,v in self.stock.items()}
197         return Inventory(new_stock)
198
199     def __deepcopy__(self, memo):
200         new_stock={k:v for k,v in self.stock.items()}
201         return Inventory(new_stock)
202
203     def __hash__(self):
204         return hash(self.stock)
205
206     def __getstate__(self):
207         return self.stock
208
209     def __setstate__(self, state):
210         self.stock=state
211
212     def __getattribute__(self, name):
213         if name in ["__dict__", "__weakref__"]:
214             return super().__getattribute__(name)
215         return self.__dict__.get(name, None)
216
217     def __setattr__(self, name, value):
218         if name in ["__dict__", "__weakref__"]:
219             return super().__setattr__(name, value)
220         self.__dict__[name]=value
221
222     def __delattr__(self, name):
223         if name in ["__dict__", "__weakref__"]:
224             return super().__delattr__(name)
225         del self.__dict__[name]
226
227     def __dir__(self):
228         return dir(self.__dict__)
229
230     def __str__(self):
231         return f"Inventory Stock: {self.stock}"
232
233     def __repr__(self):
234         return f"Inventory Stock: {self.stock}"
235
236     def __len__(self):
237         return len(self.stock)
238
239     def __iter__(self):
240         return iter(self.stock.items())
241
242     def __contains__(self, item):
243         return item in self.stock
244
245     def __eq__(self, other):
246         if not isinstance(other, Inventory):return False
247         return self.stock==other.stock
248
249     def __ne__(self, other):
250         if not isinstance(other, Inventory):return True
251         return self.stock!=other.stock
252
253     def __gt__(self, other):
254         if not isinstance(other, Inventory):return False
255         return self.stock>other.stock
256
257     def __ge__(self, other):
258         if not isinstance(other, Inventory):return False
259         return self.stock>=other.stock
260
261     def __lt__(self, other):
262         if not isinstance(other, Inventory):return True
263         return self.stock<other.stock
264
265     def __le__(self, other):
266         if not isinstance(other, Inventory):return True
267         return self.stock<=other.stock
268
269     def __add__(self, other):
270         if not isinstance(other, Inventory):return self
271         new_stock={**self.stock,**other.stock}
272         return Inventory(new_stock)
273
274     def __sub__(self, other):
275         if not isinstance(other, Inventory):return self
276         new_stock={k:v for k,v in self.stock.items() if k not in other.stock or v>other.stock[k]}
277         return Inventory(new_stock)
278
279     def __iadd__(self, other):
280         if not isinstance(other, Inventory):return self
281         self.stock={**self.stock,**other.stock}
282         return self
283
284     def __isub__(self, other):
285         if not isinstance(other, Inventory):return self
286         self.stock={k:v for k,v in self.stock.items() if k not in other.stock or v>other.stock[k]}
287         return self
288
289     def __copy__(self):
290         new_stock={k:v for k,v in self.stock.items()}
291         return Inventory(new_stock)
292
293     def __deepcopy__(self, memo):
294         new_stock={k:v for k,v in self.stock.items()}
295         return Inventory(new_stock)
296
297     def __hash__(self):
298         return hash(self.stock)
299
300     def __getstate__(self):
301         return self.stock
302
303     def __setstate__(self, state):
304         self.stock=state
305
306     def __getattribute__(self, name):
307         if name in ["__dict__", "__weakref__"]:
308             return super().__getattribute__(name)
309         return self.__dict__.get(name, None)
310
311     def __setattr__(self, name, value):
312         if name in ["__dict__", "__weakref__"]:
313             return super().__setattr__(name, value)
314         self.__dict__[name]=value
315
316     def __delattr__(self, name):
317         if name in ["__dict__", "__weakref__"]:
318             return super().__delattr__(name)
319         del self.__dict__[name]
320
321     def __dir__(self):
322         return dir(self.__dict__)
323
324     def __str__(self):
325         return f"Inventory Stock: {self.stock}"
326
327     def __repr__(self):
328         return f"Inventory Stock: {self.stock}"
329
330     def __len__(self):
331         return len(self.stock)
332
333     def __iter__(self):
334         return iter(self.stock.items())
335
336     def __contains__(self, item):
337         return item in self.stock
338
339     def __eq__(self, other):
340         if not isinstance(other, Inventory):return False
341         return self.stock==other.stock
342
343     def __ne__(self, other):
344         if not isinstance(other, Inventory):return True
345         return self.stock!=other.stock
346
347     def __gt__(self, other):
348         if not isinstance(other, Inventory):return False
349         return self.stock>other.stock
350
351     def __ge__(self, other):
352         if not isinstance(other, Inventory):return False
353         return self.stock>=other.stock
354
355     def __lt__(self, other):
356         if not isinstance(other, Inventory):return True
357         return self.stock<other.stock
358
359     def __le__(self, other):
360         if not isinstance(other, Inventory):return True
361         return self.stock<=other.stock
362
363     def __add__(self, other):
364         if not isinstance(other, Inventory):return self
365         new_stock={**self.stock,**other.stock}
366         return Inventory(new_stock)
367
368     def __sub__(self, other):
369         if not isinstance(other, Inventory):return self
370         new_stock={k:v for k,v in self.stock.items() if k not in other.stock or v>other.stock[k]}
371         return Inventory(new_stock)
372
373     def __iadd__(self, other):
374         if not isinstance(other, Inventory):return self
375         self.stock={**self.stock,**other.stock}
376         return self
377
378     def __isub__(self, other):
379         if not isinstance(other, Inventory):return self
380         self.stock={k:v for k,v in self.stock.items() if k not in other.stock or v>other.stock[k]}
381         return self
382
383     def __copy__(self):
384         new_stock={k:v for k,v in self.stock.items()}
385         return Inventory(new_stock)
386
387     def __deepcopy__(self, memo):
388         new_stock={k:v for k,v in self.stock.items()}
389         return Inventory(new_stock)
390
391     def __hash__(self):
392         return hash(self.stock)
393
394     def __getstate__(self):
395         return self.stock
396
397     def __setstate__(self, state):
398         self.stock=state
399
400     def __getattribute__(self, name):
401         if name in ["__dict__", "__weakref__"]:
402             return super().__getattribute__(name)
403         return self.__dict__.get(name, None)
404
405     def __setattr__(self, name, value):
406         if name in ["__dict__", "__weakref__"]:
407             return super().__setattr__(name, value)
408         self.__dict__[name]=value
409
410     def __delattr__(self, name):
411         if name in ["__dict__", "__weakref__"]:
412             return super().__delattr__(name)
413         del self.__dict__[name]
414
415     def __dir__(self):
416         return dir(self.__dict__)
417
418     def __str__(self):
419         return f"Inventory Stock: {self.stock}"
420
421     def __repr__(self):
422         return f"Inventory Stock: {self.stock}"
423
424     def __len__(self):
425         return len(self.stock)
426
427     def __iter__(self):
428         return iter(self.stock.items())
429
430     def __contains__(self, item):
431         return item in self.stock
432
433     def __eq__(self, other):
434         if not isinstance(other, Inventory):return False
435         return self.stock==other.stock
436
437     def __ne__(self, other):
438         if not isinstance(other, Inventory):return True
439         return self.stock!=other.stock
440
441     def __gt__(self, other):
442         if not isinstance(other, Inventory):return False
443         return self.stock>other.stock
444
445     def __ge__(self, other):
446         if not isinstance(other, Inventory):return False
447         return self.stock>=other.stock
448
449     def __lt__(self, other):
450         if not isinstance(other, Inventory):return True
451         return self.stock<other.stock
452
453     def __le__(self, other):
454         if not isinstance(other, Inventory):return True
455         return self.stock<=other.stock
456
457     def __add__(self, other):
458         if not isinstance(other, Inventory):return self
459         new_stock={**self.stock,**other.stock}
460         return Inventory(new_stock)
461
462     def __sub__(self, other):
463         if not isinstance(other, Inventory):return self
464         new_stock={k:v for k,v in self.stock.items() if k not in other.stock or v>other.stock[k]}
465         return Inventory(new_stock)
466
467     def __iadd__(self, other):
468         if not isinstance(other, Inventory):return self
469         self.stock={**self.stock,**other.stock}
470         return self
471
472     def __isub__(self, other):
473         if not isinstance(other, Inventory):return self
474         self.stock={k:v for k,v in self.stock.items() if k not in other.stock or v>other.stock[k]}
475         return self
476
477     def __copy__(self):
478         new_stock={k:v for k,v in self.stock.items()}
479         return Inventory(new_stock)
480
481     def __deepcopy__(self, memo):
482         new_stock={k:v for k,v in self.stock.items()}
483         return Inventory(new_stock)
484
485     def __hash__(self):
486         return hash(self.stock)
487
488     def __getstate__(self):
489         return self.stock
490
491     def __setstate__(self, state):
492         self.stock=state
493
494     def __getattribute__(self, name):
495         if name in ["__dict__", "__weakref__"]:
496             return super().__getattribute__(name)
497         return self.__dict__.get(name, None)
498
499     def __setattr__(self, name, value):
500         if name in ["__dict__", "__weakref__"]:
501             return super().__setattr__(name, value)
502         self.__dict__[name]=value
503
504     def __delattr__(self, name):
505         if name in ["__dict__", "__weakref__"]:
506             return super().__delattr__(name)
507         del self.__dict__[name]
508
509     def __dir__(self):
510         return dir(self.__dict__)
511
512     def __str__(self):
513         return f"Inventory Stock: {self.stock}"
514
515     def __repr__(self):
516         return f"Inventory Stock: {self.stock}"
517
518     def __len__(self):
519         return len(self.stock)
520
521     def __iter__(self):
522         return iter(self.stock.items())
523
524     def __contains__(self, item):
525         return item in self.stock
526
527     def __eq__(self, other):
528         if not isinstance(other, Inventory):return False
529         return self.stock==other.stock
530
531     def __ne__(self, other):
532         if not isinstance(other, Inventory):return True
533         return self.stock!=other.stock
534
535     def __gt__(self, other):
536         if not isinstance(other, Inventory):return False
537         return self.stock>other.stock
538
539     def __ge__(self, other):
540         if not isinstance(other, Inventory):return False
541         return self.stock>=other.stock
542
543     def __lt__(self, other):
544         if not isinstance(other, Inventory):return True
545         return self.stock<other.stock
546
547     def __le__(self, other):
548         if not isinstance(other, Inventory):return True
549         return self.stock<=other.stock
550
551     def __add__(self, other):
552         if not isinstance(other, Inventory):return self
553         new_stock={**self.stock,**other.stock}
554         return Inventory(new_stock)
555
556     def __sub__(self, other):
557         if not isinstance(other, Inventory):return self
558         new_stock={k:v for k,v in self.stock.items() if k not in other.stock or v>other.stock[k]}
559         return Inventory(new_stock)
560
561     def __iadd__(self, other):
562         if not isinstance(other, Inventory):return self
563         self.stock={**self.stock,**other.stock}
564         return self
565
566     def __isub__(self, other):
567         if not isinstance(other, Inventory):return self
568         self.stock={k:v for k,v in self.stock.items() if k not in other.stock or v>other.stock[k]}
569         return self
570
571     def __copy__(self):
572         new_stock={k:v for k,v in self.stock.items()}
573         return Inventory(new_stock)
574
575     def __deepcopy__(self, memo):
576         new_stock={k:v for k,v in self.stock.items()}
577         return Inventory(new_stock)
578
579     def __hash__(self):
580         return hash(self.stock)
581
582     def __getstate__(self):
583         return self.stock
584
585     def __setstate__(self, state):
586         self.stock=state
587
588     def __getattribute__(self, name):
589         if name in ["__dict__", "__weakref__"]:
590             return super().__getattribute__(name)
591         return self.__dict__.get(name, None)
592
593     def __setattr__(self, name, value):
594         if name in ["__dict__", "__weakref__"]:
595             return super().__setattr__(name, value)
596         self.__dict__[name]=value
597
598     def __delattr__(self, name):
599         if name in ["__dict__", "__weakref__"]:
600             return super().__delattr__(name)
601         del self.__dict__[name]
602
603     def __dir__(self):
604         return dir(self.__dict__)
605
606     def __str__(self):
607         return f"Inventory Stock: {self.stock}"
608
609     def __repr__(self):
610         return f"Inventory Stock: {self.stock}"
```

Task Description #4

(Inventory Class – Apply AI to Simulate Real- World Inventory System)

- Task: Ask AI to generate at least 3 assert-based tests for an Inventory class with stock management.
- Methods:
 - o add_item(name, quantity)
 - o remove_item(name, quantity)
 - o get_stock(name)

Example Assert Test Cases:

```
inv = Inventory()
inv.add_item("Pen", 10)
assert inv.get_stock("Pen") == 10
inv.remove_item("Pen", 5)
assert inv.get_stock("Pen") == 5
inv.add_item("Book", 3)
assert inv.get_stock("Book") == 3
```

Code:

The screenshot shows a Python code editor with a file named `assignment_8.py`. The code defines a `Inventory` class with methods for adding, removing, and getting stock levels. It includes validation logic for item names and quantities. Below the code editor is a terminal window showing the execution of the script and its output. The terminal shows the creation of an `Inventory` object, adding rice to the stock, removing some rice, and then checking the stock level.

```
8.py 27  class Inventory:
DDING 28      def __init__(self):self.stock:Dict[str,int]={}
1.docx 29          def add_item(self,n:str,q:int)->None:
1.docx 30              if not isinstance(n,str) or not isinstance(q,int) or isinstance(q,bool):raise TypeEr
y 31                  if q<0:raise ValueError("Quantity cannot be negative")
32                      self.stock[n]=self.stock.get(n,0)+q
33          def remove_item(self,n:str,q:int)->None:
34              if not isinstance(n,str) or not isinstance(q,int) or isinstance(q,bool):raise TypeEr
35                  if q<0:raise ValueError("Quantity cannot be negative")
36                  if n not in self.stock:raise KeyError(f"Item '{n}' not found")
37                  if self.stock[n]<q:raise ValueError("Insufficient stock")
38                      self.stock[n]-=q
39                      if self.stock[n]==0:del self.stock[n]
40          def get_stock(self,n:str)->int:return self.stock.get(n,0) if isinstance(n,str) else (_ f
41
42 # TASK 5: Date Validation & Formatting | Req: Validate MM/DD/YYYY, convert to YYYY-MM-DD
43 # Insert validate and format date("10/10/2023")-->"2023-10-10"; assert validate_and_format_da
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS + × ... | C
PS C:\Users\srajan\OneDrive\Desktop\AI_assisstant_coding> & C:/Users/srajan/AppData/Local/Programs/Pyt
thon/Python313/python.exe c:/Users/srajan/OneDrive/Desktop/AI_assisstant_coding/assignment_8.py
t:
Inventory Manager | Commands: add, remove, check, quit
Command: add
Item name: rice
Quantity: 32
Added 32 rice(s)
Command: remove
Item name: rice
Quantity: 21
Removed 21 rice(s)
Command: check
Item name: rice
Stock of 'rice': 11
```

Task Description #5

(Date Validation & Formatting – Apply AI for Data Validation)

- Task: Use AI to generate at least 3 assert test cases for `validate_and_format_date(date_str)` to check and convert dates.

- Requirements:

- Validate "MM/DD/YYYY" format.
- Handle invalid dates.
- Convert valid dates to "YYYY-MM-DD".

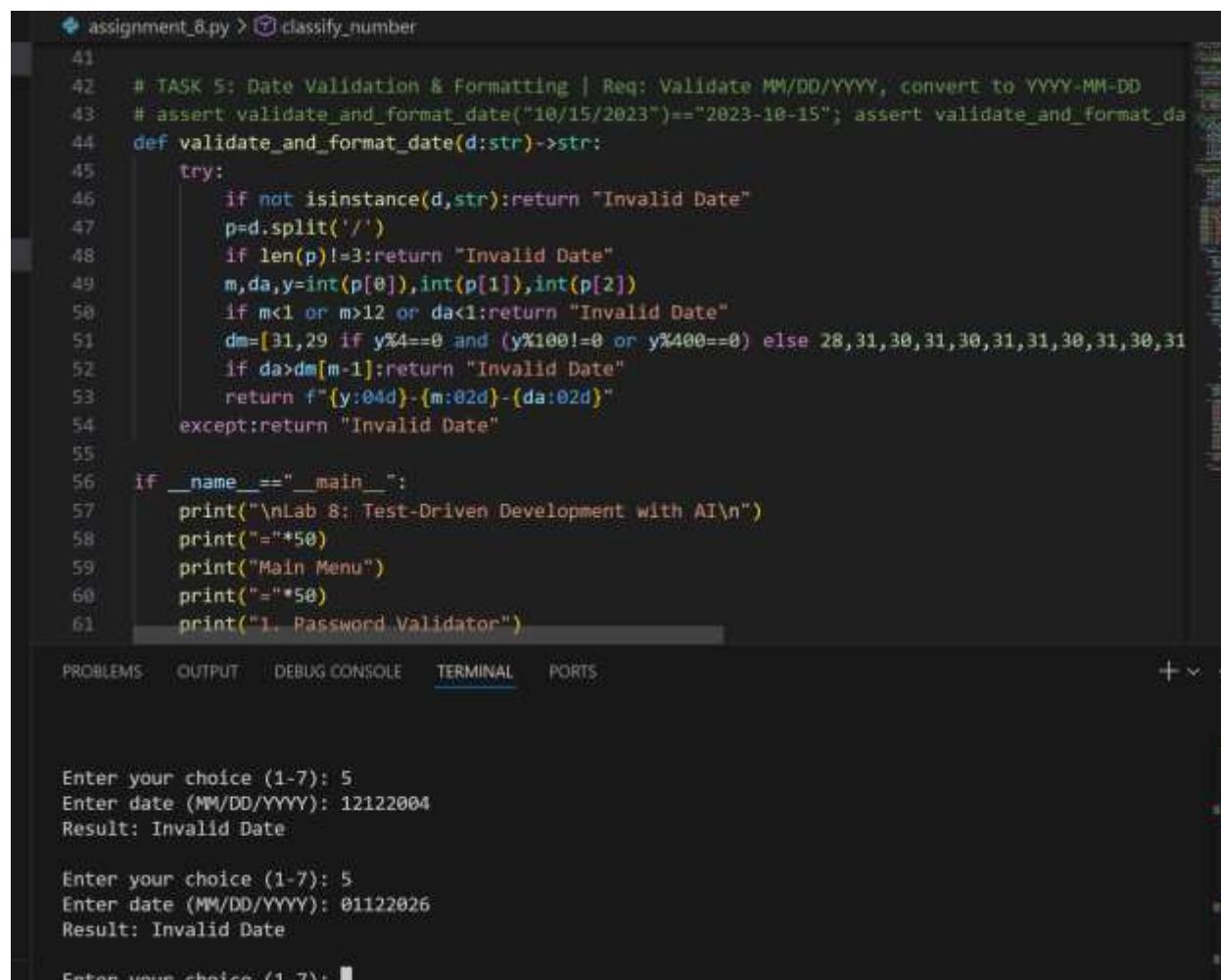
Example Assert Test Cases:

```
assert validate_and_format_date("10/15/2023") == "2023-10-15"
```

```
assert validate_and_format_date("02/30/2023") == "Invalid Date"
```

```
assert validate_and_format_date("01/01/2024") == "2024-01-01"
```

Code:



The screenshot shows a code editor window with a dark theme. The file is named `assignment_8.py`. The code implements a function `validate_and_format_date` that takes a date string in "MM/DD/YYYY" format and returns it in "YYYY-MM-DD" format. It handles invalid dates by returning "Invalid Date". The code uses a try-except block to catch any errors from the `int` conversions. It also includes a main menu loop for choice 5, which prints the date validation logic. The terminal below shows two test runs where the user enters choice 5 and a date, resulting in "Invalid Date" output.

```
assignment_8.py > classify_number
41
42 # TASK 5: Date Validation & Formatting | Req: Validate MM/DD/YYYY, convert to YYYY-MM-DD
43 # assert validate_and_format_date("10/15/2023")=="2023-10-15"; assert validate_and_format_da
44 def validate_and_format_date(d:str)->str:
45     try:
46         if not isinstance(d,str):return "Invalid Date"
47         p=d.split('/')
48         if len(p)!=3: return "Invalid Date"
49         m,da,y=int(p[0]),int(p[1]),int(p[2])
50         if m<1 or m>12 or da<1:return "Invalid Date"
51         dm=[31,29 if y%4==0 and (y%100!=0 or y%400==0) else 28,31,30,31,30,31,31,30,31,30,31
52         if da>dm[m-1]:return "Invalid Date"
53         return f"[{y:04d}]-{m:02d}-{da:02d}"
54     except: return "Invalid Date"
55
56 if __name__=="__main__":
57     print("\nLab 8: Test-Driven Development with AI\n")
58     print("*"*50)
59     print("Main Menu")
60     print("*"*50)
61     print("1. Password Validator")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
Enter your choice (1-7): 5
Enter date (MM/DD/YYYY): 12122004
Result: Invalid Date

Enter your choice (1-7): 5
Enter date (MM/DD/YYYY): 01122026
Result: Invalid Date

Enter your choice (1-7):
```