

UNIVERSIDAD NACIONAL DE SAN AGUSTÍN DE AREQUIPA

ESCUELA PROFESIONAL DE CIENCIA DE LA COMPUTACIÓN
ALGORITMOS Y ESTRUCTURAS DE DATOS



Laboratorio Nro. 07

Presentado por:

Fiorela Villarroel Ramos

Docente :

Yessenia Deysi Yari Ramos



Refactoring y SonarLint

1. Competencia del Laboratorio

- Mejorar el entendimiento, la extensibilidad y reutilización de un proyecto de software (Clean Code)

2. Equipos y Materiales

- Proyecto de software (No HTTP, no base de datos, ni aleatoriedad. Solo pura lógica... ¡enredada!): <https://github.com/emilybache/GildedRose-Refactoring-Kata>
- IDE (VS Code, Eclipse, IntelliJ IDEA, ...)
- SonarLint (Extensión de IDE escogido): <https://www.sonarsource.com/products/sonarlint/>
- Java 11 o superior
- **Grupos:** Individual

3. Actividad

- Verificar que su IDE tenga soporte al lenguaje de programación elegido (C++, C, Java, Python, Kotlin, Dart, Go o PHP) para este ejercicio.
- Instalar SonarLint como extensión de su IDE: <https://www.sonarsource.com/products/sonarlint/>
- Reiniciar su IDE.
- Clonar el proyecto de software a ser refactorizado: <https://github.com/emilybache/GildedRose-Refactoring-Kata>
- Abrir el proyecto de software a ser refactorizado: El código es desordenado, pero es 'solo' un montón de condicionales anidados. Seguramente se puede mejorar, nada abrumador.
- Construir el proyecto: Si todo va bien, podrá ver los resultados de la compilación y los warnings en la ventana View/Problems.
- Adicionar una nueva funcionalidad (feature) a un código base existente (objetivo):

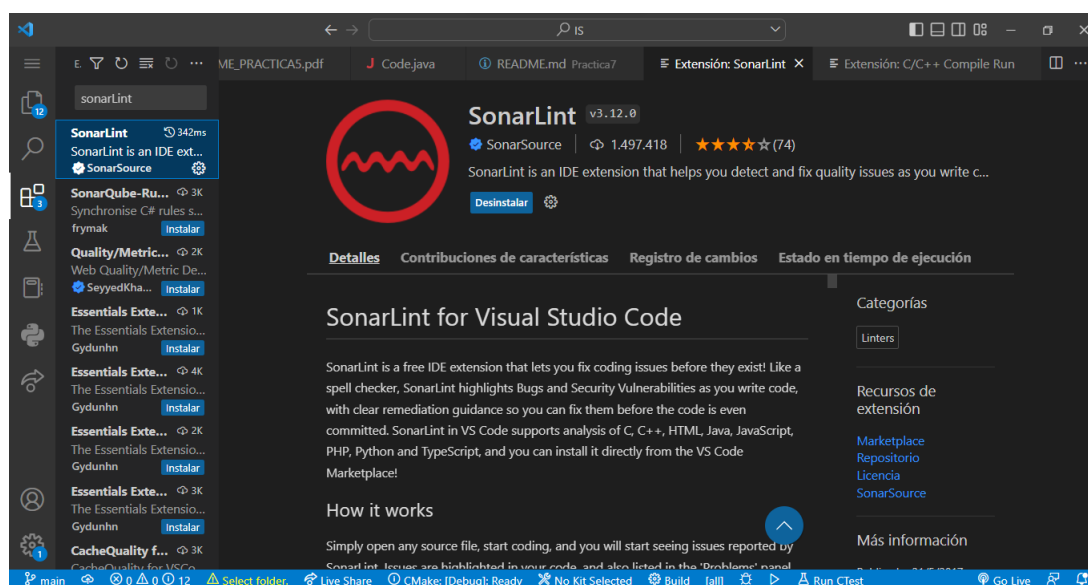
4. Desarrollo

4.1. Pre - requisitos

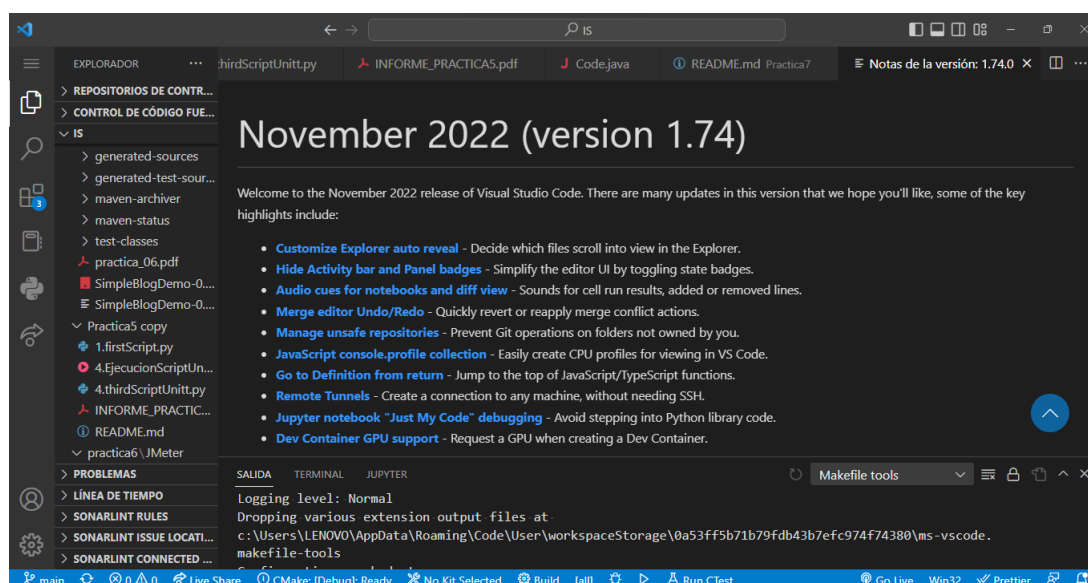
4.1.1. Verificar soporte del lenguaje python en IDE:



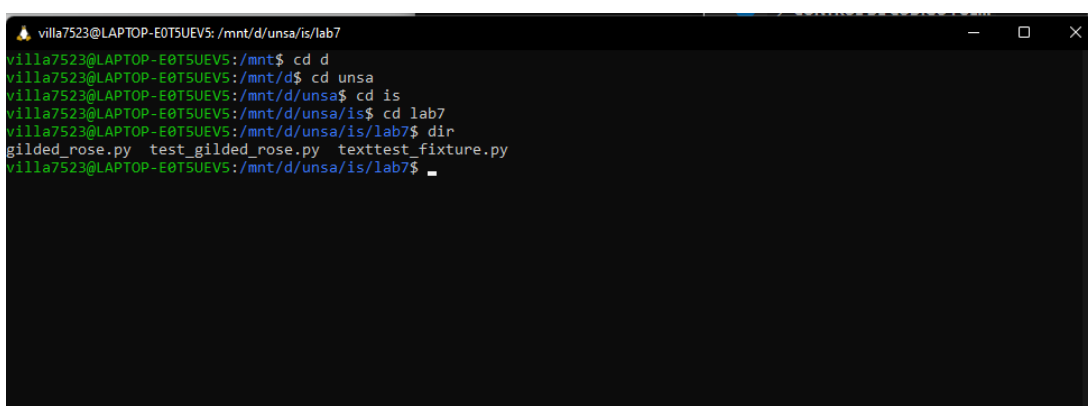
4.1.2. Instalar SonarLint como extensión del IDE



4.1.3. Reiniciar IDE



4.1.4. Clonar el proyecto de software a ser refactorizado



4.2. Refactorizando

4.2.1. Entender los requisitos de software:

4.2.2. Identificar Code Smells (*archivo por archivo*)

▼ PROBLEMAS				Filtro (por ejemplo, texto, ...)		
	Código	Mensaje	Archivo	Origen		
⚠	sonarlint(java:S10...	Replace this use of System.out or System.err by a logger.	lab7\Refactoring\src\App.java [Lín. ...	sonarlint		
💡	sonarlint(python:...	Refactor this function to reduce its Cognitive Complexity from 69 to the 15 allow...	lab7\gilded_rose.py [Lín. 8, col. 9]	sonarlint		
⚠	sonarlint(python:...	Define a constant instead of duplicating this literal "Backstage passes to a TAFKA...	lab7\gilded_rose.py [Lín. 10, col. 58]	sonarlint		
⚠	sonarlint(python:...	Define a constant instead of duplicating this literal "Sulfuras, Hand of Ragnaros" ...	lab7\gilded_rose.py [Lín. 12, col. 37]	sonarlint		
⚠	sonarlint(python:...	Merge this if statement with the enclosing one. [+1 location]	lab7\gilded_rose.py [Lín. 19, col. 29]	sonarlint		
⚠	sonarlint(python:...	Merge this if statement with the enclosing one. [+1 location]	lab7\gilded_rose.py [Lín. 22, col. 29]	sonarlint		
⚠	sonarlint(python:...	Correct one of the identical sub-expressions on both sides of operator "-". [+1 lo...	lab7\gilded_rose.py [Lín. 33, col. 55]	sonarlint		
⚠	sonarlint(python:...	Import only needed names or import the module and then use its members.	lab7\texttest_fixture.py [Lín. 4, col. 1]	sonarlint		
⚠	sonarlint(python:...	Define a constant instead of duplicating this literal "Backstage passes to a TAFKA...	lab7\texttest_fixture.py [Lín. 14, col. ...]	sonarlint		
⚠	-	Incorrect type. Expected "null".	C:\Users\LENOVO\AppData\Roamin...	_generated_di...		
⚠	-	Incorrect type. Expected "object".	C:\Users\LENOVO\AppData\Roamin...	_generated_di...		
🔗	sonarlint(java:S12...	Move this file to a named package.	lab7\Refactoring\src\App.java [Lín. ...	sonarlint		
🔗	sonarlint(java:S11...	Remove the declaration of thrown exception 'java.lang.Exception', as it cannot b...	lab7\Refactoring\src\App.java [Lín. ...	sonarlint		

4.3. Refactorizar

4.3.1. Corregir refactorización

4.3.1.1. issue S1192:

- **Descripción:** Las cadenas duplicadas hacen que el proceso de refactorización sea propenso a errores, ya que debe asegurarse de actualizar todas las ocurrencias. Por otro lado, las constantes pueden referenciarse desde muchos lugares, pero solo necesitan actualizarse en un solo lugar.
- **Excepciones** Para evitar generar algunos falsos positivos, se excluyen los literales que tienen menos de 5 caracteres.
- **Solución:** Defina una constante en lugar de duplicar este literal "cadena" número veces.

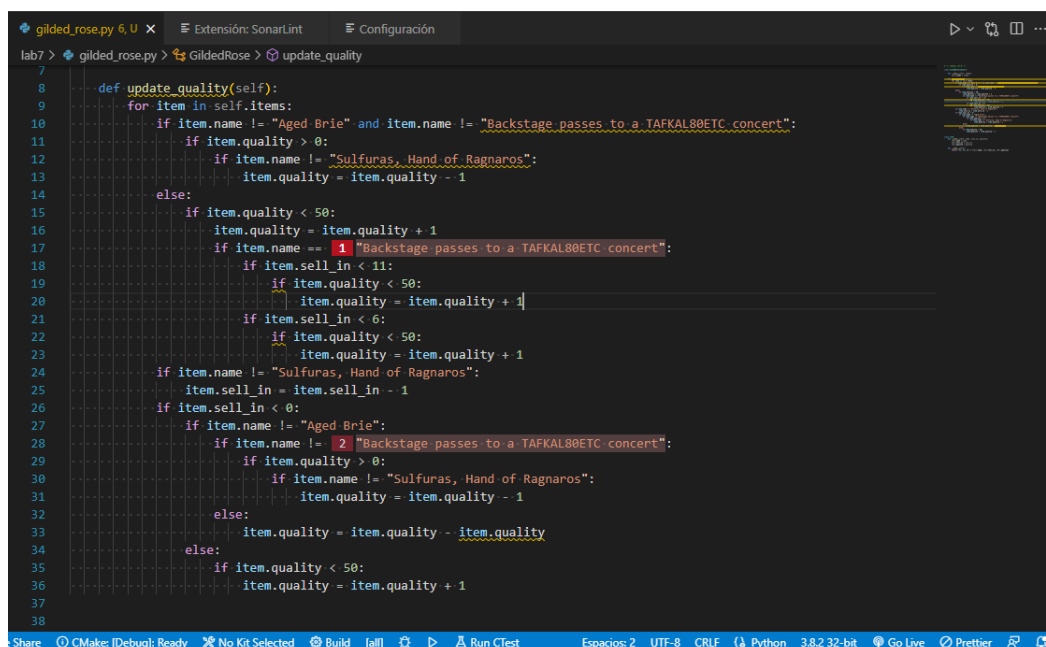
String literals should not be duplicated

Analyze your code

🔗 Code Smell 🔴 Critical ? 🛠 design

Caso I: Define a constant instead of duplicating this literal 'Backstage passes to a TAFKAL80ETC concert' 3 times. [+2 locations] sonarlint (python:S1192)

Before

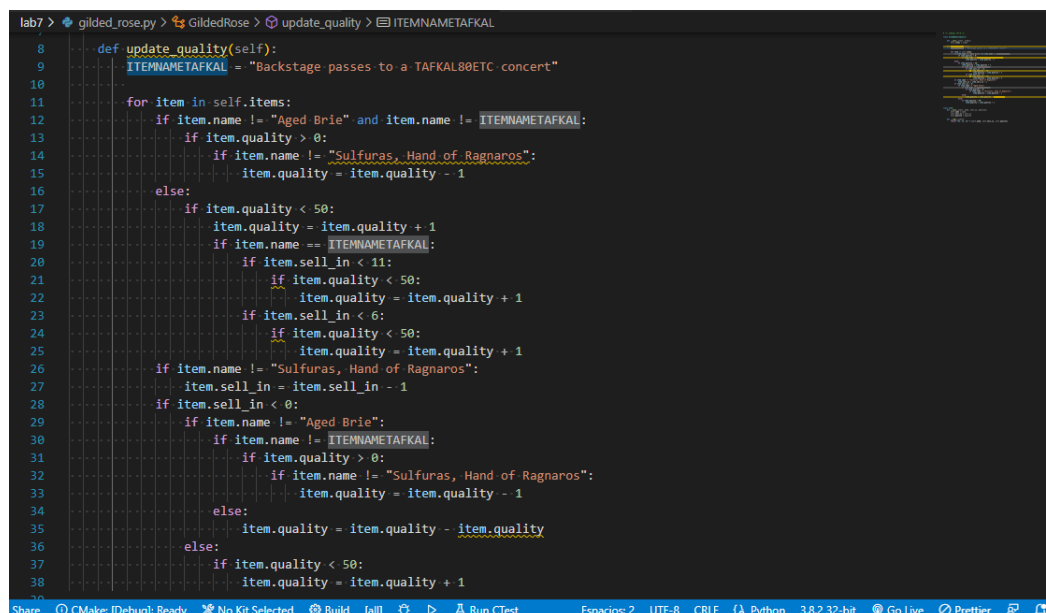


```

7
8 def update_quality(self):
9     for item in self.items:
10         if item.name != "Aged Brie" and item.name != "Backstage passes to a TAFKAL80ETC concert":
11             if item.quality > 0:
12                 if item.name != "Sulfuras, Hand of Ragnaros":
13                     item.quality = item.quality - 1
14             else:
15                 if item.quality < 50:
16                     item.quality = item.quality + 1
17                 if item.name == "Backstage passes to a TAFKAL80ETC concert":
18                     if item.sell_in < 11:
19                         if item.quality < 50:
20                             item.quality = item.quality + 1
21                     if item.sell_in < 6:
22                         if item.quality < 50:
23                             item.quality = item.quality + 1
24             if item.name != "Sulfuras, Hand of Ragnaros":
25                 item.sell_in = item.sell_in - 1
26             if item.sell_in < 0:
27                 if item.name != "Aged Brie":
28                     if item.name != "Backstage passes to a TAFKAL80ETC concert":
29                         if item.quality > 0:
30                             if item.name != "Sulfuras, Hand of Ragnaros":
31                                 item.quality = item.quality - 1
32                             else:
33                                 item.quality = item.quality - item.quality
34                         else:
35                             if item.quality < 50:
36                                 item.quality = item.quality + 1
37
38

```

After



```

lab7 > gilded_rose.py > GildedRose > update_quality > ITEMNAMETAFKAL
8 def update_quality(self):
9     ITEMNAMETAFKAL = "Backstage passes to a TAFKAL80ETC concert"
10
11     for item in self.items:
12         if item.name != "Aged Brie" and item.name != ITEMNAMETAFKAL:
13             if item.quality > 0:
14                 if item.name != "Sulfuras, Hand of Ragnaros":
15                     item.quality = item.quality - 1
16             else:
17                 if item.quality < 50:
18                     item.quality = item.quality + 1
19                 if item.name == ITEMNAMETAFKAL:
20                     if item.sell_in < 11:
21                         if item.quality < 50:
22                             item.quality = item.quality + 1
23                     if item.sell_in < 6:
24                         if item.quality < 50:
25                             item.quality = item.quality + 1
26             if item.name != "Sulfuras, Hand of Ragnaros":
27                 item.sell_in = item.sell_in - 1
28             if item.sell_in < 0:
29                 if item.name != "Aged Brie":
30                     if item.name != ITEMNAMETAFKAL:
31                         if item.quality > 0:
32                             if item.name != "Sulfuras, Hand of Ragnaros":
33                                 item.quality = item.quality - 1
34                             else:
35                                 item.quality = item.quality - item.quality
36                         else:
37                             if item.quality < 50:
38                                 item.quality = item.quality + 1
39
40

```

Caso II: Define a constant instead of duplicating this literal 'Sulfuras, Hand of Ragnaros' 3 times. [+2 locations] sonarlint (python:S1192)

Before

```
lab7 > gilded_rose.py > GildedRose > update_quality
8 def update_quality(self):
9     ITEMNAMETAFKAL = "Backstage passes to a TAFKAL88ETC concert"
10
11     for item in self.items:
12         if item.name != "Aged Brie" and item.name != ITEMNAMETAFKAL:
13             if item.quality > 0:
14                 if item.name != "Sulfuras, Hand of Ragnaros":
15                     item.quality = item.quality - 1
16             else:
17                 if item.quality < 50:
18                     item.quality = item.quality + 1
19                 if item.name == ITEMNAMETAFKAL:
20                     if item.sell_in < 11:
21                         if item.quality < 50:
22                             item.quality = item.quality + 1
23                     if item.sell_in < 6:
24                         if item.quality < 50:
25                             item.quality = item.quality + 1
26                 if item.name != 1 "Sulfuras, Hand of Ragnaros":
27                     item.sell_in = item.sell_in - 1
28                 if item.sell_in < 0:
29                     if item.name != "Aged Brie":
30                         if item.name != ITEMNAMETAFKAL:
31                             if item.quality > 0:
32                                 if item.name != 2 "Sulfuras, Hand of Ragnaros":
33                                     item.quality = item.quality - 1
34                             else:
35                                 item.quality = item.quality - item.quality
36                         else:
37                             if item.quality < 50:
38                                 item.quality = item.quality + 1
```

After

```
lab7 > gilded_rose.py > GildedRose > update_quality
8 def update_quality(self):
9     ITEMNAMETAFKAL = "Backstage passes to a TAFKAL88ETC concert"
10     ITEMNAMESULFURAS = "Sulfuras, Hand of Ragnaros"
11
12     for item in self.items:
13         if item.name != "Aged Brie" and item.name != ITEMNAMETAFKAL:
14             if item.quality > 0:
15                 if item.name != ITEMNAMESULFURAS:
16                     item.quality = item.quality - 1
17             else:
18                 if item.quality < 50:
19                     item.quality = item.quality + 1
20                 if item.name == ITEMNAMETAFKAL:
21                     if item.sell_in < 11:
22                         if item.quality < 50:
23                             item.quality = item.quality + 1
24                     if item.sell_in < 6:
25                         if item.quality < 50:
26                             item.quality = item.quality + 1
27                 if item.name != ITEMNAMESULFURAS:
28                     item.sell_in = item.sell_in - 1
29                 if item.sell_in < 0:
30                     if item.name != "Aged Brie":
31                         if item.name != ITEMNAMETAFKAL:
32                             if item.quality > 0:
33                                 if item.name != ITEMNAMESULFURAS:
34                                     item.quality = item.quality - 1
35                             else:
36                                 item.quality = item.quality - item.quality
37                         else:
38                             if item.quality < 50:
39                                 item.quality = item.quality + 1
```

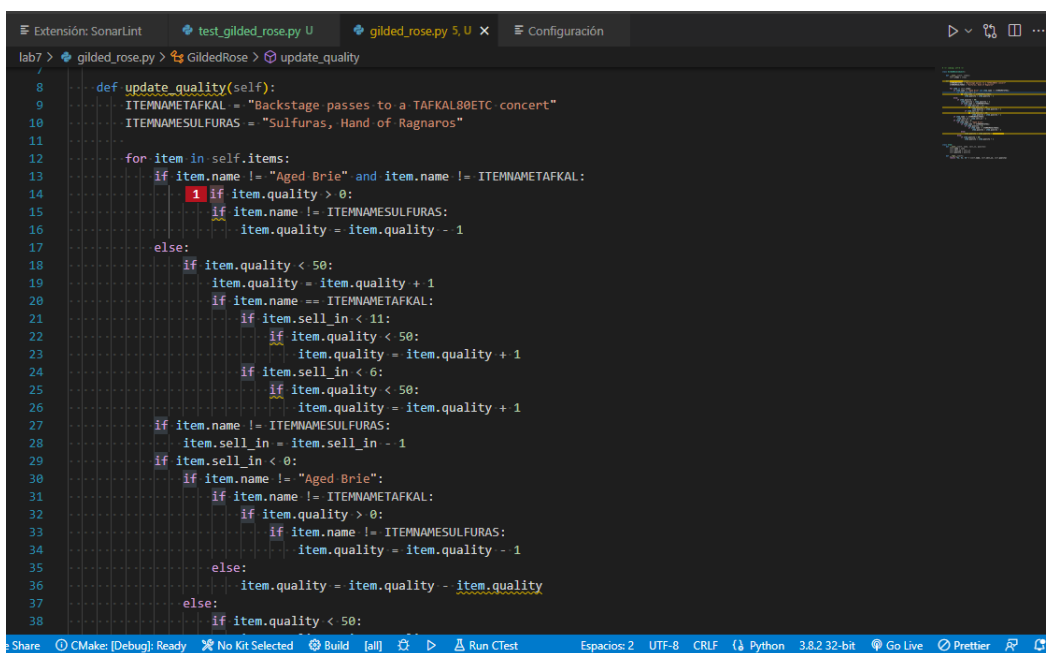
4.3.1.2. issue S1066 :

String literals should not be duplicated

Analyze your code

Code Smell Critical design

Caso I: Merge this if statement with the enclosing one. [+1 location] sonarlint (python:S1066)

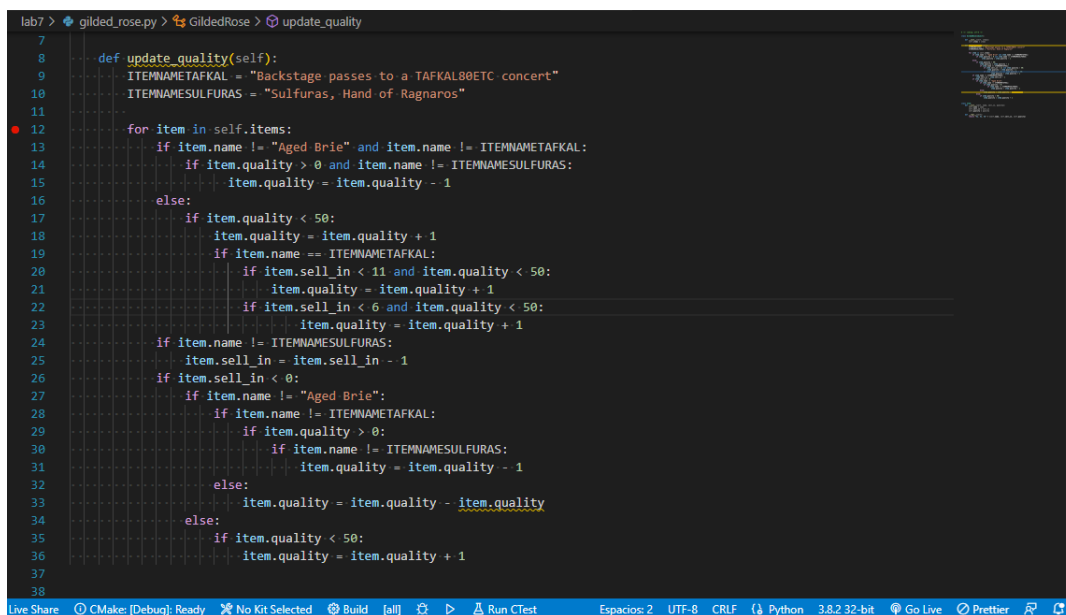


```

lab7 > gilded_rose.py > GildedRose > update_quality
8 def update_quality(self):
9     ITEMNAMEATAFKAL = "Backstage passes to a TAFKAL80ETC concert"
10    ITEMNAMESULFURAS = "Sulfuras, Hand of Ragnaros"
11
12    for item in self.items:
13        if item.name != "Aged Brie" and item.name != ITEMNAMEATAFKAL:
14            1 if item.quality > 0:
15                if item.name != ITEMNAMESULFURAS:
16                    item.quality = item.quality - 1
17            else:
18                if item.quality < 50:
19                    item.quality = item.quality + 1
20                    if item.name == ITEMNAMEATAFKAL:
21                        if item.sell_in < 11:
22                            if item.quality < 50:
23                                item.quality = item.quality + 1
24                            if item.sell_in < 6:
25                                if item.quality < 50:
26                                    item.quality = item.quality + 1
27            if item.name != ITEMNAMESULFURAS:
28                item.sell_in = item.sell_in - 1
29            if item.sell_in < 0:
30                if item.name != "Aged Brie":
31                    if item.name != ITEMNAMEATAFKAL:
32                        if item.quality > 0:
33                            if item.name != ITEMNAMESULFURAS:
34                                item.quality = item.quality - 1
35                else:
36                    item.quality = item.quality - item.quality
37            else:
38                if item.quality < 50:

```

After



```

lab7 > gilded_rose.py > GildedRose > update_quality
7
8 def update_quality(self):
9     ITEMNAMEATAFKAL = "Backstage passes to a TAFKAL80ETC concert"
10    ITEMNAMESULFURAS = "Sulfuras, Hand of Ragnaros"
11
12    for item in self.items:
13        if item.name != "Aged Brie" and item.name != ITEMNAMEATAFKAL:
14            if item.quality > 0 and item.name != ITEMNAMESULFURAS:
15                item.quality = item.quality - 1
16        else:
17            if item.quality < 50:
18                item.quality = item.quality + 1
19                if item.name == ITEMNAMEATAFKAL:
20                    if item.sell_in < 11 and item.quality < 50:
21                        item.quality = item.quality + 1
22                    if item.sell_in < 6 and item.quality < 50:
23                        item.quality = item.quality + 1
24            if item.name != ITEMNAMESULFURAS:
25                item.sell_in = item.sell_in - 1
26            if item.sell_in < 0:
27                if item.name != "Aged Brie":
28                    if item.name != ITEMNAMEATAFKAL:
29                        if item.quality > 0:
30                            if item.name != ITEMNAMESULFURAS:
31                                item.quality = item.quality - 1
32                else:
33                    item.quality = item.quality - item.quality
34            else:
35                if item.quality < 50:
36                    item.quality = item.quality + 1
37
38

```


4.3.1.3. issue S1764:

- **Descripción:** Usar el mismo valor en cualquier lado de un operador binario casi siempre es un error. En el caso de los operadores lógicos, es un error de copiar/-pegar y, por lo tanto, un error, o simplemente es un código desperdiciado y debe simplificarse. En el caso de los operadores bit a bit y la mayoría de los operadores matemáticos binarios, tener el mismo valor en ambos lados de un operador produce resultados predecibles y debe simplificarse.
- **Excepciones:** Esta regla ignora $*$, $+$ y $=$.
- **Mensaje:**
 - Corregir una de las subexpresiones idénticas a ambos lados del operador "XXX".
 - Corregir una de las subexpresiones idénticas en ambos lados de iguales.
 - Corregir una de las subexpresiones de argumento idénticas.
- **Solución:** Defina una constante en lugar de duplicar este literal "cadena" número veces.

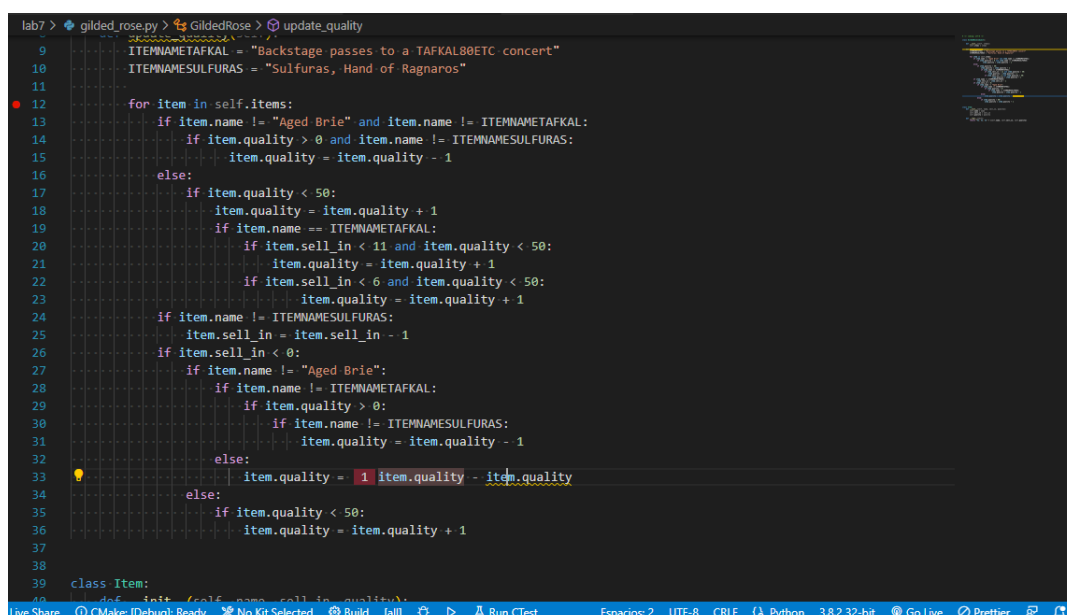
Identical expressions should not be used on both sides of a binary operator

Analyze your code

🐛 Bug 🚨 Major ? cert

Caso I: Correct one of the identical sub-expressions on both sides of operator ". [+1 location] sonarlint (python:S1764)

Before



```

lab7 > gilded_rose.py > GildedRose > update_quality
9     ITEMNAMEATAFKAL = "Backstage passes to a TAFKAL80ETC concert"
10    ITEMNAMESULFURAS = "Sulfuras, Hand of Ragnaros"
11
12    for item in self.items:
13        if item.name != "Aged Brie" and item.name != ITEMNAMEATAFKAL:
14            if item.quality > 0 and item.name != ITEMNAMESULFURAS:
15                item.quality = item.quality - 1
16            else:
17                if item.quality < 50:
18                    item.quality = item.quality + 1
19                    if item.name == ITEMNAMEATAFKAL:
20                        if item.sell_in < 11 and item.quality < 50:
21                            item.quality = item.quality + 1
22                            if item.sell_in < 6 and item.quality < 50:
23                                item.quality = item.quality + 1
24                    if item.name != ITEMNAMESULFURAS:
25                        item.sell_in = item.sell_in - 1
26                    if item.sell_in < 0:
27                        if item.name != "Aged Brie":
28                            if item.name != ITEMNAMEATAFKAL:
29                                if item.quality > 0:
30                                    if item.name != ITEMNAMESULFURAS:
31                                        item.quality = item.quality - 1
32                                else:
33                                    item.quality = 1 item.quality - item.quality
34                            else:
35                                if item.quality < 50:
36                                    item.quality = item.quality + 1
37
38    class Item:
39        def __init__(self, name, sell_in, quality):
40            self.name = name
41            self.sell_in = sell_in
42            self.quality = quality
43
44    def update_quality(self):
45        self.update_quality()
46
47    def __str__(self):
48        return f"Item: {self.name}, {self.sell_in}, {self.quality}"
49
50    def __repr__(self):
51        return f"Item: {self.name}, {self.sell_in}, {self.quality}"
52
53    def __eq__(self, other):
54        return self.name == other.name and self.sell_in == other.sell_in and self.quality == other.quality
55
56    def __lt__(self, other):
57        return self.sell_in < other.sell_in
58
59    def __gt__(self, other):
60        return self.sell_in > other.sell_in
61
62    def __le__(self, other):
63        return self.sell_in <= other.sell_in
64
65    def __ge__(self, other):
66        return self.sell_in >= other.sell_in
67
68    def __hash__(self):
69        return hash(self.name + self.sell_in + self.quality)
70
71    def __getitem__(self, index):
72        return self.items[index]
73
74    def __setitem__(self, index, value):
75        self.items[index] = value
76
77    def __len__(self):
78        return len(self.items)
79
80    def __iter__(self):
81        return iter(self.items)
82
83    def __contains__(self, item):
84        return item in self.items
85
86    def __getitem__(self, index):
87        return self.items[index]
88
89    def __setitem__(self, index, value):
90        self.items[index] = value
91
92    def __len__(self):
93        return len(self.items)
94
95    def __iter__(self):
96        return iter(self.items)
97
98    def __contains__(self, item):
99        return item in self.items
100
101    def __getitem__(self, index):
102        return self.items[index]
103
104    def __setitem__(self, index, value):
105        self.items[index] = value
106
107    def __len__(self):
108        return len(self.items)
109
110    def __iter__(self):
111        return iter(self.items)
112
113    def __contains__(self, item):
114        return item in self.items
115
116    def __getitem__(self, index):
117        return self.items[index]
118
119    def __setitem__(self, index, value):
120        self.items[index] = value
121
122    def __len__(self):
123        return len(self.items)
124
125    def __iter__(self):
126        return iter(self.items)
127
128    def __contains__(self, item):
129        return item in self.items
130
131    def __getitem__(self, index):
132        return self.items[index]
133
134    def __setitem__(self, index, value):
135        self.items[index] = value
136
137    def __len__(self):
138        return len(self.items)
139
140    def __iter__(self):
141        return iter(self.items)
142
143    def __contains__(self, item):
144        return item in self.items
145
146    def __getitem__(self, index):
147        return self.items[index]
148
149    def __setitem__(self, index, value):
150        self.items[index] = value
151
152    def __len__(self):
153        return len(self.items)
154
155    def __iter__(self):
156        return iter(self.items)
157
158    def __contains__(self, item):
159        return item in self.items
160
161    def __getitem__(self, index):
162        return self.items[index]
163
164    def __setitem__(self, index, value):
165        self.items[index] = value
166
167    def __len__(self):
168        return len(self.items)
169
170    def __iter__(self):
171        return iter(self.items)
172
173    def __contains__(self, item):
174        return item in self.items
175
176    def __getitem__(self, index):
177        return self.items[index]
178
179    def __setitem__(self, index, value):
180        self.items[index] = value
181
182    def __len__(self):
183        return len(self.items)
184
185    def __iter__(self):
186        return iter(self.items)
187
188    def __contains__(self, item):
189        return item in self.items
190
191    def __getitem__(self, index):
192        return self.items[index]
193
194    def __setitem__(self, index, value):
195        self.items[index] = value
196
197    def __len__(self):
198        return len(self.items)
199
200    def __iter__(self):
201        return iter(self.items)
202
203    def __contains__(self, item):
204        return item in self.items
205
206    def __getitem__(self, index):
207        return self.items[index]
208
209    def __setitem__(self, index, value):
210        self.items[index] = value
211
212    def __len__(self):
213        return len(self.items)
214
215    def __iter__(self):
216        return iter(self.items)
217
218    def __contains__(self, item):
219        return item in self.items
220
221    def __getitem__(self, index):
222        return self.items[index]
223
224    def __setitem__(self, index, value):
225        self.items[index] = value
226
227    def __len__(self):
228        return len(self.items)
229
230    def __iter__(self):
231        return iter(self.items)
232
233    def __contains__(self, item):
234        return item in self.items
235
236    def __getitem__(self, index):
237        return self.items[index]
238
239    def __setitem__(self, index, value):
240        self.items[index] = value
241
242    def __len__(self):
243        return len(self.items)
244
245    def __iter__(self):
246        return iter(self.items)
247
248    def __contains__(self, item):
249        return item in self.items
250
251    def __getitem__(self, index):
252        return self.items[index]
253
254    def __setitem__(self, index, value):
255        self.items[index] = value
256
257    def __len__(self):
258        return len(self.items)
259
260    def __iter__(self):
261        return iter(self.items)
262
263    def __contains__(self, item):
264        return item in self.items
265
266    def __getitem__(self, index):
267        return self.items[index]
268
269    def __setitem__(self, index, value):
270        self.items[index] = value
271
272    def __len__(self):
273        return len(self.items)
274
275    def __iter__(self):
276        return iter(self.items)
277
278    def __contains__(self, item):
279        return item in self.items
280
281    def __getitem__(self, index):
282        return self.items[index]
283
284    def __setitem__(self, index, value):
285        self.items[index] = value
286
287    def __len__(self):
288        return len(self.items)
289
290    def __iter__(self):
291        return iter(self.items)
292
293    def __contains__(self, item):
294        return item in self.items
295
296    def __getitem__(self, index):
297        return self.items[index]
298
299    def __setitem__(self, index, value):
300        self.items[index] = value
301
302    def __len__(self):
303        return len(self.items)
304
305    def __iter__(self):
306        return iter(self.items)
307
308    def __contains__(self, item):
309        return item in self.items
310
311    def __getitem__(self, index):
312        return self.items[index]
313
314    def __setitem__(self, index, value):
315        self.items[index] = value
316
317    def __len__(self):
318        return len(self.items)
319
320    def __iter__(self):
321        return iter(self.items)
322
323    def __contains__(self, item):
324        return item in self.items
325
326    def __getitem__(self, index):
327        return self.items[index]
328
329    def __setitem__(self, index, value):
330        self.items[index] = value
331
332    def __len__(self):
333        return len(self.items)
334
335    def __iter__(self):
336        return iter(self.items)
337
338    def __contains__(self, item):
339        return item in self.items
340
341    def __getitem__(self, index):
342        return self.items[index]
343
344    def __setitem__(self, index, value):
345        self.items[index] = value
346
347    def __len__(self):
348        return len(self.items)
349
350    def __iter__(self):
351        return iter(self.items)
352
353    def __contains__(self, item):
354        return item in self.items
355
356    def __getitem__(self, index):
357        return self.items[index]
358
359    def __setitem__(self, index, value):
360        self.items[index] = value
361
362    def __len__(self):
363        return len(self.items)
364
365    def __iter__(self):
366        return iter(self.items)
367
368    def __contains__(self, item):
369        return item in self.items
370
371    def __getitem__(self, index):
372        return self.items[index]
373
374    def __setitem__(self, index, value):
375        self.items[index] = value
376
377    def __len__(self):
378        return len(self.items)
379
380    def __iter__(self):
381        return iter(self.items)
382
383    def __contains__(self, item):
384        return item in self.items
385
386    def __getitem__(self, index):
387        return self.items[index]
388
389    def __setitem__(self, index, value):
390        self.items[index] = value
391
392    def __len__(self):
393        return len(self.items)
394
395    def __iter__(self):
396        return iter(self.items)
397
398    def __contains__(self, item):
399        return item in self.items
400
401    def __getitem__(self, index):
402        return self.items[index]
403
404    def __setitem__(self, index, value):
405        self.items[index] = value
406
407    def __len__(self):
408        return len(self.items)
409
410    def __iter__(self):
411        return iter(self.items)
412
413    def __contains__(self, item):
414        return item in self.items
415
416    def __getitem__(self, index):
417        return self.items[index]
418
419    def __setitem__(self, index, value):
420        self.items[index] = value
421
422    def __len__(self):
423        return len(self.items)
424
425    def __iter__(self):
426        return iter(self.items)
427
428    def __contains__(self, item):
429        return item in self.items
430
431    def __getitem__(self, index):
432        return self.items[index]
433
434    def __setitem__(self, index, value):
435        self.items[index] = value
436
437    def __len__(self):
438        return len(self.items)
439
440    def __iter__(self):
441        return iter(self.items)
442
443    def __contains__(self, item):
444        return item in self.items
445
446    def __getitem__(self, index):
447        return self.items[index]
448
449    def __setitem__(self, index, value):
450        self.items[index] = value
451
452    def __len__(self):
453        return len(self.items)
454
455    def __iter__(self):
456        return iter(self.items)
457
458    def __contains__(self, item):
459        return item in self.items
460
461    def __getitem__(self, index):
462        return self.items[index]
463
464    def __setitem__(self, index, value):
465        self.items[index] = value
466
467    def __len__(self):
468        return len(self.items)
469
470    def __iter__(self):
471        return iter(self.items)
472
473    def __contains__(self, item):
474        return item in self.items
475
476    def __getitem__(self, index):
477        return self.items[index]
478
479    def __setitem__(self, index, value):
480        self.items[index] = value
481
482    def __len__(self):
483        return len(self.items)
484
485    def __iter__(self):
486        return iter(self.items)
487
488    def __contains__(self, item):
489        return item in self.items
490
491    def __getitem__(self, index):
492        return self.items[index]
493
494    def __setitem__(self, index, value):
495        self.items[index] = value
496
497    def __len__(self):
498        return len(self.items)
499
500    def __iter__(self):
501        return iter(self.items)
502
503    def __contains__(self, item):
504        return item in self.items
505
506    def __getitem__(self, index):
507        return self.items[index]
508
509    def __setitem__(self, index, value):
510        self.items[index] = value
511
512    def __len__(self):
513        return len(self.items)
514
515    def __iter__(self):
516        return iter(self.items)
517
518    def __contains__(self, item):
519        return item in self.items
520
521    def __getitem__(self, index):
522        return self.items[index]
523
524    def __setitem__(self, index, value):
525        self.items[index] = value
526
527    def __len__(self):
528        return len(self.items)
529
530    def __iter__(self):
531        return iter(self.items)
532
533    def __contains__(self, item):
534        return item in self.items
535
536    def __getitem__(self, index):
537        return self.items[index]
538
539    def __setitem__(self, index, value):
540        self.items[index] = value
541
542    def __len__(self):
543        return len(self.items)
544
545    def __iter__(self):
546        return iter(self.items)
547
548    def __contains__(self, item):
549        return item in self.items
550
551    def __getitem__(self, index):
552        return self.items[index]
553
554    def __setitem__(self, index, value):
555        self.items[index] = value
556
557    def __len__(self):
558        return len(self.items)
559
560    def __iter__(self):
561        return iter(self.items)
562
563    def __contains__(self, item):
564        return item in self.items
565
566    def __getitem__(self, index):
567        return self.items[index]
568
569    def __setitem__(self, index, value):
570        self.items[index] = value
571
572    def __len__(self):
573        return len(self.items)
574
575    def __iter__(self):
576        return iter(self.items)
577
578    def __contains__(self, item):
579        return item in self.items
580
581    def __getitem__(self, index):
582        return self.items[index]
583
584    def __setitem__(self, index, value):
585        self.items[index] = value
586
587    def __len__(self):
588        return len(self.items)
589
590    def __iter__(self):
591        return iter(self.items)
592
593    def __contains__(self, item):
594        return item in self.items
595
596    def __getitem__(self, index):
597        return self.items[index]
598
599    def __setitem__(self, index, value):
600        self.items[index] = value
601
602    def __len__(self):
603        return len(self.items)
604
605    def __iter__(self):
606        return iter(self.items)
607
608    def __contains__(self, item):
609        return item in self.items
610
611    def __getitem__(self, index):
612        return self.items[index]
613
614    def __setitem__(self, index, value):
615        self.items[index] = value
616
617    def __len__(self):
618        return len(self.items)
619
620    def __iter__(self):
621        return iter(self.items)
622
623    def __contains__(self, item):
624        return item in self.items
625
626    def __getitem__(self, index):
627        return self.items[index]
628
629    def __setitem__(self, index, value):
630        self.items[index] = value
631
632    def __len__(self):
633        return len(self.items)
634
635    def __iter__(self):
636        return iter(self.items)
637
638    def __contains__(self, item):
639        return item in self.items
640
641    def __getitem__(self, index):
642        return self.items[index]
643
644    def __setitem__(self, index, value):
645        self.items[index] = value
646
647    def __len__(self):
648        return len(self.items)
649
650    def __iter__(self):
651        return iter(self.items)
652
653    def __contains__(self, item):
654        return item in self.items
655
656    def __getitem__(self, index):
657        return self.items[index]
658
659    def __setitem__(self, index, value):
660        self.items[index] = value
661
662    def __len__(self):
663        return len(self.items)
664
665    def __iter__(self):
666        return iter(self.items)
667
668    def __contains__(self, item):
669        return item in self.items
670
671    def __getitem__(self, index):
672        return self.items[index]
673
674    def __setitem__(self, index, value):
675        self.items[index] = value
676
677    def __len__(self):
678        return len(self.items)
679
680    def __iter__(self):
681        return iter(self.items)
682
683    def __contains__(self, item):
684        return item in self.items
685
686    def __getitem__(self, index):
687        return self.items[index]
688
689    def __setitem__(self, index, value):
690        self.items[index] = value
691
692    def __len__(self):
693        return len(self.items)
694
695    def __iter__(self):
696        return iter(self.items)
697
698    def __contains__(self, item):
699        return item in self.items
700
701    def __getitem__(self, index):
702        return self.items[index]
703
704    def __setitem__(self, index, value):
705        self.items[index] = value
706
707    def __len__(self):
708        return len(self.items)
709
710    def __iter__(self):
711        return iter(self.items)
712
713    def __contains__(self, item):
714        return item in self.items
715
716    def __getitem__(self, index):
717        return self.items[index]
718
719    def __setitem__(self, index, value):
720        self.items[index] = value
721
722    def __len__(self):
723        return len(self.items)
724
725    def __iter__(self):
726        return iter(self.items)
727
728    def __contains__(self, item):
729        return item in self.items
730
731    def __getitem__(self, index):
732        return self.items[index]
733
734    def __setitem__(self, index, value):
735        self.items[index] = value
736
737    def __len__(self):
738        return len(self.items)
739
740    def __iter__(self):
741        return iter(self.items)
742
743    def __contains__(self, item):
744        return item in self.items
745
746    def __getitem__(self, index):
747        return self.items[index]
748
749    def __setitem__(self, index, value):
750        self.items[index] = value
751
752    def __len__(self):
753        return len(self.items)
754
755    def __iter__(self):
756        return iter(self.items)
757
758    def __contains__(self, item):
759        return item in self.items
760
761    def __getitem__(self, index):
762        return self.items[index]
763
764    def __setitem__(self, index, value):
765        self.items[index] = value
766
767    def __len__(self):
768        return len(self.items)
769
770    def __iter__(self):
771        return iter(self.items)
772
773    def __contains__(self, item):
774        return item in self.items
775
776    def __getitem__(self, index):
777        return self.items[index]
778
779    def __setitem__(self, index, value):
780        self.items[index] = value
781
782    def __len__(self):
783        return len(self.items)
784
785    def __iter__(self):
786        return iter(self.items)
787
788    def __contains__(self, item):
789        return item in self.items
790
791    def __getitem__(self, index):
792        return self.items[index]
793
794    def __setitem__(self, index, value):
795        self.items[index] = value
796
797    def __len__(self):
798        return len(self.items)
799
800    def __iter__(self):
801        return iter(self.items)
802
803    def __contains__(self, item):
804        return item in self.items
805
806    def __getitem__(self, index):
807        return self.items[index]
808
809    def __setitem__(self, index, value):
810        self.items[index] = value
811
812    def __len__(self):
813        return len(self.items)
814
815    def __iter__(self):
816        return iter(self.items)
817
818    def __contains__(self, item):
819        return item in self.items
820
821    def __getitem__(self, index):
822        return self.items[index]
823
824    def __setitem__(self, index, value):
825        self.items[index] = value
826
827    def __len__(self):
828        return len(self.items)
829
830    def __iter__(self):
831        return iter(self.items)
832
833    def __contains__(self, item):
834        return item in self.items
835
836    def __getitem__(self, index):
837        return self.items[index]
838
839    def __setitem__(self, index, value):
840        self.items[index] = value
841
842    def __len__(self):
843        return len(self.items)
844
845    def __iter__(self):
846        return iter(self.items)
847
848    def __contains__(self, item):
849        return item in self.items
850
851    def __getitem__(self, index):
852        return self.items[index]
853
854    def __setitem__(self, index, value):
855        self.items[index] = value
856
857    def __len__(self):
858        return len(self.items)
859
860    def __iter__(self):
861        return iter(self.items)
862
863    def __contains__(self, item):
864        return item in self.items
865
866    def __getitem__(self, index):
867        return self.items[index]
868
869    def __setitem__(self, index, value):
870        self.items[index] = value
871
872    def __len__(self):
873        return len(self.items)
874
875    def __iter__(self):
876        return iter(self.items)
877
878    def __contains__(self, item):
879        return item in self.items
880
881    def __getitem__(self, index):
882        return self.items[index]
883
884    def __setitem__(self, index, value):
885        self.items[index] = value
886
887    def __len__(self):
888        return len(self.items)
889
890    def __iter__(self):
891        return iter(self.items)
892
893    def __contains__(self, item):
894        return item in self.items
895
896    def __getitem__(self, index):
897        return self.items[index]
898
899    def __setitem__(self, index, value):
900        self.items[index] = value
901
902    def __len__(self):
903        return len(self.items)
904
905    def __iter__(self):
906        return iter(self.items)
907
908    def __contains__(self, item):
909        return item in self.items
909

```

After

```
lab7 > gilded_rose.py > GildedRose > update_quality
9 ITEMNAMETAFKAL = "Backstage passes to a TAFKAL80ETC concert"
10 ITEMNAMESULFURAS = "Sulfuras, Hand of Ragnaros"
11
12 for item in self.items:
13     if item.name != "Aged Brie" and item.name != ITEMNAMETAFKAL:
14         if item.quality > 0 and item.name != ITEMNAMESULFURAS:
15             item.quality = item.quality - 1
16         else:
17             if item.quality < 50:
18                 item.quality = item.quality + 1
19             if item.name == ITEMNAMETAFKAL:
20                 if item.sell_in < 11 and item.quality < 50:
21                     item.quality = item.quality + 1
22                 if item.sell_in < 6 and item.quality < 50:
23                     item.quality = item.quality + 1
24             if item.name != ITEMNAMESULFURAS:
25                 item.sell_in = item.sell_in - 1
26             if item.sell_in < 0:
27                 if item.name != "Aged Brie":
28                     if item.name != ITEMNAMETAFKAL:
29                         if item.quality > 0:
30                             if item.name != ITEMNAMESULFURAS:
31                                 item.quality = item.quality - 1
32                 else:
33                     item.quality = 0
34             else:
35                 if item.quality < 50:
36                     item.quality = item.quality + 1
37
38 class Item:
39     def __init__(self, name, sell_in, quality):
40         self.name = name
41         self.sell_in = sell_in
42         self.quality = quality
```

4.3.1.4. Rule S3776

- **Descripción:** La complejidad cognitiva es una medida de lo difícil que es comprender el flujo de control de una función. Las funciones con alta Complejidad Cognitiva serán difíciles de mantener.

Cognitive Complexity of functions should not be too high

Analyze your code

Code Smell Critical brain-overload

Caso I: Refactor this function to reduce its Cognitive Complexity from 56 to the 15 allowed. [+21 locations] sonarlint (python:S3776) **Before**

```
lab7 > gilded_rose.py > GildedRose > update_quality
8 def update_quality(self):
9     ITEMNAME_TAFKAL = "Backstage passes to a TAFKAL80ETC concert"
10    ITEMNAME_SULFURAS = "Sulfuras, Hand of Ragnaros"
11
12    1 for item in self.items:
13        2 if item.name != "Aged Brie": 3 and item.name != ITEMNAME_TAFKAL:
14            4 if item.quality > 0 5 and item.name != ITEMNAME_SULFURAS:
15                item.quality = item.quality - 1
16        6 else:
17            7 if item.quality < 50:
18                item.quality = item.quality + 1
19            8 if item.name == ITEMNAME_TAFKAL:
20                9 if item.sell_in < 11 10 and item.quality < 50:
21                    item.quality = item.quality + 1
22                11 if item.sell_in < 6 12 and item.quality < 50:
23                    item.quality = item.quality + 1
24
25        13 if item.name != ITEMNAME_SULFURAS:
26            item.sell_in = item.sell_in - 1
27
28        14 if item.sell_in < 0:
29            15 if item.name != "Aged Brie":
30                16 if item.name != ITEMNAME_TAFKAL:
31                    17 if item.quality > 0:
32                        18 if item.name != ITEMNAME_SULFURAS:
33                            item.quality = item.quality - 1
34            19 else:
35                item.quality = 0
36        20 else:
37            21 if item.quality < 50:
38                item.quality = item.quality + 1
39
```

4.4. Refactoring

4.4.1. Change Value to Reference

```
1 class ItemFactory(object):
2     def create(self, name, sell_in, quality):
3         if name == "Aged Brie": return AgedBrie(name, sell_in, quality)
4         if name == "Sulfuras, Hand of Ragnaros": return Sulfuras(name,
5             ↪ sell_in, quality)
6         if name == "Backstage passes to a TAFKAL80ETC concert": return
7             ↪ Backstage(name, sell_in, quality)
8         else: return RegularItem(name, sell_in, quality)
```

4.4.2. Extract Class

```
1 class RegularItem(object):
2     def __init__(self, name, sell_in, quality):
3         self.name = name
4         self.quality = quality
```

```
5         self.sell_in = sell_in
6
7     def __repr__(self):
8         return "%s, %s, %s" % (self.name, self.sell_in, self.quality)
9
10    def update_quality(self):
11
12    class AgedBrie(RegularItem):
13        def update_quality(self):
14
15    class Sulfuras(RegularItem):
16        def update_quality(self):
17
18    class Backstage(RegularItem):
19        def update_quality(self):
```

4.4.3. Resultados:

```
1 class ItemFactory(object):
2     def create(self, name, sell_in, quality):
3         if name == "Aged Brie": return AgedBrie(name, sell_in, quality)
4         if name == "Sulfuras, Hand of Ragnaros": return Sulfuras(name,
5             ↪ sell_in, quality)
6         if name == "Backstage passes to a TAFKAL80ETC concert": return
7             ↪ Backstage(name, sell_in, quality)
8         else:
9             return RegularItem(name, sell_in, quality)
10
11 class RegularItem(object):
12     def __init__(self, name, sell_in, quality):
13         self.name = name
14         self.quality = quality
15         self.sell_in = sell_in
16
17     def __repr__(self):
18         return "%s, %s, %s" % (self.name, self.sell_in, self.quality)
19
20     def update_quality(self):
21         if self.quality < 50 and self.quality > 0:
22             self.quality = self.quality - 1
23             if self.sell_in <= 0:
24                 self.quality = self.quality - 1
25             self.sell_in = self.sell_in - 1
```

```
25
26
27 class AgedBrie(RegularItem):
28     def update_quality(self):
29         if self.quality < 50:
30             self.quality = self.quality + 1
31             self.sell_in = self.sell_in - 1
32
33
34 class Sulfuras(RegularItem):
35     def update_quality(self):
36         # TODO Este método esta vacío
37         pass
38
39 class Backstage(RegularItem):
40     def update_quality(self):
41         if self.sell_in <= 0:
42             self.quality = 0
43         elif self.sell_in <= 5:
44             self.quality = self.quality + 3
45         elif self.sell_in <= 10:
46             self.quality = self.quality + 2
47         else:
48             self.quality = self.quality + 1
49         if self.quality > 50:
50             self.quality = 50
51         self.sell_in = self.sell_in - 1
```

```
villa7523@LAPTOP-E0TSUEV5:/mnt/d/unsalab7$ python3 texttest_fixture.py
OMGHAI!
----- day 0 -----
name, sellIn, quality
+5 Dexterity Vest, 10, 20
Aged Brie, 2, 0
Elixir of the Mongoose, 5, 7
Sulfuras, Hand of Ragnaros, 0, 80
Sulfuras, Hand of Ragnaros, -1, 80
Backstage passes to a TAFKAL80ETC concert, 15, 20
Backstage passes to a TAFKAL80ETC concert, 10, 49
Backstage passes to a TAFKAL80ETC concert, 5, 49
Conjured Mana Cake, 3, 6
----- day 1 -----
name, sellIn, quality
+5 Dexterity Vest, 9, 19
Aged Brie, 1, 1
Elixir of the Mongoose, 4, 6
Sulfuras, Hand of Ragnaros, 0, 80
Sulfuras, Hand of Ragnaros, -1, 80
Backstage passes to a TAFKAL80ETC concert, 14, 21
Backstage passes to a TAFKAL80ETC concert, 9, 50
Backstage passes to a TAFKAL80ETC concert, 4, 50
Conjured Mana Cake, 2, 5
```

4.5. Nueva funcionalidad

Hace poco contratamos a un proveedor de artículos conjurados mágicamente. Esto requiere una actualización del sistema:

Los artículos conjurados degradan su calidad al doble de velocidad que los normales. Siéntete libre de realizar cualquier cambio al mensaje `updateQuality` y agregar el código que sea necesario, mientras que todo siga funcionando correctamente. Sin embargo, no alteres el objeto `Item` ni sus propiedades ya que pertenecen al goblin que está en ese rincón, que en un ataque de ira te va a liquidar de un golpe porque no cree en la cultura de código compartido.

```
1 class ItemFactory(object):
2     def create(self, name, sell_in, quality):
3         if name == "Aged Brie": return AgedBrie(name, sell_in, quality)
4         if name == "Sulfuras, Hand of Ragnaros": return Sulfuras(name,
5             ↪ sell_in, quality)
6         if name == "Backstage passes to a TAFKAL80ETC concert": return
7             ↪ Backstage(name, sell_in, quality)
8         if name == "Conjured Mana Cake": return
9             ↪ Conjured(name, sell_in, quality)
10        else:
11            return RegularItem(name, sell_in, quality)
12
13 class Conjured(RegularItem):
14     def update_quality(self):
15         if self.quality > 2:
16             self.quality = self.quality - 2
17         else:
18             self.quality = 0
19         self.sell_in = self.sell_in - 1
```

4.6. Caso de prueba para la nueva funcionalidad

```
1 def test_conjured_quality(self):
2     items = [item_factory.create("Conjured Mana Cake", 6, 2)]
3     gilded_rose = GildedRose(items)
4     gilded_rose.update_quality()
5     self.assertEqual(0, items[0].quality)
6
```

```
7 def test_conjured_sell_in(self):
8     items = [item_factory.create("Conjured Mana Cake", 6, 40)]
9     gilded_rose = GildedRose(items)
10    gilded_rose.update_quality()
11    self.assertEqual(5, items[0].sell_in)
```

4.7. Verificar y validar las actualizaciones: SonarLint y Re-ejecución de Pruebas

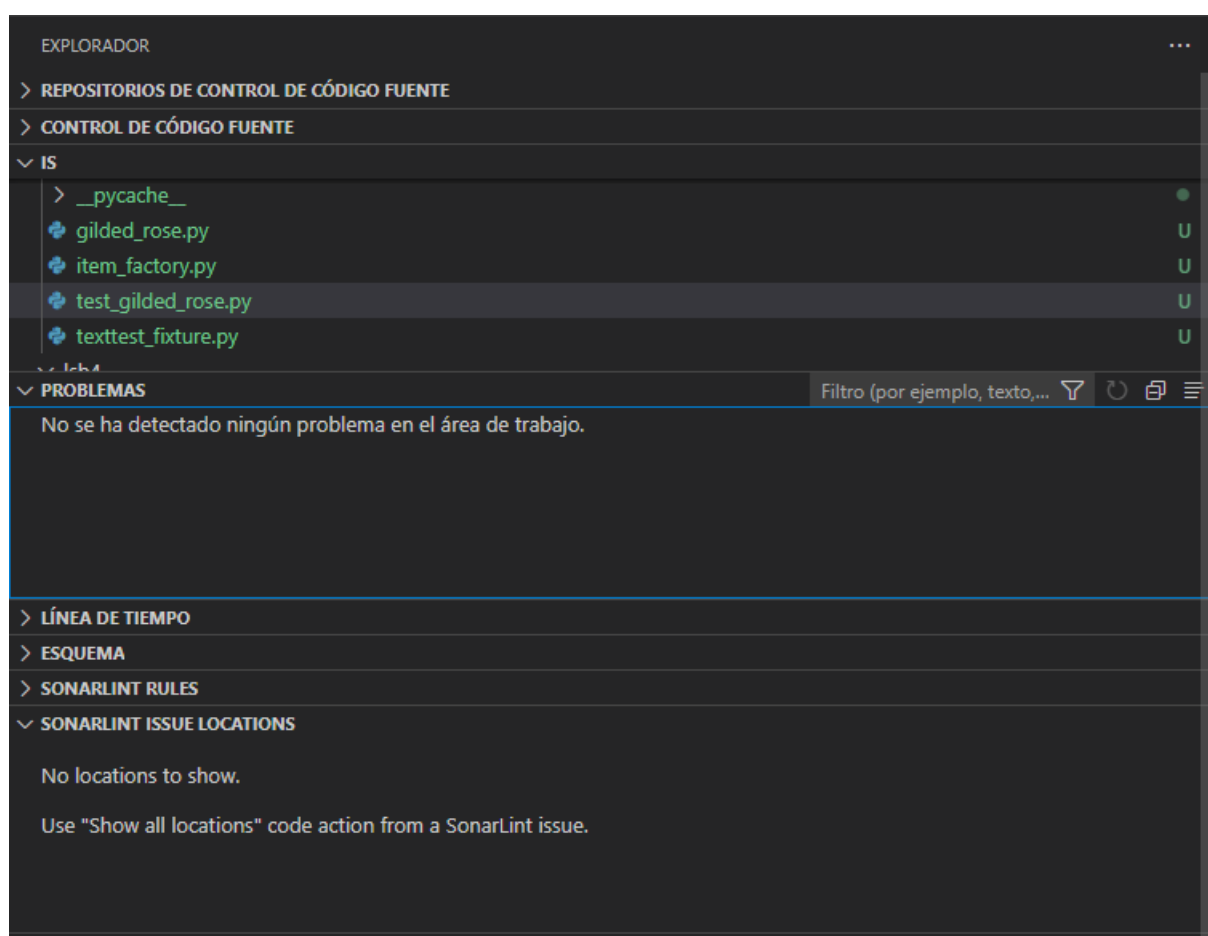


Figura 1: Sonar Lint

```
OK
villla7523@LAPTOP-E0T5UEV5:/mnt/d/unsalab7$ python3 test_gilded_rose.py
test_gilded_rose.py:27: DeprecationWarning: Please use assertEqual instead.
  self.assertEqual(3, items[0].quality)
.....
-----
Ran 10 tests in 0.019s

OK
villla7523@LAPTOP-E0T5UEV5:/mnt/d/unsalab7$
```

Figura 2: Ejecución de las pruebas para la nueva funcionalidad

```
villla7523@LAPTOP-E0T5UEV5:/mnt/d/unsalab7$ python3 texttest_fixture.py
OMGHAI!
----- day 0 -----
name, sellIn, quality
+5 Dexterity Vest, 10, 20
Aged Brie, 2, 0
Elixir of the Mongoose, 5, 7
Sulfuras, Hand of Ragnaros, 0, 80
Sulfuras, Hand of Ragnaros, -1, 80
Backstage passes to a TAFKAL80ETC concert, 15, 20
Backstage passes to a TAFKAL80ETC concert, 10, 49
Backstage passes to a TAFKAL80ETC concert, 5, 49
Conjured Mana Cake, 3, 6

----- day 1 -----
name, sellIn, quality
+5 Dexterity Vest, 9, 19
Aged Brie, 1, 1
Elixir of the Mongoose, 4, 6
Sulfuras, Hand of Ragnaros, 0, 80
Sulfuras, Hand of Ragnaros, -1, 80
Backstage passes to a TAFKAL80ETC concert, 14, 21
Backstage passes to a TAFKAL80ETC concert, 9, 50
Backstage passes to a TAFKAL80ETC concert, 4, 50
Conjured Mana Cake, 2, 5
```

Figura 3: Resultados de la ejecución principal