Comentários gerais:
-para verificar ao mesmo tempo vogais minúsculas e maiúsculas, converto para minúsculas com lower() e testo letra.lower() in 'aeiou'
-no exemplo abaixo, no primeiro ímpar o return sairá da função, sem testar todos os outros números

```
    for n in nums:  #pega cada item
        if n % 2 == 0:
            soma += n
        else:
            return None #erro: sai da função antes de testar todos
```

1.

```
def soma_pares(nums):
    soma = 0
    for n in nums:  #pega cada item
        if n % 2 == 0:
            soma += n
    return soma
```

ou

```
def soma_pares(nums):
    soma = 0
    for i in range(len(nums)): #pega cada índice de 0 até len(nums) - 1
        if nums[i] % 2 == 0:   #pega o conteúdo na posição do índice nums[i]
            soma += nums[i]
    return soma
```

ou

```
def soma_pares(nums):
    pares = []
    for n in nums:
        if n % 2 == 0:
            pares.append(n)   #inclui na lista de pares
    return sum(pares)
```

ou

```
def soma_pares(nums): return sum(n for n in nums if n % 2 == 0)
```

Comentários:
-É errado retornar None em alguma condição, o enunciado não pede isso, se a lista é vazia, ou não tem nenhum par a soma é zero
-alguns alunos confundem pegar o item com o for, com andar pelos índices. for n in nums pega cada número, for i in range(len(nums)) pega índices de 0 até len(nums) - 1, então é necessário fazer nums[i] para pegar cada número (conteúdo na posição do índice i), se você misturar as duas formas seu código não funciona

```
def soma_pares(nums):
    soma = 0
    for i in range(len(s)):  #i vale 0, 1, 2, até len(nums) - 1
        if i % 2 == 0:    #errado, estou pegando o índice e não o conteúdo dele
```

```
            soma += i
    return soma
```
-outros fizeram input dentro da função, mas a entrada já vem dentro de nums
-dentro do for, se você fizer o return, só testará o primeiro e deixará de ver o restante dos números, no exemplo abaixo o código retorna o primeiro número se ele for par, ou retorna zero, não efetuando a soma de todos os pares, pois saiu da função no primeiro item

```
    for n in nums:
        if n % 2 == 0:
            return n
        else:
            return 0
```

2.

```
def palavra_raiz(p):
    resultado = ""
    for letra in p:
        if p.count(letra) == 1:
            resultado += letra
    return resultado
```

ou

```
def palavra_raiz(p):  return ''.join(letra for letra in p if p.count(letra) == 1)
```

ou

```
def palavra raiz(p):
    resultado = ''
    for letra in p:
        if letra not in resultado:
            resultado += letra
    return resultado
```

Comentários:
-o enunciado diz "mantendo a ordem original", então fazer o uso de set() ou sorted() não preserva a ordem original. Exemplo: set('abacate') -> {'e', 'a', 'b', 't', 'c'} e sorted('abacate') -> ['a', 'a', 'a', 'b', 'c', 'e', 't'] (veja que 'e' fica antes do 't')
-alguns alunos confundem strings com listas, se eu tenho resultado = '' não posso fazer resultado.append(letra) e sim resultado += letra. Se usar resultado = [] então é possível fazer resultado.append(letra), mas no final use return ''.join(resultado) para juntar as letras que estão separadas na lista

3.

```
def espelho_parcial(s):
    meio = len(s) // 2
    primeira = s[:meio]
    segunda = s[-meio:]
    return primeira == segunda[::-1]
```

ou

```
def espelho_parcial(s):
    m = len(s) // 2
    return s[:m] == s[-m:][::-1]
```

Comentários:
-Se você usar a divisão normal, len(s) / 2, dá erro para tamanhos ímpares
-Repare que é preciso separar em duas metades e só depois inverter a segunda metade
-Alguns alunos testaram a string s toda com o inverso dela
-Outros inverteram a segunda metade errado com um único código s[m::-1], o que pega da posição do meio até o início (primeira metade), e não a segunda metade

ou sem fatiamento

```
def espelho_parcial(s):
    if len(s) % 2 != 0:
        return False  # se tamanho ímpar, não dá pra comparar metades iguais
    meio = len(s) // 2
    i = 0                 #vou andar do início até o meio
    j = len(s) - 1        #vou voltar do final até o meio
    while i < meio:
        if s[i] != s[j]:  #deu diferente posso sair com return, não preciso testar os demais
            return False
        i += 1            #anda uma posição para a direita
        j -= 1            #volta uma posição para a esquerda
    return True
```

4.

```
def consoantes_finais(palavras):
    consoantes = "bcdfghjklmnpqrstvwxyz"
    total = 0
    for palavra in palavras:
        if palavra[-1].lower() in consoantes:
            total += 1
    return total
```

ou

```
def consoantes_finais(palavras): return sum(1 for p in palavras if p[-1].lower() in "bcdfghjklmnpqrstvwxyz")
```

Comentários:
-Cuidado com as maiúsculas e minúsculas, ['aaaC', 'bbbF'] tem retorno 2
-Se você usar vogais = 'aeiou' e colocar not in vogais, irá verificar também dígitos, caracteres especiais, espaços, vogais maiúsculas, etc, não validando apenas as palavras que terminam com consoantes. ['xA', 'abc1', 'def ', 'gh*'] terá retorno errado 4, todas as palavras terminam em not in vogais, mas não são

consoantes

5.
```
def intercala_listas(l1, l2):
    resultado = []
    tam = max(len(l1), len(l2))
    for i in range(tam):
        if i < len(l1):
            resultado.append(l1[i])
        if i < len(l2):
            resultado.append(l2[i])
    return resultado
```

ou

```
def intercala_listas(l1, l2):
    return [x for pair in zip(l1, l2) for x in pair] + l1[len(l2):] + l2[len(l1):]
```

6.
```
def senha_valida(senha):
    if len(senha) < 8:
        return False
    tem_maiuscula = False
    tem_digito = False
    for c in senha:
        if c.isupper():
            tem_maiuscula = True
        if c.isdigit():
            tem_digito = True
    return tem_maiuscula and tem_digito
```

ou

```
def senha_valida(senha):
    return (
        len(senha) >= 8 and
        any(c.isupper() for c in senha) and
        any(c.isdigit() for c in senha)
    )
```

Comentários:
-Alguns alunos fizeram o código saindo na primeira condição, sem testar o resto, o que é errado

```
    if len(senha) >= 8:
        return True
    for c in senha:
        if c.isupper():
            return True
        if c.isdigit():
            return True
```

-Outros alunos testaram '0123456789' in senha, o que não funciona, pois é necessário ver digito a digito
-Testar ao mesmo tempo todas as condições não é válido, pois nunca ocorre

```
    for c in senha:
        if c.isupper() and c.isdigit() and len(senha) >= 8:
            return True
```

7.

```
def quantas_palavras_gritadas(frase):
    palavras = frase.split()
    gritos = 0
    for palavra in palavras:
        if palavra.isupper():
            gritos += 1
    return gritos
```

ou

```
def quantas_palavras_gritadas(frase): return sum(1 for p in frase.split() if p.isupper())
```

Comentários:
-A entrada é a frase, para analisar você precisar separar as palavras com split()

8.

```
def apaixonado(palavra):
    resultado = ""
    for letra in palavra:
        if letra.lower() in "aeiou":
            resultado += "♥"
        else:
            resultado += letra
    return resultado
```

ou

```
def apaixonado(palavra):
    return ''.join("♥" if c.lower() in "aeiou" else c for c in palavra)
```

Comentários:
-Strings são imutáveis, se s = 'abacate', s[0] = '*' sangra a tela, isto está claro nos vídeos de strings e em todos os exercícios das listas. Por isso uso um acumulador resultado = "" e vou concatenando ou "♥" ou a própria letra
-Não esqueça as vogais maiúsculas: letra in 'AEIOUaeiou' ou letra.lower() in 'aeiou', ou fazer palavra = palavra.lower() antes

9.

```
def gente_chata(frase):
   frase = frase.lower().split()
   #preciso converter para lower() para levar em conta maiúsculas
   #e separar as palavras da frase
```

```
   cont = 0
   for p in frase:
       if 'tipo' in p:
           cont = cont + 1
   return cont >= 3

def gente_chata(frase): return frase.lower().count("tipo") >= 3
#modo mais fácil usar.count('tipo')
```

Comentários:
-novamente não esqueça de normalizar maiúsculas e minúsculas

10.

```
def filtro_vovo(lista):
    resultado = []
    for palavra in lista:
        if "x" not in palavra.lower():
            resultado.append(palavra)
    return resultado
```

ou

```
def filtro_vovo(lista): return [p for p in lista if "x" not in p.lower()]
```

Comentários:
-A expressão 'x' not in palavra or 'X' not in palavra está errada, deve ser ao mesmo tempo, correto é and
-Veja os exemplos: 'abcX' dá True (pois 'x' not in palavra dá True) e 'abcx' também dá True (pois 'X' not in palavra da True
-A vovó não quer nenhum tipo de x na palavra