Comentários gerais:
-Funções tem entrada nos parâmetros de entrada, não temos input para as entradas. Se a função tem parâmetros e você fez input deles depois, você irá perder os valores que o usuário usou na chamada

-Quando tenho duas condições separadas por or, se a primeira é True, a linguagem não avalia a segunda, já que a condição toda já é True. Ex. candidato is None or num > candidato, se candidato for None não executa num > candidato

-Quando tenho duas condições separadas por and, se a primeira é False, a linguagem não avalia a segunda, já que a condição toda já e False. Ex. len(p) > 5 and pythonica(p), caso o tamanho de p seja menor, eu nem chamo a função pythonica(p), o que é muito interessante, pois economiza processamento

1.
```
def maior_menor_que(lista, limite):
    candidato = None
    for num in lista:
        if num < limite:
            if candidato is None or num > candidato:
                candidato = num
    return candidato
```

Versão em duas etapas:
```
def maior_menor_que(lista, limite):
    menores = []

    # Etapa 1: coletar os elementos menores que o limite
    for n in lista:
        if n < limite:
            menores.append(n)

    # Etapa 2: encontrar o maior entre os menores (sem usar max)
    if not menores:
        return None

    maior = menores[0]
    for n in menores[1:]:
        if n > maior:
            maior = n

    return maior
```

1.1 A lista pode ter números negativos, logo começar maior com zero não funciona

2.
```
def gato_atrapalha_sono(miando, hora, feriado):
    if not miando:
        return False
    if feriado:
        return hora < 8 or hora >= 23
    else:
        return hora < 5 or hora >= 21
```

2.1 A condição miando and hora < 8 or hora >= 23 sem parênteses é incorreta, pois a prioridade é do "and" em relação ao "or". Coloque parênteses para fazer a condição "or" antes. Condição correta: miando and (hora < 8 or hora >= 23)

2.2 É incorreto fazer 23 <= hora < 8 pois isso significa 23 <= hora and hora < 8. Não existe uma hora que seja ao mesmo tempo hora >= 23 e hora < 8. Logo a expressão 23 <= hora < 8 sempre será False

2.3 No horário de acordar o gato não atrapalha, só antes, hora < 8 ou hora < 5

3.
```
palavras = []
while True:
    p = input("Digite uma palavra: ")
    if p == "sair":
        break
    palavras.append(p)

def pythonica(p):
    for c in p:
        if c in 'python':
             return True
    return False

for p in palavras:
    if len(p) > 5 and pythonica(p):
        print(p)
```

3.1 Um erro muito comum é incluir uma palavra repetidamente, se houverem várias letras em 'python' nela:

```
listafinal = []
for x in palavras:
   if len(x) > 5:
      for letra in x:
          if letra in 'python':
             listafinal.append(x)
```

Seja a palavra 'yyyy', ela será incluída 4 vezes na lista final

Uma solução é ter um contador:

```
listafinal = []
for x in palavras:
   if len(x) > 5:
      cont = 0
      for letra in x:
          if letra in 'python':
             cont = cont + 1
      if cont > 0:  #vai incluir apenas uma vez
          listafinal.append(x)
```

Outra solução é dar um break na primeira letra que estiver em 'python':

```
listafinal = []
for x in palavras:
   if len(x) > 5:
      for letra in x:
          if letra in 'python':
              listafinal.append(x)
              break
```

4.

```
def válido(idade, renda):
    return idade >= 21 or (idade >= 18 and renda >= 2500)
```

4.1 A entrada de uma função vem nos parâmetros de entrada entre parênteses, fazer input de idade e renda vai alterar um valor que foi passado, o que é um erro.
Exemplo abaixo:

```
def válido(idade, renda):
  idade = int(input('Idade: '))
  renda = float(input('Renda: '))
  ...

print (válido(19, 2700))  #esses valores serão apagados por causa dos inputs da função
```

5.
Nome de função inválido: soma preços → erro de sintaxe (são duas variáveis seguidas, melhor unir o nome composto com sublinhado)
Tentativa de somar em total sem inicializá-la → erro de lógica (na verdade confundimos duas variáveis soma e total, deixar apenas com soma)
return está dentro do for → retorna no primeiro item apenas → erro de lógica (erro comum de identação, precisa estar fora do bloco de repetição)
preços está como string → precisa converter para float
print = (...) → sobrescreve função print → erro de lógica e sintaxe (erro comum, mas de difícil resolução)
print("Total: R$" + soma_preços(preços)) → precisa converter para texto o resultado de soma_preços, que é número float

```
def soma_preços(lista):
    soma = 0.0
    for preço in lista:
        soma = soma + float(preço)
    return soma

preços = "19.90 35.00 12.50 9.90".split()
print("Total: R$" + str(soma_preços(preços)))
```

Erros comuns:
5.1 Não é necessário que a variável na chamada da função tenha o mesmo nome na chamada, pode ser preços e não lista, como na definição. Não é um erro preços ter nome diferente de lista.
5.2 Podemos ter caracteres especiais no nome das variáveis, linguagens modernas permitem isso.
5.3 iniciar soma com 0.0 não é um erro, já que cada preço também é float
5.4 A lista de preços deve ser definida antes da chamada da função, não é necessário definir antes da função. A função é carregada na memória, só é executada na chamada, nesse momento lista recebe preços, já definida
5.5 Numa repetição for a variável que recebe cada item não precisa estar definida, a cada iteração da repetição preço recebe um item da lista. O código abaixo está correto:

```
    for preço in lista:
        soma = soma + float(preço)
```

5.6 preços ser uma lista de strings não está errado, basta converter no for dentro da função, assim como fizemos com os telefones da Lista V. O erro está em misturar string com float, que ocorre duas vezes, no for e no print final

6.
```
consumo = 120
mes = 0
aparelhos = 0
while consumo <= 240:
    mes = mes + 1
    if mes % 3 == 0:
        aparelhos = aparelhos + 1
    consumo = 120 + aparelhos * 10

print("Mês:", mes)
print("Consumo:", consumo)
```

7.
```
def zigue_zague(s):
    resultado = ''
    for i in range(len(s)):
        if i % 2 == 0:
            resultado += s[i].upper()
        else:
            resultado += s[i]
    return resultado
```

Outra opção, sem usar os índices, simplificando o for
```
def zigue_zague(s):
    resultado = ''
    posição = 0
    for letra in s:
        if posição % 2 == 0:
            resultado += letra.upper()
        else:
            resultado += letra
        posição = posição + 1
    return resultado
```

7.1 Um erro comum é alterar s[i].upper() diretamente, porém strings são imutáveis, uma solução é transformar s para uma lista e grudar tudo depois com join

8.
```
def pell(n):
    if n == 0: return 0        #caso particular primeiro
    a, b = 0, 1                #inicio
    k = 1
    while k <= n - 1:
    #fazer teste de mesa para saber quantas vezes fazer
        a, b = b, 2*b + a      #trocar b, a + b por b, 2*b + a
        k = k + 1
    return b
```

A partir do código de Fibonacci chegamos a conclusão que a atribuição múltipla poderia ser reaproveitada. Depois era necessário verificar casos particulares e o número de vezes que a repetição do while era feita. Se você reassistir o vídeo da explicação da Lista III, do Fibonacci, irá entender melhor o que fazer passo a passo

9.
```
x = ?  # descobrir
if x % 2 == 0 and x < 0:
    print("par negativo")
elif x % 2 != 0 and x > 0 and x < 10:
    print("ímpar pequeno")
elif x % 3 == 0 or x > 100:
    print("divisível por 3 ou grande")
else:
    print("caso genérico")
```

"par negativo" → x = -2
"ímpar pequeno" → x = 5
"divisível por 3 ou grande" → x = 99 ou x = 300
"caso genérico" → x = 8

10.
O código conta a quantidade de números pares até aparecer o 42
0 2 1 False
1 4 2 False
2 1 2 False
3 6 3 False
3

10.1 Erro comum, contar o 42 e terminar x com 4, ora 42 força um break, o que faz eu sair da repetição e não fazer o resto do bloco