CORREÇÃO 02: NOMENCLATURAS

Clean Code - Professor Ramon Venson - SATC 2026.1

Codigo 01

```
class x:
    def y(self, a, b):
        return a * b

z = x()
print(z.y(40, 160))
```

Código 01: Problemas

- Classe `x` e método `y` não descrevem intenção.
- Parâmetros `a` e `b` não indicam o que representam.
- Variável `z` não indica o papel do objeto.

> Secundário: taxa/valor por hora não estão explícitos (poderiam ser constantes).

Código 01: Sugestão

```
class CalculadoraSalario:
    def calcular_salario(self, horas_trabalhadas, taxa_por_hora):
        return horas_trabalhadas * taxa_por_hora

calculadora = CalculadoraSalario()
print(calculadora.calcular_salario(40, 160))
```

Código 01: O que foi feito?

- Renomeação de classe, método, parâmetros e variável para nomes autoexplicativos.
- Padronização para português e intenção claramente expressa.

Codigo 02

```
def d(q):
    m = q[0]
    for i in q:
        if i > m:
            m = i
    return m

mx = d([3, 7, 2, 9, 4])
print("Maior número: ", mx)
```

Código 02: Problemas

- Função `d` não descreve o que faz.
- Parâmetro `q`, variável `m` e iterador `i` são genéricos.
- `mx` é abreviação pouco clara.

> Secundário: não trata lista vazia.

Código 02: Sugestão

```
def encontrar_maior_numero(numeros):
    maior_numero = numeros[0]
    for numero in numeros:
        if numero > maior_numero:
            maior_numero = numero
    return maior_numero

maior = encontrar_maior_numero([3, 7, 2, 9, 4])
print("Maior número: ", maior)
```

Código 02: O que foi feito?

- Renomeação para expressar domínio (números/maior).
- Redução de abreviações e nomes genéricos.

Codigo 03

```
def t(x):
    return (x * 9/5) + 32

temp = t(25)
print("Temperatura em Fahrenheit: ", temp)
```

Código 03: Problemas

- Função `t` e parâmetro `x` não descrevem conversão/escala.
- Variável `temp` não explicita unidade.

> Secundário: poderia haver constantes para os fatores (9/5 e 32).

Código 03: Sugestão

```
def celsius_para_fahrenheit(temperatura_celsius):
    return (temperatura_celsius * 9/5) + 32

temperatura_fahrenheit = celsius_para_fahrenheit(25)
print("Temperatura em Fahrenheit: ", temperatura_fahrenheit)
```

Código 03: O que foi feito?

- Nomes com unidade no identificador para evitar ambiguidade.
- Função descreve claramente a transformação.

Codigo 04

```
def calculate_square(perimeter):
    return 3.14 * (perimeter ** 2)

tamanho = 5
tamanho = calculate_square(raio)
print(tamanho)
```

Código 04: Problemas

- Nome `calculate_square` não condiz com o que o código sugere (círculo).
- Parâmetro `perimeter` não representa raio.
- Variáveis `tamanho` e `raio` estão inconsistentes (usa `raio` sem definir).

> Secundário: fórmula/nomenclatura do enunciado está confusa (raio vs área); `3.14` poderia ser `math.pi`.

Código 04: Sugestão

```
def calcular_area_circulo(raio):
    return 3.14 * (raio ** 2)

raio = 5
area = calcular_area_circulo(raio)
print(area)
```

Código 04: O que foi feito?

- Renomeação de função/parâmetros/variáveis para termos do domínio (raio/área).
- Remoção de ambiguidade de nomes genéricos.

Codigo 05

```
def find_smallest_odd_number(numbers):
    largest = None
    for num in numbers:
        if num % 2 == 0 and (largest is None or num > largest):
            largest = num
    return largest

print(find_smallest_odd_number([3, 10, 7, 8, 15]))
```

Código 05: Problemas

- Nome da função diz “menor ímpar”, mas a lógica busca “maior par”.
- `numbers`, `largest`, `num` poderiam indicar explicitamente o domínio (pares).

> Secundário: deveria decidir o que retornar quando não existir número par.

Código 05: Sugestão

```
def encontrar_maior_numero_par(numeros):
    maior_par = None
    for numero in numeros:
        if numero % 2 == 0 and (maior_par is None or numero > maior_par):
            maior_par = numero
    return maior_par

print(encontrar_maior_numero_par([3, 10, 7, 8, 15]))
```

Código 05: O que foi feito?

- Função renomeada para refletir exatamente o que o corpo faz.
- Variáveis renomeadas para reduzir contradições sem alterar lógica.

Codigo 06

```
def is_full(text):
    return len(text) == 0

strData = ""

print(is_full(strData))
```

Código 06: Problemas

- `is_full` contradiz a lógica (retorna `True` quando está vazio).
- `strData` é genérico e mistura estilo.

> Secundário: pode usar `text == ""` ou `not text` (mais idiomático), mas aqui o foco é nome.

Código 06: Sugestão

```python
def esta_vazia(texto):
    return len(texto) == 0

texto = ""

print(esta_vazia(texto))
```

Código 06: O que foi feito?

- Renomeação para remover contradição sem alterar implementação.
- Padronização de estilo de nomes.

Codigo 07

```
def convert_temp(temp):
    return (temp * 9/5) + 32

def transform_temp(temp):
    return (temp - 32) * 5/9

intTemp = 25
intTemp2 = 77

print(convert_temp(intTemp))
print(transform_temp(intTemp2))
```

Código 07: Problemas

- Funções não indicam direção da conversão.
- Ambas usam `temp` genérico.
- Variáveis `intTemp` / `intTemp2` não indicam unidade (C/F).

> Secundário: nomes em inglês e português misturados.

Código 07: Sugestão

```
def celsius_para_fahrenheit(temperatura_celsius):
    return (temperatura_celsius * 9/5) + 32

def fahrenheit_para_celsius(temperatura_fahrenheit):
    return (temperatura_fahrenheit - 32) * 5/9

temperatura_celsius = 25
temperatura_fahrenheit = 77

print(celsius_para_fahrenheit(temperatura_celsius))
print(fahrenheit_para_celsius(temperatura_fahrenheit))
```

Código 07: O que foi feito?

- Direção da conversão explícita no nome da função.
- Variáveis com unidade no identificador.

Codigo 08

```
def calcula(notas):
    return sum(grades) / len(grades)

def CalculaMais(notas):
    return value(grades) >= 60

Notas = [70, 80, 50]
MediaDasNotas = value(Notas)
print(CalculaMais(MediaDasNotas))
```

Código 08: Problemas

- Nomes inconsistentes: `calcula`, `CalculaMais`, `Notas`, `MediaDasNotas`.
- Parâmetro `notas` não é usado; usa `grades` (inconsistência).
- `value(...)` não indica o que faz (e não existe no trecho).
- Chamada final mistura “média” com “lista de notas” (nomenclatura confusa).

> Secundário: há erros funcionais (variáveis/funções inexistentes e tipos incoerentes), mas o foco aqui é nomenclatura.

Código 08: Sugestão

```python
def calcular_media(notas):
    return sum(notas) / len(notas)

def aluno_aprovado(media):
    return media >= 60

notas_aluno = [70, 80, 50]
media = calcular_media(notas_aluno)
print(aluno_aprovado(media))
```

Código 08: O que foi feito?

- Padronização de nomes (snake_case) e remoção de ambiguidades.
- Separação clara entre “notas” (lista) e “média” (número).

Codigo 09

```
class Thing:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def action1(self, z)
    def action2(self, z)
    def action3(self)
    def action4(self)
```

```python
def compute(q):
    return sum(a.y for a in q)

def compute2(q):
    return max(q, key=lambda a: a.y)
```

```
# Exemplo de uso
a1 = Thing("Alice", 500)
a2 = Thing("Bob", 1200)
a3 = Thing("Charlie", 700)

a1.action1(200)
a2.action2(300)

print(a1.action3())
print(a2.action4())

accounts = [a1, a2, a3]
print(compute([a1, a2, a3]))
print(compute2(accounts).x)
```

Código 09: Problemas

- `Thing` não descreve o conceito (conta bancária).
- Atributos `x` / `y` não indicam `titular` / `saldo`.
- Métodos `action1..4` não indicam `depositar/sacar/saldo/resumo`.
- `compute` / `compute2` não deixam claro o que calculam.
- Variáveis `a1/a2/a3` são genéricas.

> Secundário: regra do enunciado diz “Bob retira 200”, mas o código retira 300.

Código 09: Sugestão

```
class ContaBancaria:
    def __init__(self, titular, saldo):
        self.titular = titular
        self.saldo = saldo
```

```
    def depositar(self, valor):
        self.saldo += valor

    def sacar(self, valor):
        if valor <= self.saldo:
            self.saldo -= valor
        else:
            print("Not enough funds")

    def obter_saldo(self):
        return self.saldo

    def obter_resumo(self):
        return f"Account: {self.titular}, Funds: {self.saldo}"
```

```
class ContaBancariaUtilidades:
    def calcular_saldo_total(contas):
        return sum(conta.saldo for conta in contas)

    def encontrar_conta_com_maior_saldo(contas):
        return max(contas, key=lambda conta: conta.saldo)
```

```
# Exemplo de uso
conta_alice = ContaBancaria("Alice", 500)
conta_bob = ContaBancaria("Bob", 1200)
conta_charlie = ContaBancaria("Charlie", 700)

conta_alice.depositar(200)
conta_bob.sacar(300)

print(conta_alice.obter_saldo())
print(conta_bob.obter_resumo())

contas = [conta_alice, conta_bob, conta_charlie]
print(calcular_saldo_total(contas))
print(encontrar_conta_com_maior_saldo(contas).titular)
```

Código 09: O que foi feito?

- Renomeação de classe/atributos/métodos para vocabulário do domínio (banco).
- Renomeação de funções utilitárias para indicar cálculo/resultado.
- Variáveis passaram a indicar a pessoa/conta correspondente.
- Criação de classe utilitária para separar lógica de negócio (conta) de operações sobre coleções de contas.

Codigo 10

```
semana = ['Domingo', 'Segunda', 'Terça', 'Quarta', 'Quinta', 'Sexta', 'Sábado']

dia = semana[2]
```

Código 10: Problemas

- `semana` poderia explicitar que é uma lista de dias.
- `dia` não indica se é “pelo índice”/“selecionado”.
- O índice `2` é um número mágico sem contexto.

> Secundário: seria melhor parametrizar o índice ou validar faixa.

Código 10: Sugestão

```
dias_da_semana = ['Domingo', 'Segunda', 'Terça', 'Quarta', 'Quinta', 'Sexta', 'Sábado']

indice_dia = 2
dia_selecionado = dias_da_semana[indice_dia]
```

Código 10: O que foi feito?

- Renomeação para explicitar coleção, índice e resultado.
- Remoção de ambiguidade com nomes mais descritivos.