CORREÇÃO 01 - CÓDIGO SUJO

Clean Code - Professor Ramon Venson - SATC 2026.1

Exemplo 01

```
def c(x, y):
    return x * y + 10
a = 5
b = 3
print(c(a, b))
```

Código que multiplica dois números e adiciona 10.

Correção 01

```python
def calcular_preco_total(preco_unitario, quantidade):
    taxa_fixa = 10
    return preco_unitario * quantidade + taxa_fixa

preco = 5
quantidade = 3
print(calcular_preco_total(preco, quantidade))
```

- Nomes descritivos para a função e variáveis.
- Melhor compreensão do propósito do código.

Exemplo 02

```
def rodar(lista):
    r = 0
    for i in range(len(lista)):
        if lista[i] % 2 == 0:
            r += lista[i]
        else:
            r -= lista[i]
    return r / len(lista)
```

Código que calcula a média de uma lista de números, somando os pares e subtraindo os ímpares.

Correção 02

```python
def calcular_saldo_pares(lista_numeros):
    numeros_pares = [num for num in lista_numeros if num % 2 == 0]
    numeros_impares = [num for num in lista_numeros if num % 2 != 0]
    saldo = sum(numeros_pares) - sum(numeros_impares)
    return saldo / len(lista_numeros)
```

- A função tem um nome mais claro.
- O código foi dividido em partes mais compreensíveis.

Exemplo 03

```python
# Função que retorna o dobro de um número
def f(n):
    # Cria uma variável
    x = 1
    # Faz um loop
    for i in range(n):
        # Multiplica x por 2
        x *= 2
    # Retorna x
    return x
```

Código que calcula 2 elevado a um expoente `n`.

Correção 03

```
def calcular_potencia_de_dois(expoente):
    resultado = 1
    for _ in range(expoente):
        resultado *= 2
    return resultado
```

- Comentários desnecessários foram removidos.
- O nome da função descreve claramente o que ela faz.
- Variáveis foram renomeadas para maior clareza.

Exemplo 04

```
def calc(x,y,z):
    if z == 1:
        return x+y
    elif z == 2:
        return x-y
    elif z == 3:
        return x*y
    elif z == 4:
        return x/y
a=10
b=5
c=3
print(calc(a,b,c))
```

Código que realiza operações matemáticas com base em uma entrada de dois números e uma operação.

Correção 04

```
def calcular_operacao(a, b, operacao):
    operacoes = {
        "soma": a + b,
        "subtracao": a - b,
        "multiplicacao": a * b,
        "divisao": a / b
    }
    return operacoes.get(operacao, "Operação inválida")

print(calcular_operacao(10, 5, "multiplicacao"))
```

- O nome da função e dos parâmetros ficaram mais intuitivos.
- Uso de um dicionário para evitar múltiplas condições repetitivas.
- O código ficou mais flexível e fácil de modificar.

Exemplo 05

```
def verifica(idade):
    if idade >= 0 and idade < 12:
        return "Criança"
    elif idade >= 12 and idade < 18:
        return "Adolescente"
    elif idade >= 18 and idade < 60:
        return "Adulto"
    elif idade >= 60:
        return "Idoso"
    else:
        return "Idade inválida"

print(verifica_idade(25))
```

Código que classifica uma idade com base em categorias.

Correção 05

```
def classificar_idade(idade):
    categorias = {
        range(0, 12): "Criança",
        range(12, 18): "Adolescente",
        range(18, 60): "Adulto",
        range(60, 200): "Idoso"
    }
    return next((
        categoria for faixa, categoria in categorias.items() if idade in faixa
    ), "Idade inválida")

print(classificar_idade(25))
```

- Evitamos repetições de condições usando um dicionário.
- O código ficou mais organizado e escalável.

Exemplo 06

```
def area_q(lado):
    return lado * lado

def area_r(base, altura):
    return base * altura

def area_t(base, altura):
    return (base * altura) / 2
```

Código que calcula a área de diferentes formas geométricas.

Correção 06

```
def calcular_area(forma, *medidas):
    areas = {
        "quadrado": lambda l: l * l,
        "retangulo": lambda b, h: b * h,
        "triangulo": lambda b, h: (b * h) / 2
    }
    return areas.get(forma, lambda *_: "Forma inválida")(*medidas)

print(calcular_area("triangulo", 10, 5))
```

- Código mais reutilizável e compacto.
- Uso de funções anônimas para evitar repetição de código.
- Agora é possível adicionar novas formas geométricas sem criar várias funções novas.

Código 07

```
def calcular_salario(horas_trabalhadas):
    return horas_trabalhadas * 50 + (horas_trabalhadas * 50 * 0.1)

print(calcular_salario(40))
```

Código que calcula o salário de um funcionário com base em suas horas trabalhadas.

Correção 07

```
VALOR_HORA = 50
BONUS_PERCENTUAL = 0.1

def calcular_salario(horas_trabalhadas):
    salario_base = horas_trabalhadas * VALOR_HORA
    bonus = salario_base * BONUS_PERCENTUAL
    return salario_base + bonus

print(calcular_salario(40))
```

- Os números mágicos foram substituídos por constantes nomeadas, tornando o código mais legível e fácil de manter.

Exemplo 08

```
def teste(n):
    if n % 2 == 0:
        return "par"
    else:
        return "ímpar"

print(tipo_numero(7))
```

Código que verifica se um número é par ou ímpar.

Correção 08

```
def par_ou_impar(n):
    return "Par" if n % 2 == 0 else "Ímpar"

print(par_ou_impar(7))
```

- Uso de *inline if* para reduzir linhas desnecessárias sem perder a clareza.

Exemplo 09

```
numeros = [3, 5, 2, 8, 1, 4]
soma_pares = 0
for numero in numeros:
    if numero % 2 == 0:
        soma_pares += numero
print(soma_pares)
```

Código que calcula a soma dos números pares em uma lista.

Correção 09

```python
def soma_numeros_pares(lista):
    return sum(num for num in lista if num % 2 == 0)

numeros = [3, 5, 2, 8, 1, 4]
print(soma_numeros_pares(numeros))
```

- Criamos uma função reutilizável.
- Substituímos o loop tradicional por list comprehension, tornando o código mais elegante e legível.

Exemplo 10

```
def criar_usuario(nome, idade, cidade, estado, telefone, email,
    empresa, local_empresa, cargo, cidade_empresa, estado_empresa):
    return f"{nome}, {idade} anos, mora em {cidade}/{estado}."
        + f"Contato: {telefone}, {email}. Trabalha na {empresa} como {cargo}, "
        + f"localizada em {cidade_empresa}/{estado_empresa}."

print(criar_usuario("João", 25, "São Paulo", "SP", "99999-9999",
"joao@email.com", "ECorp", "São Paulo", "Desenvolvedor", "São Paulo", "SP"))
```

Código que cria um usuário com informações específicas.

Correção 10

```
# Criando um objeto usando interface fluente
usuario = (
    Usuario()
    .set_nome("João")
    .set_idade(25)
    .set_cidade("São Paulo")
    .set_estado("SP")
    .set_telefone("99999-9999")
    .set_email("joao@email.com")
    .set_empresa(empresa)
)
```

- O objeto usuario é criado de forma fluente, chamando vários métodos em sequência.

```
empresa = Empresa()
    .set_nome("ECorp")
    .set_cidade("São Paulo")
    .set_estado("SP")
    .set_cargo("Desenvolvedor")
```

- A empresa é criada separadamente para melhor organização.
- O objeto da empresa pode ser reutilizado na aplicação, com várias vantagens:
  - Facilita a manutenção e atualização dos dados.
  - Melhora a legibilidade do código.
  - Economiza memória ao evitar duplicação de informações.

```
class Usuario:
    def __init__(self):
        self.nome = None
        self.idade = None

    def set_nome(self, nome):
        self.nome = nome
        return self

    def set_idade(self, idade):
        self.idade = idade
        return self
```

- Cada método *setter* retorna self, permitindo o encadeamento de chamadas.
- A classe inicia com valores `None`, e os métodos definem os atributos.