Simule detalhadamente a execução do programa abaixo, indicando na próxima página a saída (o que vai ser impresso na tela). Em cada parte, você *deve* deixar indicado os valores assumidos pelas variáveis, preenchendo as tabelas ao lado.

```
print("Parte 1")
i = 2
while i>0:
    j = 0
    while j<3:
        print(i,"+",j,"=",i+j)
        j = j+1
    i = i-1
```

i	j

```
print("Parte 2")
soma = 0
k = 5
while k < 11:
    if (k % 2 == 0) or (k % 3 == 0):
       soma = soma + k
    else:
       soma = soma - k
    print("soma =", soma)
    k = k+1
```

soma	k

```
print("Parte 3")
i = 1
j = 5
while i <= 5:
    if (2*i + j) % 3 == 0:
        print(2*i+j, "é multiplo de 3")
        j = j+1
    i = i+1
    j = j-1
```

i	j

```
print("Parte 4")
num = 23098
print("Numero original =", num)
new = 0
while num > 0:
    dig = num % 10
    new = new * 10 + dig
    num = num / 10
print("Novo numero =", new)
```

num	new	x

SAÍDA DO PROGRAMA:

---

a) Escreva em Python uma função `mirabolante(n)` que recebe um inteiro `n` com representação decimal $d_1d_2d_3d_4d_5d_6d_7d_8d_9$ (cada d_j é um dígito decimal entre 0 e 9) e devolve o valor da soma $1 * d_1 + 2 * d_2 + 3 * d_3 + \cdots + 9 * d_9$.

b) Um CPF válido é um número inteiro de 11 dígitos $d_1d_2d_3d_4d_5d_6d_7d_8d_9d_{10}d_{11}$ que satisfaz a seguinte condição: d_{10} é o resto da divisão de $1 * d_1 + 2 * d_2 + 3 * d_3 + \cdots + 9 * d_9$ por 11, e d_{11} é o resto da divisão de $1 * d_2 + 2 * d_3 + 3 * d_4 + \cdots + 9 * d_{10}$ por 11 (atenção à diferença entre as duas expressões). Escreva em Python uma função `CPFválido(n)` que devolve um valor booleano `True` ou `False` indicando se `n` representa ou não um CPF válido. Dica: use a função do item (a), mesmo que você não a tenha feito.

Considere dada uma função `f(x,y)`, onde `x` e `y` são dois argumentos em ponto flutuante (`float`). Escreva um programa em Python que pede ao usuário um valor positivo `D` e uma precisão `epsilon` e encontra o par $(x_{\max}, y_{\max})$ que maximiza o valor `f(x,y)` na grade definida pelos pontos (x_m, y_n) da forma $x_m = m*$`epsilon` e $y_n = n*$`epsilon`, onde m e n são inteiros e $x_m, y_n \in [0, D]$. Seu programa deve imprimir os valores de $(x_{\max}, y_{\max})$ e também o valor de $f(x_{\max}, y_{\max})$.

Dados dois inteiros `a` e `b`, com `a`$\geq$`0` e `b`$\geq$`1`, os seguintes trechos de programa têm como objetivo calcular o quociente e o resto da divisão de `a` por `b` (sem o uso dos operadores "`//`" e "`%`"). No final, a variável `q` deve ter valor do quociente (que corresponde à expressão `a//b` em Python) e a variável `r` deve ter o valor do resto (que corresponde a `a%b`).
Sabendo que os dois trechos de programa dados a seguir estão incorretos, você deve:

1. para cada programa, simular o código para valores de `a` e `b` que produzam uma saída incorreta. Você deve escolher esses valores e indicá-los no início da simulação.

2. corrigir cada um dos programas, modificando-os minimamente a fim de eliminar os erros. Você **não pode** escrever um código novo, devendo manter a estrutura lógica de cada uma das versões, intervindo apenas pontualmente nas instruções que geram os erros. **Evidentemente** você não pode usar os operadores "`//`" e "`%`".

```
# Programa 1

q = 1
r = a - b
while r >= b:
    r = r - b
    q = q + 1
```

```
# Programa 2

q = 0
while (q+1)*b <= a:
    r = a - q * b
    q = q + 1
```

Escreva uma função `seno` que recebe um valor x (ângulo em radianos) e calcula o seno de x através da série

$$\text{sen}x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots + (-1)^k \frac{x^{2k+1}}{(2k+1)!} + \ldots$$

incluindo todos os termos até que $\left|(-1)^k \frac{x^{2k+1}}{(2k+1)!}\right| < 0.00001$ (esse último termo *não* deve ser incluído). Dica: não escreva uma função fatorial e não use o operador exponencial (**).

O programa abaixo tenta resolver o seguinte problema: "Dados $n \geq 3$ e uma sequência com n inteiros, verificar se a sequência está em ordem estritamente crescente."

```
n = int(input("Digite o valor de n"))
x1 = int(input("Digite o primeiro numero"))
x2 = int(input("Digite o proximo numero"))
x3 = int(input("Digite o proximo numero"))
if x1 < x2 and x2 < x3:
    ordenado = True
else:
    ordenado = False

i = 0
while i<n-3:
    x2 = x3
    x3 = int(input("Digite o proximo numero"))
    if x3 > x2:
        ordenado = True
    else:
        ordenado = False
    i=i+1

if ordenado:
    print("A sequência está ordenada.")
else:
    print("A sequência não está ordenada.")
```

Este programa está com erro(s) de lógica. Você não precisa explicar o(s) erro(s) de lógica do programa, mas deve mostrar uma sequência de entrada *válida* para o problema, para a qual o programa *não funcione* corretamente. Para isso você deve simular a execução do programa para a sua entrada, mostrando qual seria a saída (incorreta) produzida pelo programa, e qual deveria ser a saída correta.

Faça um programa em Python que leia um inteiro positivo N e determina, entre todos os pares de números naturais (x, y) tais que $1 \leq x \leq y \leq N$, um par para o qual o valor da expressão $xy^3 - x^4 + y$ seja máximo, além de calcular também esse valor máximo.

Seu programa deve imprimir o par (x, y) encontrado e o valor máximo da expressão. Você pode usar funções auxiliares se quiser.

Uma sequência de números é do *Tipo Fibonacci* se cada termo é a soma dos dois que o antecedem. A mais famosa sequência desse tipo é conhecida como *sequência de Fibonacci*: $0, 1, 1, 2, 3, 5, 8, 13, 21, 34, \ldots$

Escreva um programa em Python que leia um inteiro $N \geq 3$, seguido por uma sequência de N inteiros, e imprime ao final uma mensagem informando se a sequência é do *Tipo Fibonacci* ou não.

Seu programa não precisa se preocupar com a possibilidade de $N < 3$.

O governo acaba de criar um novo imposto sobre empréstimos, o Imposto Sobre Financiamento Obtido de Legítimo Empréstimo (ISFOLE), com diversas alíquotas que incidem sobre faixas de valores diferentes. A alíquota é progressiva e varia dependendo do valor do empréstimo, da seguinte maneira:

0,00	$\leq$	EMPRÉSTIMO	$<$	10.000,00	$\rightarrow$	Alíquota = 0,03%
10.000,00	$\leq$	EMPRÉSTIMO	$<$	100.000,00	$\rightarrow$	Alíquota = 0,10%
		EMPRÉSTIMO	$\geq$	100.000,00	$\rightarrow$	Alíquota = 0,25%

Por exemplo, uma determinada pessoa tomou emprestado R$ 250.000,00 e deverá pagar de ISFOLE 0,03% sobre os primeiros R$ 10.000,00 (R$ 3,00), 0,10% sobre os 90.000 seguintes (R$ 90,00) 0,25% sobre os restantes R$150.000 (R$ 375,00), num total de R$ 468,00 devidos de ISFOLE.
Outra pessoa tomou emprestado R$ 25.000,00 e deverá pagar de ISFOLE 0,03% sobre os primeiros R$ 10.000,00 (R$ 3,00) e 0,10% sobre os R$ 15.000,00 remanescentes (R$ 15,00), num total de R$ 18,00 de imposto ISFOLE.
Uma terceira pessoa que tomou emprestado R$5.000,00 paga apenas 0,03% sobre R$ 5.000,00, num total de R$ 1,50 de ISFOLE.
Faça uma função `Isfole` que recebe como parâmetro o valor do empréstimo e retorna o valor do imposto ISFOLE que deve ser pago.

Um número inteiro positivo n é chamado de *perfeito* se for igual à soma de todos os seus divisores positivos (com exceção de n). Por exemplo, 6 é perfeito pois é divisível por 1, 2 e 3 e também vale

$$6 = 1 + 2 + 3;$$

Outro número perfeito é 28, pois

$$28 = 1 + 2 + 4 + 7 + 14$$

(observe que a soma é de divisores quaisquer, não necessariamente divisores primos).
Escreva uma função `éPerfeito` que testa se um argumento inteiro é perfeito ou não (devolvendo um valor de retorno True ou False).

Simule o programa abaixo para os valores $n = 25$ e $m = 9$:

```
def main():

    n = int(input("Digite n: "))
    m = int(input("Digite m: "))

    anterior  = n
    atual     = m

    resto = atual % anterior
    while resto != 0:
        resto = anterior % atual;
        anterior = atual;
        atual = resto;

    print("o mdc de", n,"e", m, "é", anterior)

#-----------------------------------------------
main()
```