

REPETIÇÃO: ENQUANTO

MC102 - Algoritmos e
Programação de
Computadores

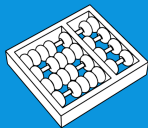
Santiago Valdés Ravelo
[https://ic.unicamp.br/~santiago/
ravelo@unicamp.br](https://ic.unicamp.br/~santiago/ravelo@unicamp.br)

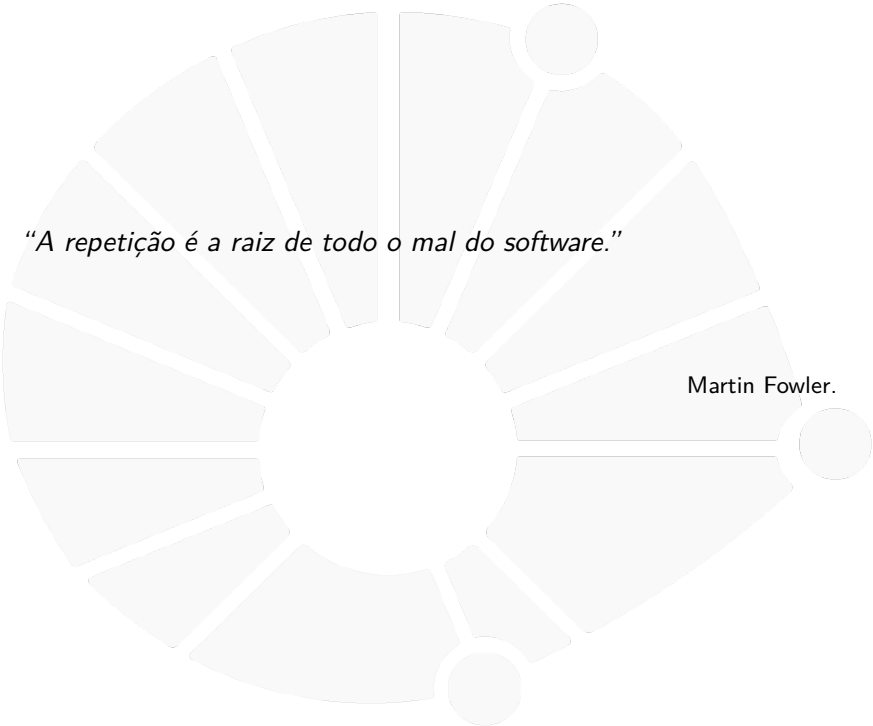
03/24

5



UNICAMP





“A repetição é a raiz de todo o mal do software.”

Martin Fowler.



DÚVIDAS DA AULA ANTERIOR

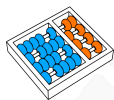


Dúvidas selecionadas

- ▶ É possível utilizar dois dos operadores "and", "or" e "not" na mesma linha de código? Ou é melhor utilizá-los separadamente, um em cada linha?
- ▶ Eu consegui entender a ideia do not preceder o and que precede o or, mas porque isso acontece? Existe algum significado por trás?
- ▶ O último elif de uma cadeia de condições, pode sempre (ou deve sempre) ser substituído por um else?
- ▶ O else e o if not x, a diferença é que o programa testa a condição 2 vezes no if not, porém o resultado é o mesmo na prática?
- ▶ Entendo o que o not faz e como ele funciona, mas existem situações onde é necessário usar o not? (Situações que o != não resolveria)
- ▶ Ao utilizar a função split, entendi que ela separa a string recebida de acordo com os espaços dentro dela e aloca cada termo entre espaços dentro de uma variável. É possível fazer com que a função split separe a string de acordo com outros caracteres? (Exemplo: a entrada é pé-de-moleque e desejo utilizar split com variáveis a,b,c para que a=pé, b=de, c=moleque)
- ▶ Quais são as aplicações práticas da álgebra booleana?



MOTIVAÇÃO



Imprimindo números

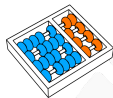
Queremos imprimir 3 números inteiros consecutivos.

```
1 n = int(input("Entre com n: "))
2 print(n)
3 print(n + 1)
4 print(n + 2)
```

E se quisermos 5 números inteiros consecutivos?

```
1 n = int(input("Entre com n: "))
2 print(n)
3 print(n + 1)
4 print(n + 2)
5 print(n + 3)
6 print(n + 4)
```

E se quisermos 100 números inteiros consecutivos?



100 números consecutivos

```
1 n = int(input("Entre com n: "))
2 print(n)
3 print(n + 1)
4 print(n + 2)
5 print(n + 3)
6 print(n + 4)
7 print(n + 5)
8 print(n + 6)
9 print(n + 7)
10 print(n + 8)
11 print(n + 9)
12 ... # o ... não é um código Python válido
13 print(n + 99)
```

Dois problemas:

- ▶ Código é repetitivo: **DRY** (Don't repeat yourself)!
- ▶ Imprime um número fixo de números consecutivos: O usuário não pode entrar com a quantidade!



Menor número

Dados 2 números inteiros, queremos saber qual é o menor.

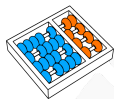
```
1 a = int(input("Entre com o número 1: "))
2 b = int(input("Entre com o número 2: "))
3 if a <= b:
4     print(a)
5 else:
6     print(b)
```

Dados 3 números inteiros, queremos saber qual é o menor.

```
1 a = int(input("Entre com o número 1: "))
2 b = int(input("Entre com o número 2: "))
3 c = int(input("Entre com o número 3: "))
4 if a <= b and a <= c:
5     print(a)
6 elif b <= c:
7     print(b)
8 else:
9     print(c)
```

Dados 100 números inteiros, queremos saber qual é o menor.

- ▶ Seguindo o mesmo padrão, o código seria bem longo...



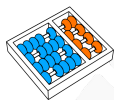
Pergunta



Como evitar esses problemas e conseguir repetir instruções sem ter código repetitivo?



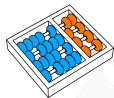
REPETIÇÃO



Comandos de Repetição

Felizmente, podemos usar um comando de repetição!

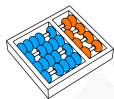
- ▶ Repete o mesmo código.
- ▶ Até que algo aconteça!



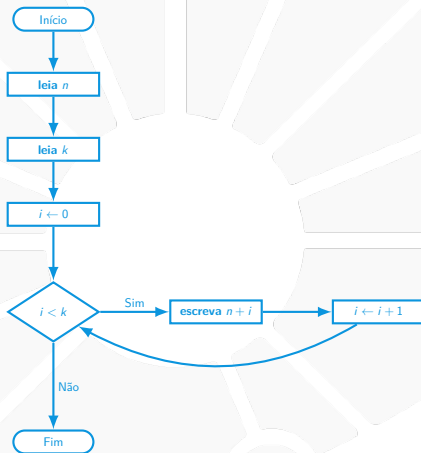
Pseudocódigo

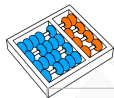
Dados os inteiros n e k , queremos imprimir os k inteiros consecutivos começando em n .

```
1 leia  $n$ 
2 leia  $k$ 
3  $i \leftarrow 0$ 
4 enquanto  $i < k$ 
5   escreva  $n + i$ 
6    $i \leftarrow i + 1$ 
```



Fluxograma

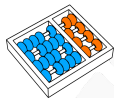




Em Python

```
1 n = int(input("Entre com n: "))
2 k = int(input("Entre com k: "))
3 i = 0
4 while i < k:
5     print(n + i)
6     i += 1
```

- ▶ A sintaxe do **while** é similar a do **if**:
 - ▶ O **:** indica o começo do bloco.
 - ▶ Que tem que ser indented.
- ▶ $x += y$ é o mesmo que $x = x + y$.
- ▶ Existem também:
 - ▶ $x -= y$,
 - ▶ $x *= y$,
 - ▶ $x /= y$,
 - ▶ $x //= y$,
 - ▶ $x %= y$,
 - ▶ $x **= y$.
 - ▶ E mais alguns outros...



Outras formas

Vimos esse código:

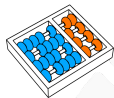
```
1 n = int(input("Entre com n: "))
2 k = int(input("Entre com k: "))
3 i = 0
4 while i < k:
5     print(n + i)
6     i += 1
```

Mas existem outras formas de fazer!

```
1 n = int(input("Entre com n: "))
2 k = int(input("Entre com k: "))
3 atual = n
4 while atual < n + k:
5     print(atual)
6     atual += 1
```



BUGS



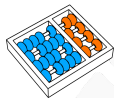
Erros?

Como saber se o programa está certo?

- ▶ Que ele não tem um *bug*?

Podemos executar o programa para alguns valores de entrada...

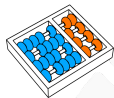
E se tiver bug, o que fazer?



Pergunta



Como encontrar bugs no código?



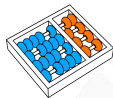
Livrando-se de bugs: teste de mesa

Uma forma prática é fazer um **teste de mesa**:

- ▶ Simular o programa usando papel e lápis.
- ▶ Considerando alguns valores de entrada.

Podemos olhar também para alguns casos mais críticos:

- ▶ E se k ou n for zero?
- ▶ E se k ou n for negativo?



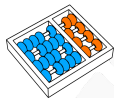
Exemplo de teste de mesa

```

1 n = int(input("Entre com n: "))
2 k = int(input("Entre com k: "))
3 i = 0
4 while i < k:
5     print(n + i)
6     i += 1

```

linha	n	k	i	obs
1	2	?	?	
2	2	3	?	
3	2	3	0	
4	2	3	0	$0 < 3$
5	2	3	0	imprime 2
6	2	3	1	
4	2	3	1	$1 < 3$
5	2	3	1	imprime 3
6	2	3	2	
4	2	3	2	$2 < 3$
5	2	3	2	imprime 4
6	2	3	3	
4	2	3	3	$3 == 3$

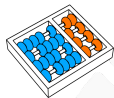


Livrando-se de bugs: depurador

Uma maneira parecida de fazer isso é usando um *debugger*:

- ▶ Um programa que permite testar o seu programa!
- ▶ Executa o programa passo a passo.
- ▶ E mostra os valores das variáveis.
- ▶ Entre várias outras funcionalidades.

O VSCode tem *debugger*!



Livrando-se de bugs: lendo o código

Outra forma de se livrar de bugs é lendo o código:

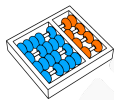
- ▶ Ler seu código com atenção.
- ▶ Pensando o que cada coisa faz.

Costuma ser mais rápido do que os outros métodos.

- ▶ Quando o bug é claro, pelo menos...



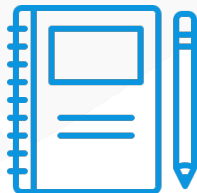
EXERCÍCIOS



Seleção condicional



Vamos fazer alguns exercícios?





Exercício 1

Relembrando. Uma Progressão Aritmética (PA):

- ▶ É uma sequência de números (a_1, a_2, \dots, a_n) ,
- ▶ onde existe um número r tal que:
- ▶ $a_{i+1} = a_i + r$
- ▶ para todo $1 \leq i < n$.

A soma S da progressão aritmética é:

$$S = \frac{n(a_1 + a_n)}{2} = a_1 n + \frac{n(n-1)r}{2}$$

Ou é o que dizem!

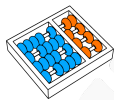
- ▶ Vamos fazer um código para ver se fórmula está correta!
- ▶ E vamos fazer um teste de mesa!



Possível solução

```
1 a_1 = int(input("Entre com a_1: "))
2 n = int(input("Entre com n: "))
3 r = int(input("Entre com r: "))
4 esperado = a_1 * n + n * (n - 1) * r / 2
5 i = 1
6 soma = 0
7 atual = a_1
8 while i <= n:
9     soma += atual
10    atual += r
11    i = i + 1
12 if soma != esperado:
13     print("Fórmula incorreta!")
14     print("Esperado: ", esperado)
15     print("Obtido: ", soma)
16 else:
17     print("Fórmula correta!")
```

- ▶ A variável `i` está *contando* até `n`.
- ▶ A variável `soma` está *acumulando* o resultado.



Exercício 2

Relembrando. Um número inteiro $p > 1$ é primo se seus divisores positivos são apenas 1 e p

Escreva um programa que, dado p , diz se p é primo ou não.

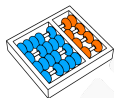


Possível solução

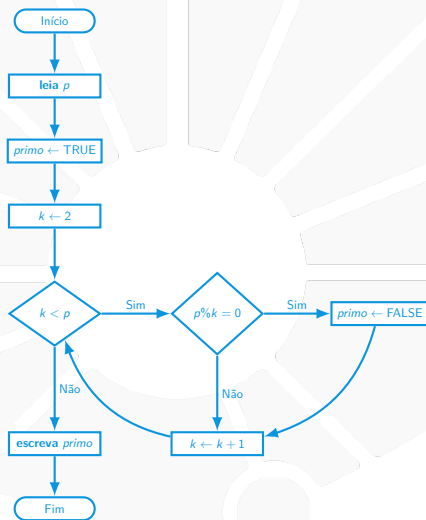
```
1 p = int(input("Entre com p: "))
2 primo = True
3 k = 2
4 while k < p:
5     if p % k == 0:
6         primo = False
7     k += 1
8 print(p >= 2 and primo)
```

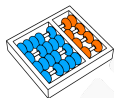
- ▶ A variável **primo** indica se **p** é primo ou não.

Vamos fazer um teste de mesa!



Fluxograma

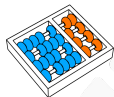




Otimizando

Podemos melhorar o código:

- ▶ Testando k apenas até \sqrt{p} :
 - ▶ Porque se p tem um divisor maior ou igual a \sqrt{p} ,
 - ▶ então p tem um divisor menor ou igual a \sqrt{p} .
- ▶ Saindo do laço quando achamos um divisor.
 - ▶ Porque já sabemos que o número não é primo.

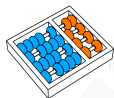


Otimizando

```
1 p = int(input("Entre com p: "))
2 primo = True
3 k = 2
4 # é melhor testar k * k <= p do que k <= p ** (1/2)
5 while k * k <= p and primo:
6     if p % k == 0:
7         primo = False
8     k += 1
9 print(p >= 2 and primo)
```

Faz diferença?

- ▶ Teste cada solução com $p = 27644437$ e veja o tempo.



Exercício 3

- (a) Leia uma sequência de números e imprima a soma.
- (b) Leia uma sequência de números e imprima o menor.
- (c) Leia uma sequência de números e veja se todos são pares.
- (d) Leia uma sequência de números e conte quantos são pares.
- (e) Dado um número, imprima a sua decomposição em números primos.



Solução (e)

Dado um número, imprima a sua decomposição em números primos:

```
1 n = int(input("Entre com o n: "))
2 d = 2
3 while n != 1:
4     if n % d == 0:
5         n //= d
6         print(d)
7     else:
8         d += 1
```

Vamos simular!

REPETIÇÃO: ENQUANTO

MC102 - Algoritmos e
Programação de
Computadores

Santiago Valdés Ravelo
[https://ic.unicamp.br/~santiago/
ravelo@unicamp.br](https://ic.unicamp.br/~santiago/ravelo@unicamp.br)

03/24

5



UNICAMP

