

## Problema 03: Fibonacci



Enunciado de um problema:

Faça um algoritmo e o programa em Python correspondente que leia um valor inteiro e informe o termo equivalente da Série de Fibonacci.

# Problema 03: Fibonacci



## Análise do Problema

Definição da Série de Fibonacci:

$$fib(pos) = \begin{cases} 0, & \text{se } pos = 0 \\ 1, & \text{se } pos = 1 \\ fib(pos - 2) + fib(pos - 1), & \text{se } pos > 1 \end{cases}$$

Descrita por Leonardo de Pisa em 1202. Nessa sequência, todo o termo a partir do segundo corresponde à soma dos dois termos imediatamente anteriores.

0 1 1 2 3 5 8 13 21 ...

# Problema 03: Fibonacci



## Descrição dos Dados e Algoritmos

1. Precisamos “lembrar” os valores Fibonacci de pos-1 e pos-2

✓ Criamos duas variáveis para isso

$$t\_2 = 0$$

$$t\_1 = 1$$

$$t = t\_1 + t\_2, \text{ para } n > 2$$

(atualizar os valores de  $t\_1$  e  $t\_2$ )



# Problema 03: Fibonacci

```
pos = int(input("Entre com o termo desejado: "))

if pos < 0:
    print("A posição não pode ser negativa!")
else:
    if pos == 0:
        fib = 0
    elif pos == 1:
        fib = 1
    else:
        t_2 = 0
        t_1 = 1
        for i in range(2, pos + 1):
            fib = t_2 + t_1
            t_2 = t_1
            t_1 = fib

print(f"O termo {pos} da série de Fibonacci é {fib}")
```

# Resumo

- \* Repetições tanto com `while` quanto com `for` permitem que se execute um conjunto de comandos várias vezes
- \* No `while` o controle do laço deve ser feito explicitamente pelo programador (condição de parada, variável de controle, etc.)
- \* No `for` esse controle fica mais implícito dependendo da definição do objeto iterável (sequência, lista, etc.)
- \* Quando se sabe exatamente quantas repetições serão necessárias, em geral é preferível aplicar o comando `for`
- \* Quando essa quantidade de repetições é incerta, o ideal é utilizar o comando `while`

# Comandos Iterativos Avançado



- Aula 06 -  
Pensamento Computacional

Prof. Me. Lucas R. C. Pessutto

# Na aula de hoje...

**01**

## Uso de variáveis na Iteração

- Variáveis Acumuladoras
- Variáveis Contadoras
- Variáveis Sinalizadoras

**02**

## Comandos Iterativos Aninhados

**03**

## Instruções de Controle de Fluxo

- Comando pass
- Comando break
- Comando continue

# Variáveis do tipo contador

- \* Utilizada em programas para contar o número de vezes em que um evento (comando ou conjunto de comandos) acontece durante a sua execução
- \* Exemplos:
  - Total de valores positivos lidos
  - Total de clientes em débito
  - Total de alunos aprovados / reprovados
- \* Características:
  - Normalmente inicializada com zero
  - Incrementada de um valor constante sempre que o um novo evento acontece
- \* Representação:

`cont = cont + 1` ou `cont += 1`



# Exemplo - Contador

Para uma turma de 10 alunos, **contar** quantos foram aprovados (média maior ou igual a 7)

```
aprovados = 0
for i in range(10):
    media = float(input(f"Nota do aluno {i + 1}: "))
    if media > 7:
        aprovados += 1

print(f"Total de aprovados: {aprovados}")
```

# Variáveis do tipo acumulador

- \* Utilizada em programas para **somar** (acumular) valores
- \* Exemplos:
  - Total de receitas / despesas no mês de abril
  - Somatório das quatro notas de um aluno
  - Somatórios / produtórios de funções matemáticas
- \* Características:
  - Normalmente inicializada com zero
  - Incrementada de um valor variável
- \* Representação:

`acc = acc + valor` ou `acc += valor`

# Exemplo - Acumulador

Somar as despesas de uma pessoa até que ela digite zero para sair do laço

```
despesas = 0
valor = 1
while valor != 0:
    valor = float(input("Informe a despesa (0 p/ sair): "))
    despesas += valor

print(f"Total Despesas: R${despesas:.2f}")
```

## Variáveis do tipo sinalizador (flag)

- \* Utilizada em programas para sinalizar quando um evento ocorrer, não importando quantas vezes
- \* Exemplos:
  - Dizer se um número é primo
  - Marcar se o usuário digitou um valor inválido ao menos uma vez durante a execução do programa
- \* Características:
  - Inicializada com um valor que indique que ela está inativa (por exemplo, zero)
  - Durante a execução do programa ela pode mudar para um valor que indique que ela foi ativada (por exemplo, um)
- \* Representação:

`flag = 0`    ou    `flag = 1`

## Exemplo - flag

Algoritmo para determinar se um número é primo. Um número primo não deve possuir divisores no intervalo  $[2, \text{num} - 1]$ . Se pelo menos um divisor for encontrado nesse intervalo, então sinalizar que o número não é primo.

```
nao_eh_primo = False

valor = int(input("Digite um número: "))

for div in range(2, valor):
    if valor % div == 0:
        nao_eh_primo = True

if nao_eh_primo:
    print("O número não é primo")
else:
    print("O número é primo")
```

# Exercício

Faça um programa que leia a idade de 5 pessoas e mostre:

- o somatório dessas idades
- quantas idades são menores de 18 anos
- se alguma das idades lidas está entre 60 e 65 anos

**Acumulador** => somar as idades

**Contador** => contar menores de idade

**Flag** => idade entre 60 e 65

# Exercício

```
soma = 0 # Acumulador
cont = 0 # Contador
digitou_60 = False # Flag

for i in range(5):
    idade = int(input(f"Digite a idade da pessoa {i + 1}: "))
    soma += idade # Usa o acumulador para somar idades

    if idade < 18:
        cont += 1 # Conta pessoas menores de idade

    if idade >= 60 and idade <= 65:
        digitou_60 = True # Modifica o valor da flag caso a idade esteja no intervalo

print(f"Somatório das idades: {soma} anos")
print(f"Foram informadas {cont} idades inferiores a 18 anos")
if digitou_60:
    print("Foram informadas idades entre 60 e 65 anos.")
else:
    print("NÃO foram informadas idades entre 60 e 65 anos.")
```

# Aninhamento de Laços

Ler 10 notas de alunos de uma turma e exigir consistência (nota no intervalo entre 0 e 10)

```
for i in range(1, 11):  
    nota = -1  
    while nota < 0 or nota > 10:  
        nota = float(input(f"Nota do aluno {i}: "))
```



# Aninhamento de Laços

Quantas vezes as expressões “Laço interno” e “Laço externo” são impressas?

**Resposta:** o while externo executa **10 vezes** enquanto que o while interno executa **10 vezes para cada repetição do laço externo**, ou seja, o laço interno executa 100 vezes.

Logo:

“Laço externo”: 10 vezes

“Laço interno”: 100 vezes

```
i = 0
while i != 10:
    print("Laço externo")
    j = 0
    for j in range(10):
        print(" - Laço interno")

    i += 1
```

Laço externo

- Laço interno
- Laço interno
- Laço interno
- Laço interno
- Laço interno
- Laço interno
- Laço interno
- Laço interno
- Laço interno
- Laço interno

Laço externo

- Laço interno
- Laço interno

...

# Problema 01: Tabuada



## Enunciado de um problema:

Faça um programa que gere o resultado da tabuada de todos os números de 1 a 10.  
O resultado deve aparecer como na tela ao lado.

Dica: use **2 comandos de repetição aninhados**: 1 para controlar os multiplicadores e outro para controlar os multiplicandos.

Tabuada do 1:

$1 \times 1 = 1$   
 $2 \times 1 = 2$   
 $3 \times 1 = 3$   
 $4 \times 1 = 4$   
 $5 \times 1 = 5$   
 $6 \times 1 = 6$   
 $7 \times 1 = 7$   
 $8 \times 1 = 8$   
 $9 \times 1 = 9$   
 $10 \times 1 = 10$

Tabuada do 2:

$1 \times 2 = 2$   
 $2 \times 2 = 4$   
 $3 \times 2 = 6$   
 $4 \times 2 = 8$   
 $5 \times 2 = 10$   
 $6 \times 2 = 12$   
 $7 \times 2 = 14$   
 $8 \times 2 = 16$   
 $9 \times 2 = 18$   
 $10 \times 2 = 20$

...

# Problema 01: Tabuada



## Implementação do Programa

```
for multiplicador in range(1, 11):  
    print(f"Tabuada do {multiplicador}")  
    for multiplicando in range(1, 11):  
        print(f"{multiplicando} x {multiplicador} = {multiplicando * multiplicador}")  
    print() # Deixa uma linha em branco
```

## Problema 02: Séries Potência



Enunciado de um problema:

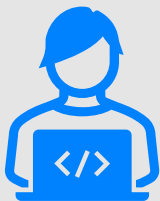
Várias funções matemáticas podem ser escritas através de séries de potências. Por exemplo, para qualquer número real  $x$ , a exponencial de  $x$  pode ser calculada via:

$$e^x = \sum_{n=0}^{+\infty} \frac{x^n}{n!} = \frac{x^0}{0!} + \frac{x^1}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} \dots$$

Podemos truncar a série, ou seja, definir uma quantidade finita de termos, e calcular a exponencial de forma aproximada usando essa quantidade de termos.

Escreva um programa que tenha como entrada o número de termos, o valor de  $x$ , e retorne como saída a aproximação da função exponencial com essa quantidade de termos.

# Problema 02: Séries Potência



## Implementação do Programa

```
import math

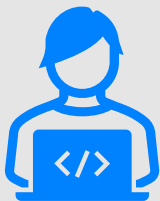
x = float(input("Informe o valor de x: "))
n = int(input("Informe o número de termos da série: "))

exp_x = 0
for i in range(1, n):
    fat = 1
    for cont_fat in range(i, 1, -1):
        fat = fat * cont_fat
    termo = (x ** i) / fat
    exp_x += termo
    print(f"Valor calculado até a iteração {i}: {exp_x}")

print(f"Valor calculado de exp({x:.2f}) = {exp_x}")
print(f"Valor de exp({x:.2f}) = {math.exp(x)}")
```

← Dá pra melhorar?

# Problema 02: Séries Potência



## Implementação do Programa

```
import math

x = float(input("Informe o valor de x: "))
n = int(input("Informe o número de termos da série: "))

exp_x = 0
fat = 1
for i in range(1, n):
    fat = fat * i
    termo = (x ** i) / fat
    exp_x += termo
    print(f"Valor calculado até a iteração {i}: {exp_x}")

print(f"Valor calculado de exp({x:.2f}) = {exp_x}")
print(f"Valor de exp({x:.2f}) = {math.exp(x)}")
```

Laço interno é desnecessário!

# Problemas Comuns

## Laços infinitos

- \* Muitas vezes escrevemos programas sintaticamente corretos mas que não funcionam
- \* Um erro muito comum é escrever laços que nunca terminam devido a sua condição de parada
- \* Você sabe dizer se o laço ao lado termina?

```
soma = 0
i = 10
while i < 100:
    soma = soma + i
    i = i - 1
print(f"Soma vale {soma}")
```

# Problemas Comuns

## Aninhamento Incorreto

- \* Note que o aninhamento de comandos é sempre hierárquico
- \* Comandos mais externos incluem os mais internos
- \* Não é permitido terminar um comando externo antes de um interno
- \* Esse código gera um `SyntaxError` na linha 5 por indentação incorreta

```
i = 0
j = 0
while i < 10:
    while j < 10:
        i += 1
            j += 1
```

Esta linha estaria supostamente dentro do bloco do primeiro `while`, mas isso não é permitido.



# Alterando o fluxo de execução de laços: Comandos pass, break e continue

# Comando pass

- \* Comando que não faz absolutamente nada!
- \* É utilizado quando queremos criar um bloco de código vazio

```
valor = 0
while valor >= 0:
    valor = int(input("Informe um valor: "))
    if valor < 10:
        pass
    else:
        print("Você digitou um número muito alto!")
```

```
Informe um valor: 8
Informe um valor: 11
Você digitou um número muito alto!
Informe um valor: 9
Informe um valor: -1
```

O comando pass não faz absolutamente nada...  
mas o bloco do if é obrigatório!

# Comando break

- \* O comando break interrompe imediatamente a execução de um laço

```
for i in range(10):  
    if i == 5:  
        break  
    print(f"i = {i}")
```

```
i = 0  
i = 1  
i = 2  
i = 3  
i = 4
```

Apesar do laço for contar até 10, a instrução break faz com que ele pare no momento em que i assume o valor 5

# Comando break-else

- \* Podemos acrescentar um bloco else ao laço de repetição, que será executado quando o laço termina normalmente, ou seja, quando o comando break não tiver sido executado

```
for i in range(5):  
    num = int(input("Digite um número: "))  
    if num == 10:  
        break  
else:  
    print("Não foi digitado o valor 10!")
```

```
Digite um número: 1  
Digite um número: 3  
Digite um número: 4  
Digite um número: 29  
Digite um número: 11  
Não foi digitado o valor 10!
```

```
Digite um número: 8  
Digite um número: 4  
Digite um número: 10
```

# Problema 04: Primos



Enunciado de um problema:

Faça um programa que leia um número maior ou igual a dois, e escreva se este número é primo



Um número  $n$  é primo caso ele seja divisível apenas por 1 e por ele mesmo.

Portanto, ele não deverá possuir divisores no intervalo  $[2, n - 1]$

Análise do Problema

# Problema 04: Primos



## Implementação do Programa

```
valor = 0
while valor < 2:
    valor = int(input("Informe um valor: "))

for div in range(2, valor):
    if valor % div == 0:
        print(f"O número {valor} não é primo")
        break
else:
    print(f"O número {valor} é primo")
```

# Comando continue

- \* O comando continue desvia a execução para o início do laço de repetição, ignorando o resto do bloco de comandos
- \* Em um laço while, a condição de parada é testada novamente para determinar se o laço continua executando
- \* No laço for, o próximo item da sequência é processado, se existir.

```
for i in range(10):  
    if i == 5:  
        continue  
    print(i, end=' ')
```



O valor 5 não foi impresso

# Problema 05: Fatoração



Enunciado de um problema:

Faça um programa que mostre os fatores primos de um número informado



**Entrada:** um número

**Saída:** os números primos que fatoram a entrada

**Processamento:** algoritmo que aprendemos na escola:

144		2
72		2
36		2
18		2
9		3
3		3
1		

Análise do Problema



# Problema 05: Fatoração



## Implementação do Programa

```
valor = 0
while valor < 2:
    valor = int(input("Informe um valor: "))

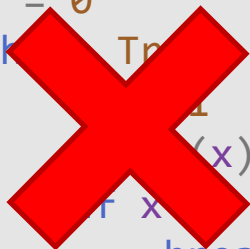
div = 2
while valor > 1:
    if valor % div == 0:
        print(f"{valor:4} | {div}")
        valor = valor // div
        continue
    div += 1

print(f"{valor:4}")
```

# Considerações Finais

- \* Use break com moderação
- \* Crie laços - especialmente while - que indiquem uma condição de parada significativa!
- \* Lembre-se: pode ser mais legível considerar comandos if e else ao invés de continue

```
x = 0
while True:
    # ...
    if x > 10:
        break
```



```
x = 0
while x <= 10:
    x += 1
    print(x)
```

