



Operadores e Controlo de fluxo



Operadores



Operadores aritméticos

- Os operadores aritméticos disponíveis são:

Operador	Significado
+	Adição
-	Subtração
*	Multiplicação
/	Divisão
%	Resto da divisão
**	Potência
//	Quociente inteiro



Operadores de atribuição

- Os operadores de atribuição principais são:

Operador	Exemplo	Significado
=	x = y	x = y
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y



Operadores relacionais

- Os operadores relacionais disponíveis são:

Operador	Significado
<code>==</code>	Igual a
<code>!=</code>	Diferente de
<code><</code>	Menor que
<code>></code>	Maior que
<code><=</code>	Menor ou igual a
<code>>=</code>	Maior ou igual a



Operadores lógicos e de identidade

- Os operadores lógicos disponíveis são:

Operador	Exemplo	Explicação
and	a and b	retorna true se a e b forem ambos true. Senão retorna false.
or	a or b	retorna true se a ou b forem true. Senão retorna false.
not	not(a)	retorna true se a for false. Senão retorna false.

- Os operadores de identidade disponíveis são:

Operador	Exemplo	Explicação
is	a is b	retorna true se a e b forem o mesmo objeto. Senão retorna false.
is not	a is not b	retorna true se a e b não forem o mesmo objeto. Senão retorna false.



Operadores de grupo

- Os operadores de grupo disponíveis são:

Operador	Exemplo	Explicação
in	a in b	retorna true se uma sequencia com um determinado valor estiver presente no objeto. Senão retorna false.
not in	a not in b	retorna true se uma sequencia com um determinado valor não estiver presente no objeto. Senão retorna false.



Operadores - Exercícios

- Indique qual o output dos seguintes programas:

```
# Prog op1
```

```
x = 15
y = 2
print(x // y)
y += 3
y *= 3
print(x==y)
print(x>=y)
print(x>15 and y>5)
print(x>15 or y>5)
```

```
# Prog op2
```

```
x = ["maçã", "banana"]
y = ["maçã", "banana"]
z = x
print(x is not z)
print(x is not y)
print(x != y)
print("banana" in z)
```

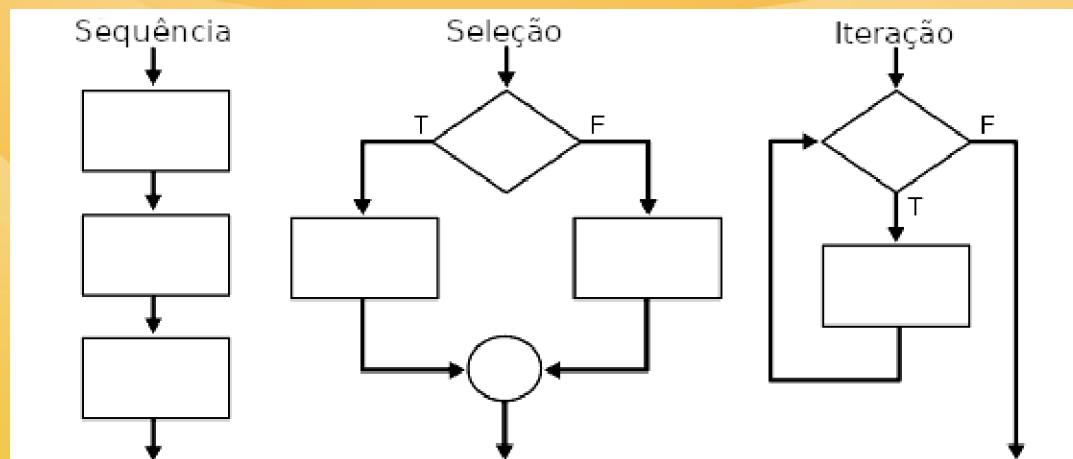


Controlo de fluxo



Controlo de fluxo

- As estruturas de controlo permitem o controlo do fluxo de dados.
- Em Python existem três tipos de estruturas de controlo:

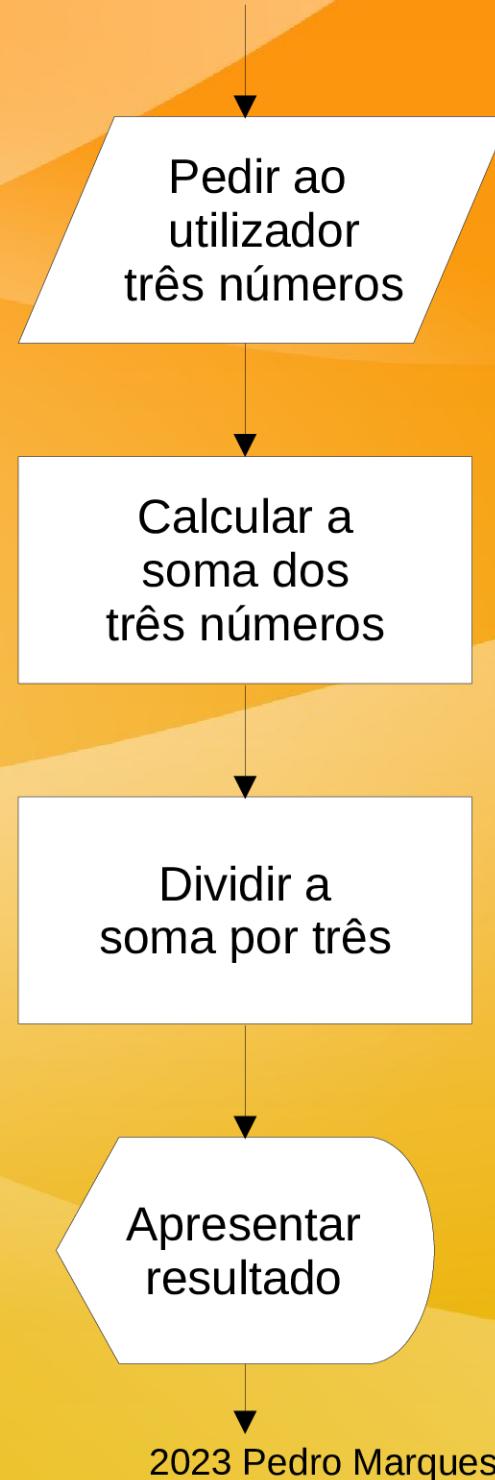




Controlo de fluxo

- Sequencia:

- Fluxo de um programa que se executa numa determinada ordem, sem mudar para outro bloco de código.
- No DFD seguinte a segunda instrução não pode ser executada antes de executar a primeira. Para além disso, não se pode dividir uma soma que não foi c





Controlo de fluxo

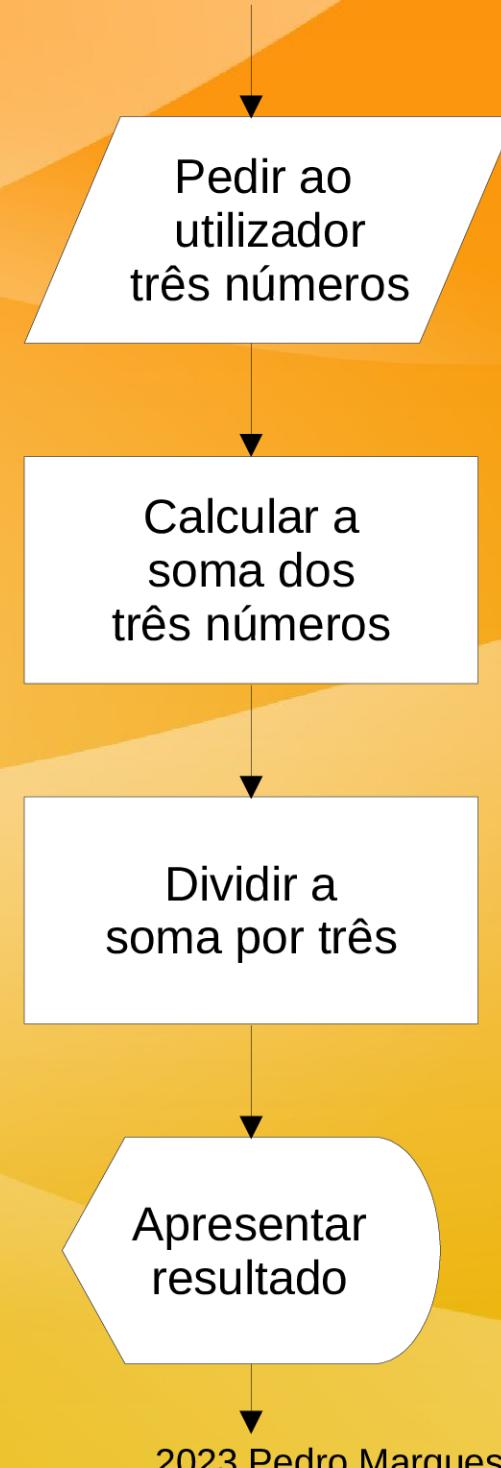
- Controlo de fluxo sequencial

```
num1 = int(input())
num2 = int(input())
num3 = int(input())

sum = num1 + num2 + num3

sum=sum/3

print("O resultado é", sum)
```

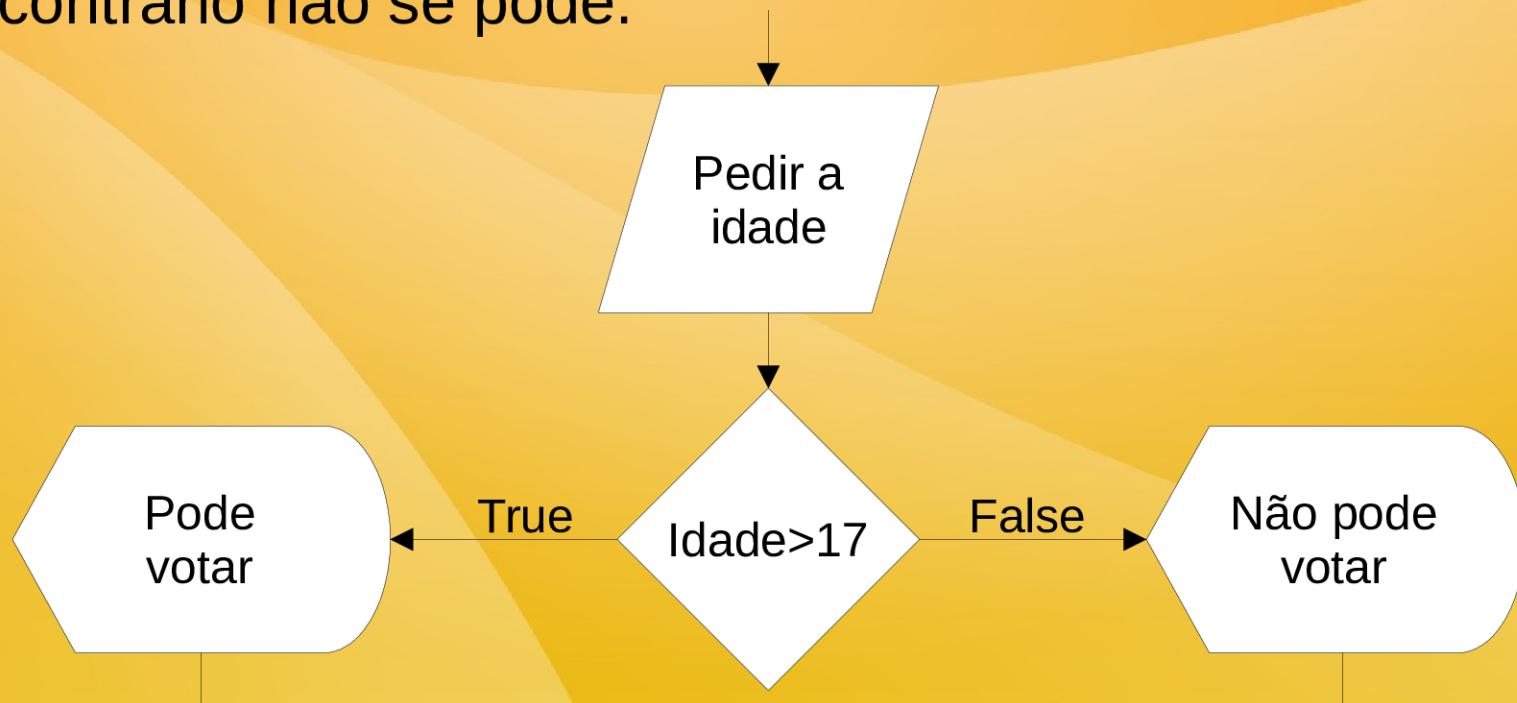




Controlo de fluxo

- Seleção:

- Fluxo de um programa que toma decisões através de um determinado critério ou condição.
- No DFD se a idade for maior que 17 pode-se votar caso contrário não se pode.

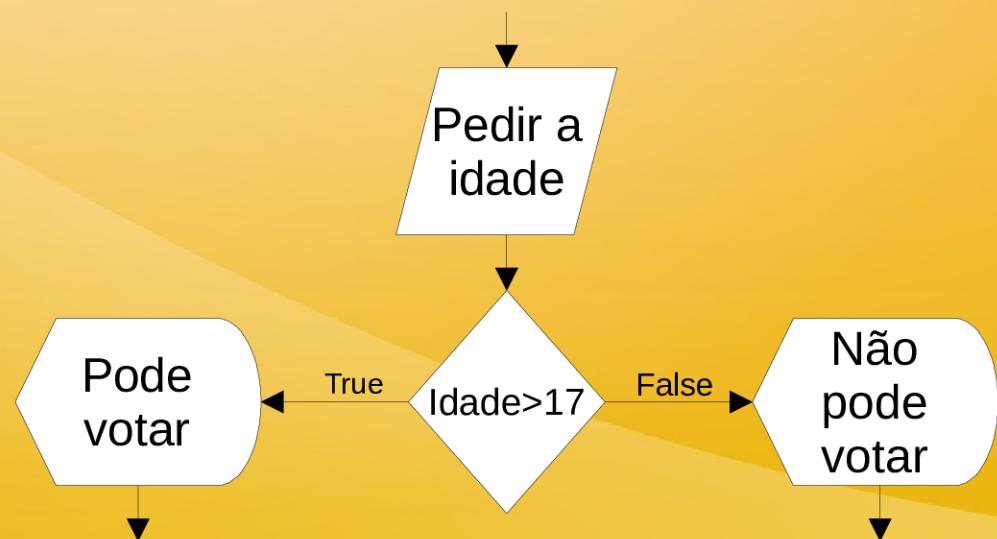




Controlo de fluxo

- Controlo de fluxo seletivo
- if - Se a condição for verdadeira executa isto ...
- else – caso a/s condições anteriores não se verifiquem então executa isto ...

```
idade = int(input())
if ( idade > 17 ):
    print("Pode votar")
else:
    print("Não pode votar")
```





Controlo de fluxo

- Elif : se a condição anterior não for verdadeira, então tenta esta condição...

```
a = 200
b = 33
if b > a:
    print("b é maior que a")
elif a == b:
    print("a e b são iguais")
else:
    print("a é maior que b")
```



Controlo de fluxo

- **and, or:** verifica se uma condição é verdadeira e / ou outra condição também o é:

```
a = 200
b = 33
c = 500
if a > b and c > a:
    print("As duas condições são verdadeiras")
if a > b or a > c:
    print("Pelo menos uma das condições é verdadeira")
```

- **not:** verifica se uma condição não é verdadeira.

```
a = 33
b = 200
if not a > b:
    print("a NÃO é maior que b")
```



Controlo de fluxo

- **Nested if:** os if podem ser utilizados dentro de outros if:

```
x = 41
if x > 10:
    print("Acima de dez,")
    if x > 20:
        print("e também acima de 20!")
    else:
        print("mas não acima de 20.")
```

- **pass:** as declarações if nunca podem estar vazias, no mínimo têm que ser ignoradas.

```
a = 33
b = 200
if b > a:
    pass
```



Controlo de fluxo

- Iterativo (repetitivo):

- Fluxo de um programa que se repete enquanto, ou até se verificar uma determinada condição.
- Os exemplos mais comuns são os ciclos while e for.





Controlo de fluxo

- Controlo de fluxo iterativo

- while:

```
conta = 5

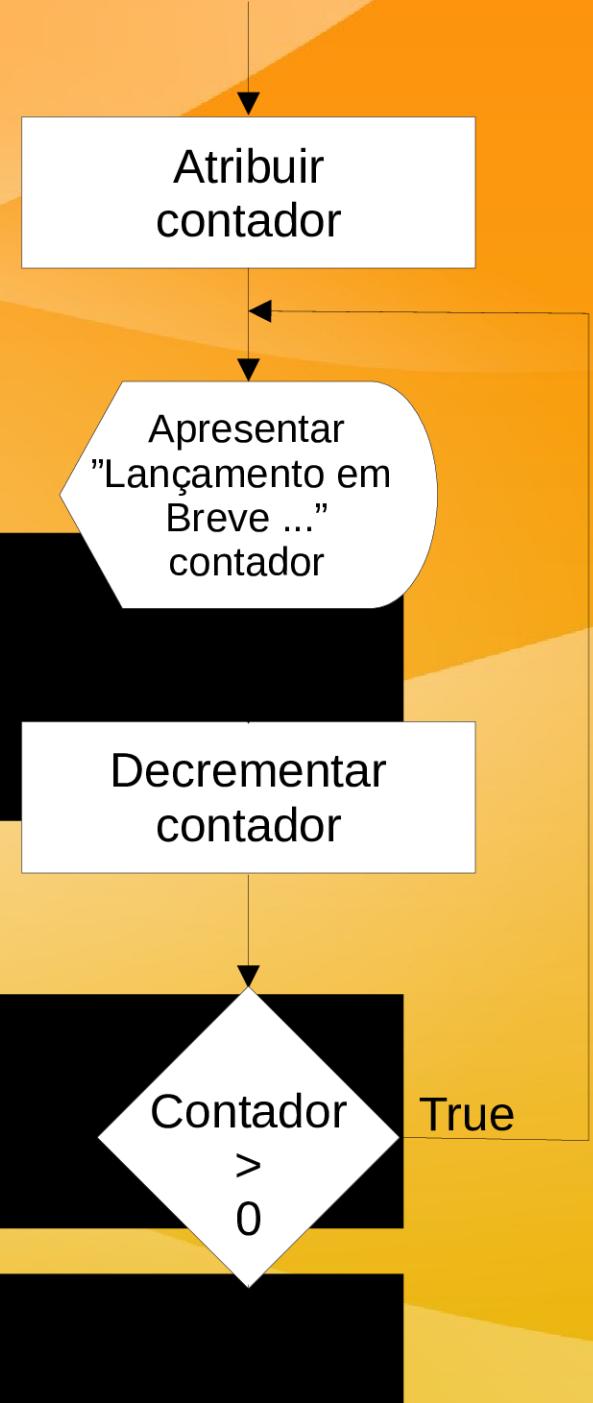
while ( conta >= 0 ):
    print("Lançamento em breve...",conta)
    conta = conta - 1
```

- for:

```
lista = ("estudante1","estudante2","estudante3")

for estudante in lista:
    print(estudante)
```

```
for x in "banana":
    print(x)
```





Controlo de fluxo

- **break**: Termina o ciclo mesmo que a condição seja verdadeira:

```
i = 1
while i < 6:
    print(i)
    if i == 3:
        break
    i += 1
```

```
frutas = ["maçã", "banana", "cereja"]
for x in frutas:
    if x == "banana":
        break
    print(x)
```

- **continue**: Termina a iteração corrente, continuando com a seguinte:

```
i = 0
while i < 6:
    i += 1
    if i == 3:
        continue
    print(i)
```

```
frutas = ["maçã", "banana", "cereja"]
for x in frutas:
    if x == "banana":
        continue
    print(x)
```



Controlo de fluxo

- **range**: A função range() retorna uma sequencia de números, com inicio em 0, por defeito, incrementada por 1, por defeito, e termina no numero especificado:

```
for x in range(6):  
    print(x)
```

- **range(inicio,fim,incremento)**

```
# sequencia de números,  
# com inicio em 2 e fim em 30,  
# incrementada por 3.
```

```
for x in range(2, 30, 3):  
    print(x)
```



Controlo de fluxo

- **else:** executa um bloco de código quando a condição já não for verdadeira:

```
i = 1
while i < 6:
    print("i=",i)
    i += 1
else:
    print("i já não é menor que 6")
```

```
for x in range(6):
    print(x)
else:
    print("Terminou finalmente!")
```



Controlo de fluxo

- **Nested loops:** os **for** podem ser utilizados dentro de outros **for**:

```
frutas = ["maçã", "banana", "cereja"]
adj = ["fresca", "grande", "saborosa"]

for x in frutas:
    for y in adj:
        print(x, y)
```

- **pass:** os loops for nunca podem estar vazios, no mínimo têm que ser ignorados.

```
for x in [0, 1, 2]:
    pass
```



Exercícios

3) Considere a lista seguinte:

```
a = [1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
```

e escreva, no menor numero de linhas possível, um programa que imprima todos os elementos da lista menores que 10.

4) Crie um programa que peça ao utilizador um número e imprima uma lista de todos os divisores desse número. (Um divisor, é um número divisível por outro número. Por exemplo, 13 é um divisor de 26 porque $26/13$ tem resto 0.)