

Processamento da Informação

Estruturas de seleção

Profa. Debora Medeiros

Determinar se um número é par ou não

- Pensando de maneira informal

- 1.

Determinar se um número é par ou não

- Pensando de maneira informal
 1. Calcular o resto da divisão por 2
 2. Se o resto for 0, é par
 3. Senão, é ímpar

Determinar se um número é par ou não

- Pensando de maneira informal
 1. Calcular o resto da divisão por 2
 2. Se o resto for 0, é par
 3. Senão, é ímpar
- Para isso, utilizamos o **if**

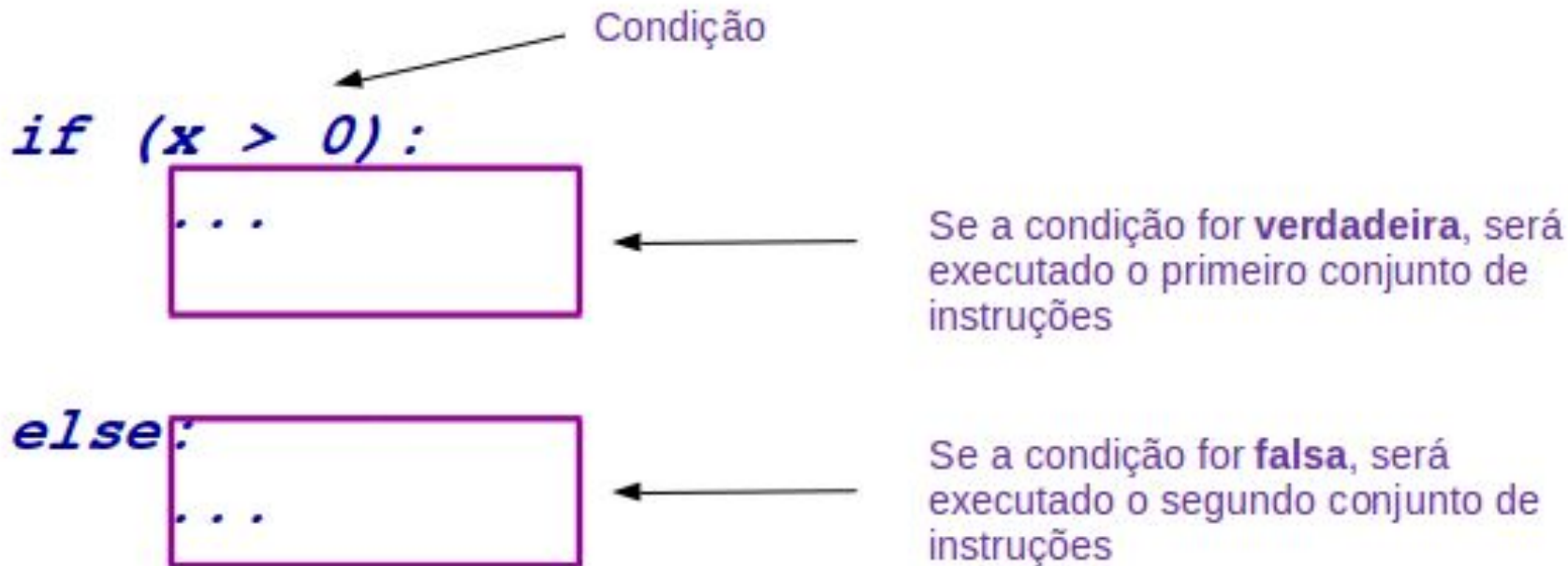
```
if (x>0):  
    Instrucao1  
    Instrucao2  
    ...
```

Se a condição for verdade, então as instruções no bloco de instruções são executadas.

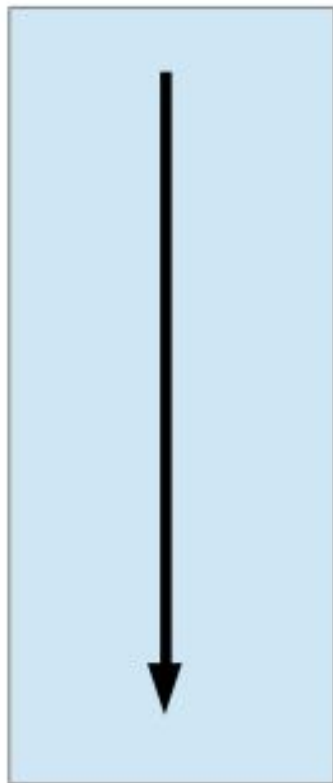
O bloco deve conter a mesma indentação.

Execução condicional

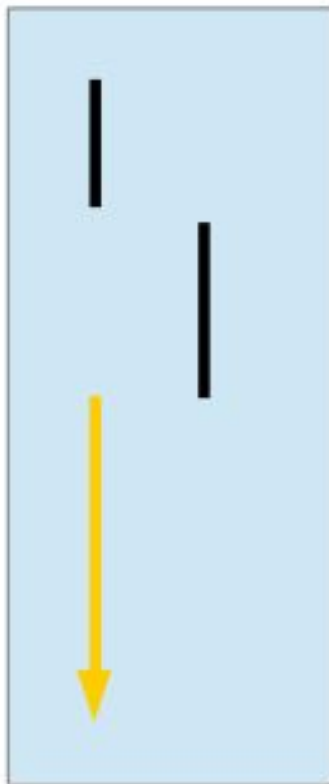
- Neste caso existem 2 possibilidades



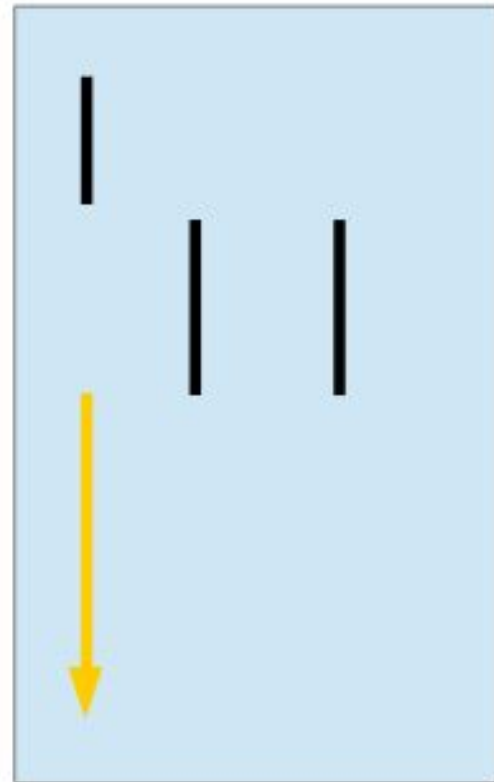
Execução condicional



Processamento
sequencial



Seleção simples



Seleção composta

Voltando ao exemplo dos números pares

```
def ehPar(x):  
    if (x % 2 > 0):  
        return False  
    else:  
        return True
```

Outro exemplo

- Se não existisse a função **min**
 - Vamos começar pensando em 2 parâmetros

Outro exemplo

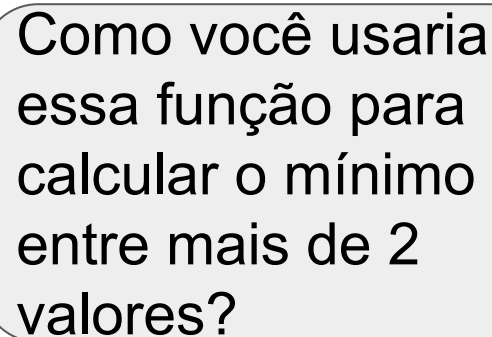
- Se não existisse a função **min**
 - Vamos começar pensando em 2 parâmetros

```
def minimo(x, y):  
    if(x < y):  
        return x  
    else:  
        return y  
  
print(minimo(4, 6))
```

Outro exemplo

- Se não existisse a função **min**
 - Vamos começar pensando em 2 parâmetros

```
def minimo(x, y):  
    if(x < y):  
        return x  
    else:  
        return y  
  
print(minimo(4, 6))
```



Como você usaria
essa função para
calcular o mínimo
entre mais de 2
valores?

Outro exemplo

- Se não existisse a função **min**
 - Vamos começar pensando em 2 parâmetros

```
def minimo(x, y):  
    if(x < y):  
        return x  
    else:  
        return y
```

```
print(minimo(4, 6))
```

```
print(minimo(4, minimo(6, 2)))
```

Desafio

O que a seguinte função realiza?

```
def que(n, m, i):  
    if(n < m):  
        print(n)  
        que(n + i, m, i)
```

Tente descobrir sem utilizar o compilador

Expressões booleanas

Expressões booleanas

- Uma expressão booleana é uma expressão que é ou Verdadeira ou Falsa.
- Os seguintes exemplos usam o operador “==”, utilizado para comparar dois operandos e produzir **true** se eles forem iguais ou **false** em caso contrário.
 - 5 == 5 → True
 - 5 == 6 → False
 - True == True → True

Expressões booleanas

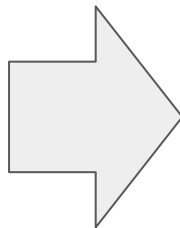
- O operador “==” é um dos operadores relacionais, os outros são:
 - $x \neq y$ # x não é igual a y
 - $x > y$ # x é maior que y
 - $x < y$ # x é menor que y
 - $x \geq y$ # x é maior ou igual a y
 - $x \leq y$ # x é menor ou igual a y
- Um erro comum é usar “=” no lugar de “==”.

Expressões booleanas

- Crie uma função que permite devolver o conceito dada a nota de um aluno
 - A: $\text{nota} \geq 9$
 - B: $7,5 \leq \text{nota} < 9$
 - C: $6 \leq \text{nota} < 7,5$
 - D: $5,0 \leq \text{nota} < 6$
 - F: $\text{nota} < 5,0$

Expressões booleanas

```
def atribuirConceito(nota):  
    if nota > 9:  
        conceito = "A"  
    else:  
        if nota >= 7.5:  
            conceito = "B"  
        else:  
            if nota >= 6:  
                conceito = "C"  
            else:  
                if nota >= 5:  
                    conceito = "D"  
                else:  
                    conceito = "F"  
    return conceito
```



```
def atribuirConceito(nota):  
    if nota > 9:  
        conceito = "A"  
    elif nota >= 7.5:  
        conceito = "B"  
    elif nota >= 6:  
        conceito = "C"  
    elif nota >= 5:  
        conceito = "D"  
    else:  
        conceito = "F"  
    return conceito
```

Expressões booleanas

- Efeito do *return*

```
def atribuirConceito(nota):  
    if nota > 9:  
        return "A"  
    if nota >= 7.5:  
        return "B"  
    if nota >= 6:  
        return "C"  
    if nota >= 5:  
        return "D"  
    return "F"
```

Operadores lógicos

Operadores lógicos

- Em Python existem 3 operadores lógicos:
 - *and*
 - *or*
 - *not*
- A semântica destes operadores é similar ao seu significado em Inglês/Português.
- Exemplo: $x > 0$ and $x < 10$
 - É verdadeira somente se x é maior a 0 e menor do que 10

Operadores lógicos

- O operador *not* inverte uma expressão booleana:
 - $\text{not } x > y$
 - é verdadeira se $x > y$ for falso
 - isto é, se x é menor ou igual a y

Exercício: risco cardíaco

- Podemos usar uma versão simplificada para calcular o risco de doença cardíaca de uma pessoa usando as seguintes regras baseadas na idade e no índice de massa corporal (IMC):

IMC	idade	
	<45	>= 45
	< 22.0	>= 22.0
	baixo	médio
	médio	alto

Exercício: risco cardíaco

```
def riscoCardiaco(idade, imc):  
    if idade < 45:  
        if imc < 22:  
            risco = "baixo"  
        else:  
            risco = "médio"  
    else:  
        if imc < 22:  
            risco = "médio"  
        else:  
            risco = "alto"  
    return risco
```

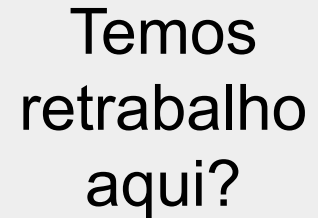
IMC

idade

	<45	>= 45
< 22.0	baixo	médio
>= 22.0	médio	alto

Exercício: risco cardíaco


```
def riscoCardiaco(idade, imc):  
    if idade < 45:  
        if imc < 22:  
            risco = "baixo"  
        else:  
            risco = "médio"  
    else:  
        if imc < 22:  
            risco = "médio"  
        else:  
            risco = "alto"  
    return risco
```



Temos
retrabalho
aqui?

Exercício: risco cardíaco

```
def riscoCardiaco(idade, imc):  
    if idade < 45:  
        if imc < 22:  
            risco = "baixo"  
        else:  
            risco = "médio"  
    else:  
        if imc < 22:  
            risco = "médio"  
        else:  
            risco = "alto"  
    return risco
```



imc < 22 é
testado 2x

Exercício: risco cardíaco

```
def riscoCardiaco(idade, imc):  
    jovem = idade < 45  
    magro = imc < 22  
  
    if jovem:  
        if magro:  
            risco = "baixo"  
        else:  
            risco = "médio"  
    else:  
        if magro:  
            risco = "médio"  
        else:  
            risco = "alto"  
    return risco
```

IMC

idade

	<45	>= 45
< 22.0	baixo	médio
>= 22.0	médio	alto

Exercício: risco cardíaco

```
def riscoCardiaco(idade, imc):  
    jovem = idade < 45  
    magro = imc < 22  
  
    if jovem and magro:  
        risco = "baixo"  
    elif jovem and not magro:  
        risco = "médio"  
    elif not jovem and magro:  
        risco = "médio"  
    elif not jovem and not magro:  
        risco = "alto"  
    return risco
```


IMC

idade

	<45	>= 45
< 22.0	baixo	médio
>= 22.0	médio	alto

Exercício: risco cardíaco

```
def riscoCardiaco(idade, imc):  
    jovem = idade < 45  
    magro = imc < 22  
  
    if jovem and magro:  
        risco = "baixo"  
    elif jovem and not magro:  
        risco = "médio"  
    elif not jovem and magro:  
        risco = "médio"  
    elif not jovem and not magro:  
        risco = "alto"  
    return risco
```



Ainda tem coisa
desnecessária
aqui?

Exercício: risco cardíaco

```
def riscoCardiaco(idade, imc):  
    jovem = idade < 45  
    magro = imc < 22  
  
    if jovem and magro:  
        risco = "baixo"  
    elif jovem and not magro:  
        risco = "médio"  
    elif not jovem and magro:  
        risco = "médio"  
    else:  
        risco = "alto"  
    return risco
```

Exercício: soma

- Crie uma função em que, dados 3 números como parâmetros, permita verificar se a soma de quaisquer par de números gera a soma do terceiro número.
 - Tente não usar condicionais
 - Sua função deve devolver True ou False:

Exercício: soma

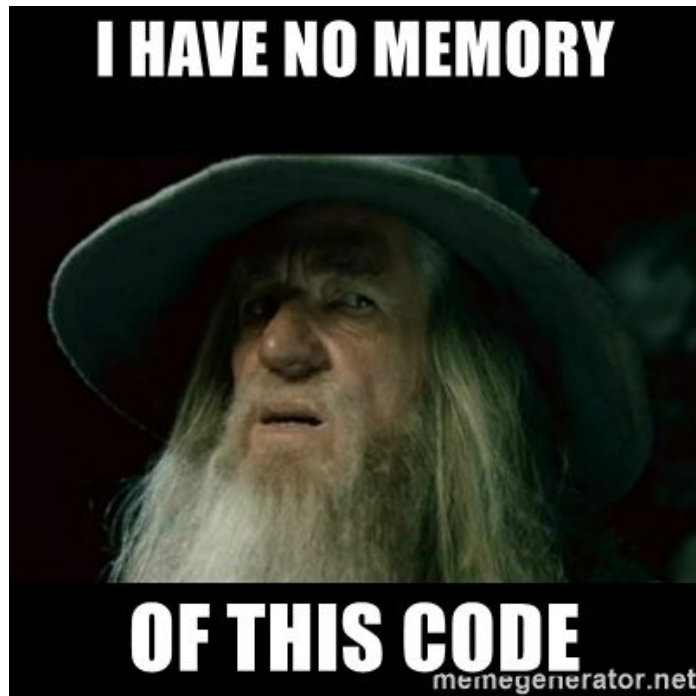
- Crie uma função em que, dados 3 números como parâmetros, permita verificar se a soma de quaisquer par de números gera a soma do terceiro número.
 - Tente não usar condicionais
 - Sua função deve devolver True ou False:

```
def verificaSoma(a, b, c):  
    return a + b == c or a + c == b or b + c == a
```


Organização

Organização

- Algumas frases:
 - *Code is like humor. When you have to explain it, it's bad.* – Cory House
 - *Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live.* – John Woods



Organização

- Algumas coisas já comentamos
 - Organizar o código em funções
 - Dar nomes intuitivos a variáveis e métodos
 - anoNascimento
 - ano_nascimento

Organização

- Documentação nos comentários

```
"""  
Isso é uma string multi-linha, perfeita para  
- fazer comentários longos sobre o código  
"""  
  
minha_str = """Mas, também podemos usar como uma string normal!  
- Não se assuste se ver isso em um código.  
- Ela permite que o texto seja escrito em várias linhas,  
mantendo a formatação!"""  
print(minha_str)
```

Organização

- Documentação nos comentários
 - Descrição breve
 - Descrição longa
 - Embasamento para codificar a solução
 - Entrada
 - Saída
 - Estado atual
 - Funcionalidades que ainda devem ser implementadas
 - Limitações
 - Autor

Organização

- Uso do método *main*
 - Para evitar instruções jogadas no meio do código
 - Concentrar o procedimento principal no método *main*
 - A única instrução que deve ficar fora de algum método é a chamada para o método *main*

Organização

```
def xCedulas (cedula, quantia):  
    x = quantia // cedula  
    restante = quantia % cedula  
    return x, restante  
  
valor = int(input("Informe o valor (múltiplos de 5 reais): "))  
  
notas100, valor = xCedulas(100, valor)  
notas50, valor = xCedulas(50, valor)  
notas20, valor = xCedulas(20, valor)  
notas10, valor = xCedulas(10, valor)  
notas05, valor = xCedulas(5, valor)  
  
print("Você receberá " + str(notas100) + " notas de 100, "  
      + str(notas50) + " notas de 50, " + str(notas20)  
      + " notas de 20, " + str(notas10) + " notas de 10 e "  
      + str(notas05) + " notas de 5")
```

Organização

```
def xCedulas(cedula, quantia):  
    x = quantia // cedula  
    restante = quantia % cedula  
    return x, restante  
  
def main():  
    valor = int(input("Informe o valor (apenas múltiplos de 5): "))  
  
    notas100, valor = xCedulas(100, valor)  
    notas50, valor = xCedulas(50, valor)  
    notas20, valor = xCedulas(20, valor)  
    notas10, valor = xCedulas(10, valor)  
    notas05, valor = xCedulas(5, valor)  
  
    print("Você receberá " + str(notas100) + " nota(s) de 100, " + str(notas50)  
    + " nota(s) de 50, " + str(notas20) + " nota(s) de 20, " + str(notas10)  
    + " nota(s) de 10 e " + str(notas05) + " nota(s) de 5")  
  
main()
```


Referências

- Material do prof. Jesús P Mena-Chalco (UFABC)
- Material do prof. Thiago Covões (UFABC)