Introdução

Python Básico

Mauricio Ferste

Roteiro

Primeiros Passos

Primeiros Comandos em

Python Tipos e Variáveis

Operadores

Conversões de Tipos

Entrada de Dados

Primeiros Passos

Versão do Python

- Nesse curso, aprenderemos como programar utilizando a versão 3 da linguagem Python.
- Você pode verificar a versão do Python instalada no seu computador abrindo o terminal e digitando o comando:

```
python3 --version
```

- A resposta esperada para o comando deve ser:

```
Python 3.x.x
```

Ambiente Interativo do Python

- Nesse ambiente, é possível fornecer um comando ou bloco de comandos e verificar o resultado da execução.
- Para abrir o ambiente interativo basta digitar no terminal:

```
python3
```

- Quando o ambiente interativo é carregado algumas informações são exibidas e o Python fica aguardando algum comando para ser executado:

```
>>>
```

Primeiros Comandos em Python

Imprimindo no Console

- A função `print` é responsável por imprimir uma mensagem.
- A função `print` pode ser utilizada para informar o usuário sobre:
  - A resposta de um processamento.
  - O andamento da execução do programa.
  - Comportamentos inesperados do programa.
  - Outros motivos em que o usuário precise ser informado sobre algo.

Imprimindo no Console

- Com o ambiente interativo do Python carregado, também chamado de console, digite o seguinte comando:

```
1 print("Hello world!")
```

- Como resposta desse comando, na linha seguinte do console, deve aparecer a mensagem:

```
1 Hello world!
```

Imprimindo no Console

- Iremos estudar posteriormente como criar nossas próprias funções, mas agora vamos aprender um pouco mais sobre a função `print`.
- Como todas as funções, a sintaxe para a função de impressão começa com o nome da função (que neste caso é `print`), seguida de uma lista de argumentos, incluída entre parênteses.

```
1 print("Argumento 1", "Argumento 2", "Argumento 3")
```

```
1 Argumento 1 Argumento 2 Argumento 3
```

Imprimindo no Console

- Note que, quando informamos mais de um argumento para a função `print`, eles são automaticamente separados por um espaço.

```
print("Hello", "world!")
```

```
Hello world!
```

- Podemos modificar isso utilizando o parâmetro `sep`.

```
print("Hello", "world!", sep = "+")
```

```
Hello+world!
```

Imprimindo no Console

- Os comandos a seguir produzem o mesmo resultado:

```
1 print("Hello world!")
2 print("Hello", "world!")
3 print("Hello", "world!", sep = " ")
```

- Resposta obtida:

```
1
2 Hello world! Hello world! Hello world!
3
```

Imprimindo no Console

- A função `print` imprime automaticamente o caractere de quebra de linha (\n) no fim de cada execução.

```
1 print("Faculdade") print("Algoritmos")
2
```

```
1 Faculdade Algoritmos
2
```

- Também podemos modificar isso utilizando o parâmetro end.

```
1
2 print("Faculdade", end = "")
  print("Algoritmos")
```

```
1 Faculdade Algoritmos
```

Imprimindo no Console

- Sem o caractere de controle de quebra de linha (\n) no fim:

```
print("Algoritmos", "Faculdade", "2021", sep = " - ",
end = "!")
print("Novo Texto!")
```

```
Algoritmos - Faculdade - 2021! Novo Texto!
```

- Com o caractere de controle de quebra de linha (\n) no fim:

```
print("Algoritmos", "Faculdade", "2021", sep = " - ",
end = "!\n")
print("Novo Texto!")
```

```
Algoritmos - Faculdade - 2021! Novo Texto!

```

Comentários

- Em Python é possível adicionar um comentário utilizando o caractere #, seguido pelo texto desejado.
- Os comentátios não são interpretados pela linguagem, isso significa que todo texto após o caractere # é desconsiderado.
- Exemplo:

```
1 print("Hello world!") # Exemplo de função print
```

- Como resposta para o código acima obtemos apenas:

```
1 Hello World!
```

Comentários

- Vantagens de comentar o seu código:
  - Comentários em trechos mais complexos do código ajudam a explicar o que está sendo realizado em cada passo.
  - Torna mais fácil para outras pessoas que venham a dar manutenção no seu código ou mesmo para você relembrar o que foi feito.

```
1 # Parâmetros importantes da função print
2 # sep: Texto usado na separação dos argumentos
3 recebidos. # end: Texto impresso no final da execução da
4 função. print("Algoritmos", "Faculdade", sep = " - ", end
5 = "!\n")
```

Comentários

- O caractere # é utilizado para comentar um única linha.
- É possível comentar múltiplas linhas utilizando a sequência de caracteres ''' no início e no fim do trecho que se deseja comentar.

```
'''
Parâmetros importantes da função print
sep: Texto usado na separação dos argumentos recebidos.
end: Texto impresso no final da execução da função.
'''
print("Algoritmos", "Faculdade", sep = " - ", end = "!\
n") # Algoritmos - Faculdade!
```

Exercício

Descrição

Escreva um comando utilizando a função `print` que informe seu
primeiro nome, seu RA, o código da disciplina e o ano
seguindo o formato: {nome} - {RA} - {código da disciplina} - {ano}.

Exemplo

- Nome: José
- RA: 999999
- Código da disciplina: Algoritmos
- Ano: 2021

```
# ?
# José - 999999 - Algoritmos - 2021
```

Exercício

Descrição

Escreva um comando utilizando a função `print` que informe seu
primeiro nome, seu RA, o código da disciplina e o ano seguindo o formato: {nome} - {RA} - {código da disciplina} - {ano}.

Exemplo

- Nome: José
- RA: 999999
- Código da disciplina: Algoritmos
- Ano: 2021

```
print("José - 999999 - Algoritmos - 2021") # José - 999999 - Algoritmos - 2021
```

Tipos e Variáveis

Tipos

- Em Python existem diferentes tipos de dados.
- Podemos ter dados no formato:
  - Numérico.
  - Textual.
  - Lógico.
- Para isso, em Python, temos alguns tipos:

`int` Números inteiros (Exemplos: `-3`, `7`, `0`, `2020`).
`float` Números reais (Exemplos: `-3.2`, `1.5`, `1e-8`, `3.2e5`).
`str` Cadeia de caracteres/Strings (Exemplos: "`Faculdade`" e "`Algoritmos`"). `bool` Valores boleanos: `True` (Verdadeiro) e `False` (Falso).

Tipos

- A função `type` pode ser utilizada para mostrar o tipo de um dado.
- Essa função recebe um argumento que terá o tipo identificado.
- Como resposta, a função informa o tipo do dado fornecido como argumento.
- Exemplo da estrutura da função:

```
type(<argumento>)
```

Exemplos de Tipos

```
type(10)

# <class
'int'>
```

```
print(type
(10.0))

# <class
```

```
print(type(True), type(False), type("True"), type
("False")) # <class 'bool'> <class 'bool'> <class
'str'> <class 'str'>
```

```
print(type        type("10.0"))
("10"),

# <class 'str'> <class 'str'>
```

Variáveis

- Ao escrevermos um código, surge a necessidade de armazenarmos valores de maneira temporária, para isso temos as variáveis.
- Em Python, o caractere = é utilizado para atribuir um valor a uma variável.
- Exemplo:

```
1 pi = 3.1416
2 print(pi) # 3.1416
3
```

Variáveis

- Também é possivel, utilizando o caractere =, atribuir um mesmo valor para múltiplas variáveis num único comando.
- Exemplo:

```
a = b = c = 3 print(a, b, c) # 3 3 3
```

- É possivel também atribuir valores diferentes para múltiplas variáveis com um único comando.
- Exemplo:

```
a, b, c = 1, 2, 3
print(a, b, c) # 1 2 3
```

Regras para Nomes de Variáveis

- Nomes de variáveis devem começar com uma letra (maiúscula ou minúscula) ou um subscrito (_).
- Nomes de variáveis podem conter letras maiúsculas, minúsculas, números ou subscritos.
- Cuidado: a linguagem Python é *case sensitive*, ou seja, ela diferencia letras maiúsculas de minúsculas.
- Por exemplo, as variáveis `c1` e `C1` são consideradas diferentes:

```
1 c1 = 0
2 C1 = "1"
3 print(c1, type(c1), C1, type(C1)) # 0 <class 'int'>
4 1 <class 'str'>
```

Exemplos de Variáveis

- Exemplo de variáveis do tipo **int** e **float**:

```
nota_1 = 10
nota_2 = 7.8
nota_final = 8.75
```

```
print(nota_1, type(nota_1)) # 10 <class 'int'>

```

```
print(nota_2, type(nota_2)) # 7.8 <class 'float'>

```

```
print(nota_final, type(nota_final)) # 8.75 <class
'float'>
```

Exemplos de Variáveis

- Exemplo de variáveis do tipo **str**:

```
1 Faculdade = "Centro Universitário das Indústrias"
2 print(Faculdade, type(Faculdade))
3 # Centro Universitário das Indústrias <class 'str'>
```

```
1 disciplina = "algoritmos"
2 print(disciplina, type(disciplina)) # Disciplina <class
3 'str'>
```

Exemplos de Variáveis

- Exemplo de variáveis do tipo **bool**:

```
1 verdadeiro = True
2 falso = False
3 print(verdadeiro, type(verdadeiro), falso, type
4 (falso)) # True <class 'bool'> False <class 'bool'>
```

Operadores

Operadores Matemáticos - Adição

```
1 + 1
# 2
```

```
1.5 +                                   2
# 3.5
```

```
a = 10
a + 10
# 20
```

4
30

Operadores Matemáticos - Subtração

```
5 - 1.5
# 3.5
```

```
a = 100
a - 50
# 50
```

```
a = 1000
b = 0.1
b - a
# -999.9
a - b
# 999.9
```

Operadores Matemáticos - Multiplicação

```
11 * 13
# 143
```

```
2.5 * 2.5
# 6.25
```

```
3 * 0.5
# 1.5
```

```
a = 11
b = 17
a * b
# 187
```

Operadores Matemáticos - Divisão

- Divisão:

```
7 / 2
# 3.5
```

```
a = 10
a / 7
# 1.4285714285714286
```

- Divisão Inteira:

```
7 // 2
# 3
```

```
a = 10
a // 3.4
# 2.0
```

Operadores Matemáticos - Exponenciação

```
2 ** 2 # 4

```

```
a = 10
2 ** a # 1024 a ** 2 # 100




```

```
2.5 ** 3.5
# 24.705294220065465
```

```
3.5 ** 2.5
# 22.91765149399039
```

Operadores Matemáticos - Módulo

- Módulo: resto da divisão inteira

```
57 % 13
# 5
```

```
3 % 2
# 1
```

```
5.5 % 2
# 1.5
```

```
5 % 1.5
# 0.5
```

Atualizações Compactas

- Para os operadores matemáticos, é possível utilizar uma forma compacta para atualizar o valor de uma variável.
- `x += y` é equivalente a `x = x + y`.
- `x -= y` é equivalente a `x = x - y`.
- `x *= y` é equivalente a `x = x * y`.
- `x /= y` é equivalente a `x = x / y`.
- `x %= y` é equivalente a `x = x % y`.

Atualizações Compactas

```
a = 100
a += 50
print(a) # 150
```

```
a -= 75
print(a) # 75
```

```
a*= 3 print(a) # 225
```

```
a /= 15 print(a) # 15.0
```

```
a %= 4
print(a) # 3.0
```

Operadores Matemáticos - Ordem de Precedência

- Precedência é a ordem na qual os operadores serão avaliados quando o programa for executado. Em Python, os operadores são avaliados na seguinte ordem de precedência:
  - Exponenciação.
  - Multiplicação e divisão (na ordem em que aparecem).
  - Módulo.
  - Adição e subtração (na ordem em que aparecem).
- Podemos controlar a ordem com que as expressões são avaliadas com o uso de parênteses.
- Procure usar sempre parênteses em expressões para deixar claro em qual ordem a mesma deve ser avaliada.

Exemplos de Precedência de Operadores Matemáticos

```
print(2 ** 2 % 2)
# 0
```

```
print(2 ** (2 % 2))
# 1
```

```
print(2 + 2 / 2)
# 3.0
```

```
print((2 + 2) / 2)
# 2.0
```

Exercício

Análise de Expressões

Analise as expressões a seguir e escreva a resposta para cada uma delas com base na ordem de precedência:

- Exponenciação.
- Multiplicação e divisão.
- Módulo.
- Adição e subtração.

```
1  print(2 * 2 +       2)
   8 / 2 **

2  # ?
```

```
1  print(100 - 99 /
   3 % 2)

2  # ?
```

Exercício

Análise de Expressões

Analise as expressões a seguir e escreva a resposta para cada uma delas com base na ordem de precedência:

- Exponenciação.
- Multiplicação e divisão.
- Módulo.
- Adição e subtração.

```
1  print(2 * 2 +       2)
   8 / 2 **
2  # 6.0
```

```
1  print(100 - 99 /
   3 % 2)
2  # ?
```

Exercício

Análise de Expressões

Analise as expressões a seguir e escreva a resposta para cada uma delas com base na ordem de precedência:

- Exponenciação.
- Multiplicação e divisão.
- Módulo.
- Adição e subtração.

```
1  print(2 * 2 +       2)
   8 / 2 **

2  # 6.0
```

```
1  print(100 - 99 /
   3 % 2)
```

Erros Comuns com Operadores Matemáticos

- Divisão por zero:

```
10 / 0
# ZeroDivisionError: division by zero
```

```
10 / 0.0
# ZeroDivisionError: float division by zero
```

```
2 // 0
# ZeroDivisionError: integer division or modulo by zero
```

```
2 // 0.0
# ZeroDivisionError: float divmod()
```

Erros Comuns com Operadores Matemáticos

- Resto da divisão por zero:

```
10 % 0

# ZeroDivisionError: integer division or modulo by zero
```

```
10 % 0.0

# ZeroDivisionError: float modulo
```

Erros Comuns com Operadores Matemáticos

```
3 + * 3
# SyntaxError: invalid syntax
```

```
2 + % 3
# SyntaxError: invalid syntax
```

```
5 - / 2
# SyntaxError: invalid syntax
```

Quais os Resultados destas Operações?

```
3 * + 3
# ?
```

```
2 % + 3
# ?
```

```
5 / - 2
```

Quais os Resultados destas Operações?

```
3 * + 3
# 9
```

```
2 % + 3
# 2
```

```
5 / - 2
# -2.5
```

Operadores com Strings - Concatenação

```
"Hello" +" World" #'Hello World'

```

```
universidade = "Universidade" + " das" + "
Industrias"
print(universidade)
# Universidade Estadual de Campinas
```

```
nome = "Fulano"
mensagem = ", você está na turma de algoritmos!"
print(nome + mensagem)
#  Fulano, você está na turma de ????
```

Operadores com Strings - Replicação

```
"ABC" * 3
#  'ABCABCABC'
```

```
print(4 * "Faculdade ")
#
```

```
letra ="Z" n = 10
print(letra * n) # ZZZZZZZZZ


```

Operadores com Strings - Ordem de Precedência

- A ordem de precedência dos operadores com strings é a seguinte:
  - Replicação
  - Concatenação
- Podemos controlar a ordem com que as expressões são avaliadas com o uso de parênteses.
- Exemplos:

```
"a" + "b" * 3 # abbb

```

```
("a" + "b") * 3
# ababab
```

Strings vs. Números

```
4 + 5
# 9
```

```
"4" + "5"
# '45'
```

```
"4" + 5
# TypeError: can only concatenate str (not "int") to str
```

```
4 + "5"
# TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Comparações Numéricas

```
5 < 4
# False
```

```
5 > 4
# True
```

```
5 <= 4
# False
```

```
5 <= 5
# True
```

```
5 >= 4
# True
```

Comparações Numéricas

```
5 != 4
# True
```

```
5 == 4
# False
```

```
5 == 5.0
# True
```

```
5 == 5.000001
# False
```

```
5 == "5"
# False
```

Comparações de Strings

- Ordem considerada para os caracteres do alfabeto:
  - ABC...XYZabc...xyz

```
"a" > "b"
# False
```

```
"a" < "b"
# True
```

```
"a" == "a"
# True
```

```
"a" == "A"
# False
```

Comparações de Strings

```
"A" < "a"
# True
```

```
"A" > "a"
# False
```

```
"Z" < "a"
# True
```

```
"z" < "a"
# False
```

Comparações de Strings

```
"Araraquara" < "Araras" # True

```

```
"Maria" < "Maria Clara" # True

```

```
"maria" < "Maria Clara" # False

```

```
"Marvel" > "DC" # True

```

Comparações de Strings

- Para obter a ordem relativa de outros caracteres, consulte a Tabela ASCII:
  - https://pt.wikipedia.org/wiki/ASCII

```
"sen    > "s3nh4"
ha"

#
True
```

```
"aa     >= "aaaa"
aa"

#
Fals
e
```

Operadores Lógicos - E (and)

```
1  True      True
   and

2  #
   True
```

```
1  True      False
   and

2  #
   False
```

```
1  False      True
   and

2  #
```

Operadores Lógicos - E (and)

```
(3 < 4) and ("banana" > "abacaxi") # True

```

```
(4 == 4.0) and (4 == "4")
# False
```

```
(4 < 4) and ("aaaa" >= "aaa") # False

```

```
(3 >= 4) and ("casa" > "peixe") # False

```

Operadores Lógicos - OU (or)

```
True     True
or

#
True
```

```
True     False
or

#
True
```

```
False     True
 or

#
```

Operadores Lógicos - OU (or)

```
1  (3 <      ("banana" > "abacaxi")
   4) or
2  # True
```

```
1  (4 ==     or (4 == "4")
   4.0)
2  # True
```

```
1 (3 >= 4) or ("casa" > "peixe") # False
2
```

```
1  (4 <      ("aaaa" >= "aaa")
   4) or
2  # True
```

Operadores Lógicos - Negação (not)

```
not True
# False
```

```
not False
# True
```

```
not True and False
# False
```

```
not (True and False)  # True

```

Operadores Lógicos - Negação (not)

```
1  not (4 < 5)
2  # False
```

```
1  not ("amor" >
   "dinheiro")
2  # True
```

```
1
2
```

Operadores Lógicos - Ordem de Precedência

- A ordem de precedência dos operadores lógicos é a seguinte:
  - not
  - and
  - or
- Podemos controlar a ordem com que as expressões são avaliadas com o uso de parênteses.
- Exemplos:

```
a = 7
print(a > 5 or a < 0 and a != 7) # True

```

```
print((a > 5 or a < 0) and a != 7) # False

```

Operadores Lógicos Preguiçosos

- Os operadores lógicos `and` e `or` são classificados como preguiçosos (*lazy*).
- Os operadores recebem essa classificação pois eventualmente somente alguns valores da expressão serão verificados para determinar o seu resultado final (`True` ou `False`).
- As expressões lógicas são avaliadas seguindo a ordem de precedência entre os operadores, da esqueda para direita.

Operadores Lógicos Preguiçosos

- Os operadores lógicos preguiçosos podem trazer um melhor desempenho computacional, uma vez que:
  - O operador lógico `and` necessita apenas que um dos valores da expressão seja falso para que ela seja considerada falsa.
  - O operador lógico `or` necessita apenas que um dos valores da expressão seja verdadeiro para que ela seja considerada verdadeira.

Operadores Lógicos Preguiçosos

- Exemplos:

```
1 x = 3
2 y = 0
3 print(x / y)
4 # ZeroDivisionError: division by zero
5 print((y != 0) and (x / y)) # False
6
```

```
1 print(teste)
2 # NameError: name 'teste' is not defined
3 print((x > y) or teste) # True
4
```

Operadores Lógicos Não Preguiçosos

- Os operadores lógicos E e OU também possuem uma versão não preguiçosa.
- Operador E não preguiçoso: &.
- Operador OU não preguiçoso: |.
- Ao utilizar operadores não preguiçosos todos os valores da expressão são avaliados independentemente se é possível determinar o valor final da expressão utilizando somente alguns deles.

Operadores Lógicos Preguiçosos

- Exemplos:

```
x = 3
y = 0
print(x / y)
# ZeroDivisionError: division by zero
print((y != 0) & (x / y))
# ZeroDivisionError: division by zero
```

```
print(teste)
# NameError: name 'teste' is not defined
print((x > y) | teste)
# NameError: name 'teste' is not defined
```

Conversões de Tipos

Conversões de Tipos

- Alguns tipos de dados permitem que o seu valor seja convertido para outro tipo (cast).
- Para isso temos algumas funções:
  - `int()` converte o valor para o tipo `int` (número inteiro).
  - `float()` converte o valor para o tipo `float` (número real).
  - `str()` converte o valor para o tipo `str` (string).

Exemplos de Conversões de Tipos

- Convertendo uma string para um número inteiro:

```
a = "45"
b = int(a) print(a, type(a)) # 45 <class 'str'>
print(b, type(b)) # 45 <class 'int'>
```

Exemplos de Conversões de Tipos

- Convertendo uma string para um número real:

```
1 a = "4.5"
2 b = float(a) print(a, type(a))
3 # 4.5 <class 'str'>
4 print(b, type(b))
5 # 4.5 <class 'float'>
6
```

Exemplos de Conversões de Tipos

- Nem toda string pode ser convertida para valores numéricos:

```
a = "Faculdade"
int(a)
# ValueError: invalid literal for int() with base 10
float(a)
# ValueError: could not convert string to float: 'Faculdade'
```

Exemplos de Conversões de Tipos

- Convertendo um número inteiro e um número real para string:

```
1 a = 45
2 b = str(a) print(a * 3)
3 # 135
4 print(b * 3)
5 # 454545
6
```

Exemplos de Conversões de Tipos

- Convertendo valores numéricos:

```
1 a = 3.3
2 b = int(a) print(b)
3 # 3
4 a = float(b) print(a)
5 # 3.0
6
7
```

Exercício

Hipotenusa de um Triângulo (versão 1)

Escreva um código que calcule a hipotenusa de um triângulo retângulo,
cujos catetos são $a = 6$ e $b = 8$. Note que $\sqrt{x} = x^{\frac{1}{2}}$.

- Rascunho:

```
1
2
3
4
```

```
a = 6
b = 8
# c = ?
print(c)
```

Exercício

Hipotenusa de um Triângulo (versão 1)

Escreva um código que calcule a hipotenusa de um triângulo retângulo cujos catetos são $a = 6$ e $b = 8$. Note que $\sqrt{x} = x^{\left(\frac{1}{2}\right)}$.

- Possível resposta:

```
1 a = 6
2 b = 8
3 c = ((a * a) + (b * b)) ** (1/2)
4 print(c)
```

Entrada de Dados

Recebendo Dados do Usuário

- A função `input` é responsável por receber dados do usuário.
- O usuário deve escrever algo e pressionar a tecla <enter>.
- Normalmente, armazenamos o valor lido em uma variável.
- A função `input` obtém os dados fornecidos pelo console no formato de string (`str`).
- Devemos fazer uma conversão dos dados se quisermos trabalhar com números.

Exemplos de Entrada de Dados

- Sintaxe da função input:

```
x = input("Mensagem opcional")
```

- Armazenando os valores lidos nas variáveis a e b:

```

a = input("Digite um valor para a variável a: ") b = input("Digite um valor para a variável b: ") print(int(a) + float(b))
```

Exercício

Hipotenusa de um Triângulo (versão 2)

Modifique o exercício anterior para receber os valores dos catetos `a` e `b`
pelo console. Lembre-se de converter os valores para um tipo numérico antes de efetuar o cálculo da hipotenusa.

- Rascunho:

```
1 # a_str = ? # b_str = ? # a = ?
2 # b = ?
3 c = ((a * a) + (b * b)) ** (1/2)
4 print(c)
5
6
```

Exercício

Hipotenusa de um Triângulo (versão 2)

Modifique o exercício anterior para receber os valores dos catetos a e b
pelo console. Lembre-se de converter os valores para um tipo numérico antes de efetuar o cálculo da hipotenusa.

- Possível resposta:

```
1 a_str = input("Digite um valor para o cateto a: ") b_str
2 = input("Digite um valor para o cateto b: ") a =
3 float(a_str)
4 b = float(b_str)
5 c = ((a * a) + (b * b)) ** (1/2)
6 print(c)
```

Exercício

Inteiros com Paridades Distintas

Escreva um programa que leia dois números inteiros e imprima `True`, se os números tiverem paridades distintas, e `False`, caso contrário.

- Rascunho:

```
a = int(input()) b = int(input())


ok = ?
print(ok)

```

Exercício

Inteiros com Paridades Distintas

Escreva um programa que leia dois números inteiros e imprima `True`, se os números tiverem paridades distintas, e `False`, caso contrário.

- Rascunho:

```
a = int(input()) b = int(input())
ok1 = ((a % 2 == 1) and (b % 2 == 0))
ok2 = ((a % 2 == 0) and (b % 2 == 1))
ok = ?
print(ok)

```

Exercício

Inteiros com Paridades Distintas

Escreva um programa que leia dois números inteiros e imprima `True`, se os números tiverem paridades distintas, e `False`, caso contrário.

- Possível resposta:

```
1 a = int(input()) b = int(input())
2 ok1 = ((a % 2 == 1) and (b % 2 == 0))
3 ok2 = ((a % 2 == 0) and (b % 2 == 1))
4 ok = (ok1 or ok2)
5 print(ok)
6
```

Exercício

Inteiros com Paridades Distintas

Escreva um programa que leia dois números inteiros e imprima `True`, se os números tiverem paridades distintas, e `False`, caso contrário.

- Possível resposta:

```
1 a = int(input()) b = int(input())
2
3 ok = (a % 2) != (b % 2)
4
5 print(ok)
6
```

Exercício

Inteiros com Paridades Distintas

Escreva um programa que leia dois números inteiros e imprima `True`, se os números tiverem paridades distintas, e `False`, caso contrário.

- Possível resposta:

```
1 a = int(input()) b = int(input())
2
3 ok = ((a + b) % 2 == 1)
4
5 print(ok)
6
```

Fim

- Fim