

**Instituto de  
Computação**

UNIVERSIDADE ESTADUAL DE CAMPINAS



**MC102 - Aula 02**

**Python Básico**

Algoritmos e Programação de Computadores

Turmas

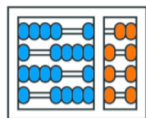
**OVXZ**

**Prof. Lise R. R. Navarrete**

[lrommel@ic.unicamp.br](mailto:lrommel@ic.unicamp.br)

Terça-feira, 22 de março de 2022

21:00h - 23:00h (CB06)



**Instituto de  
Computação**

UNIVERSIDADE ESTADUAL DE CAMPINAS



UNICAMP

**MC102** – Algoritmos e Programação de Computadores

---

Turmas

**OVXZ**

<https://ic.unicamp.br/~mc102/>

Site da Coordenação de MC102

Aulas teóricas:

Terça-feira, 21:00h - 23:00h (CB06)

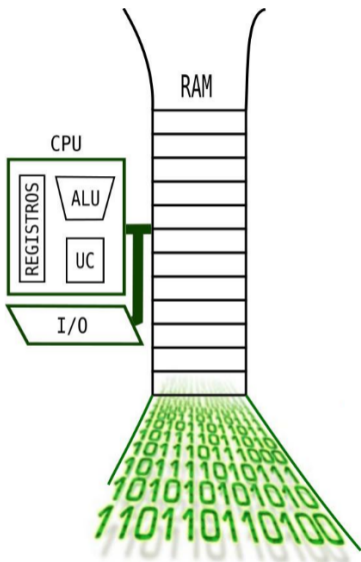
Quinta-feira, 19:00h - 21:00h (CB06)

# Conteúdo

- Primeiros Passos
- Primeiros Comandos em Python
- Tipos de Variáveis
- Operadores
- Conversões de Tipos
- Entrada de Dados

# Primeiros Passos

# Programação



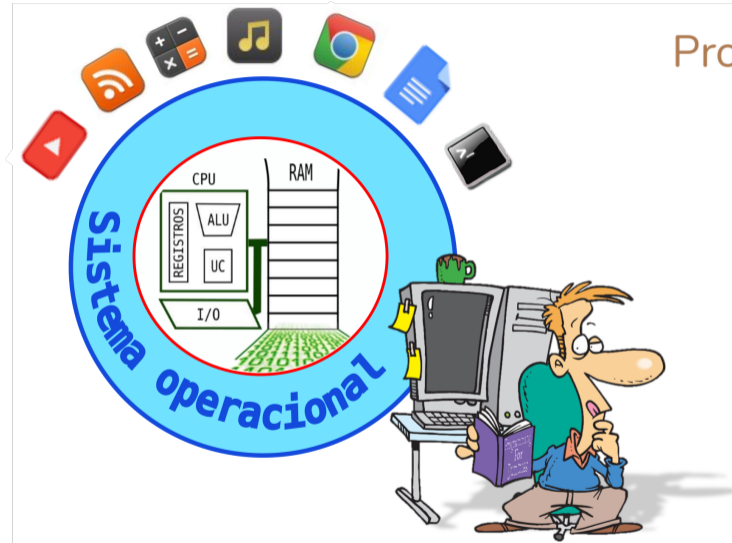
# Programação



# Programação

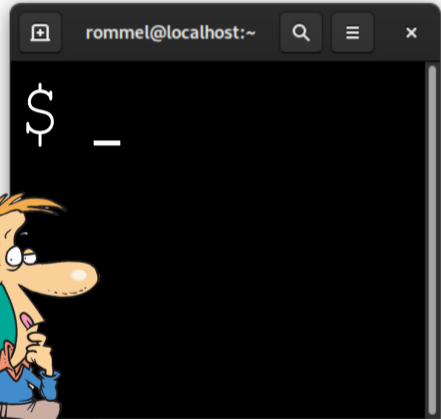


# Programação





# Programação





## Programação

```
rommel@localhost:~
$ pwd ↵
/home/lrommel/
$ _
```



mkdir  
 cat  
 rm  
 cp  
 cd  
 man

# Programação

```
rommel@localhost:~
$ ls
dir1 dir2
$
```



## Programação



```
rommel@localhost:~
$ gedit _
```

Programação

Sistema operacional

rommel@localhost:~

```
$ gedit
```

Documento sin título 1

1

Texto plano Anchura del tabulador: 8 Ln 1, Col 1 INS

- 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:

```
1 python3 --version
```

- A resposta esperada para o comando deve ser:

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

- 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:

```
1 python3
```

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

```
1 >>>
```

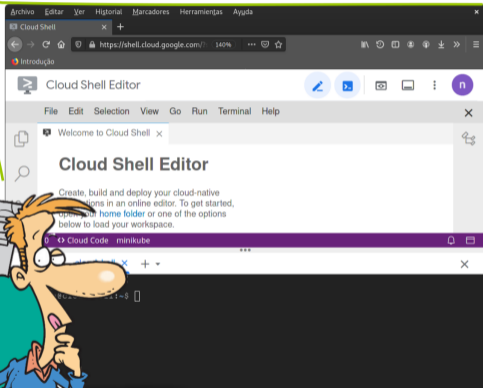
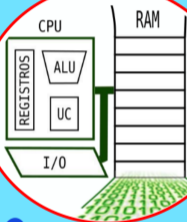
<https://shell.cloud.google.com/>  
Google Cloud Shell

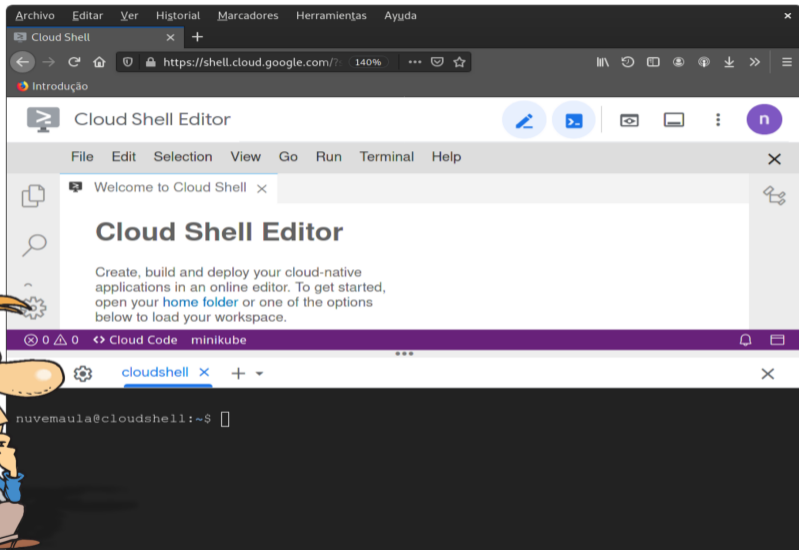




# Programação

Sistema operacional





Archivo Editar Ver Historial Marcadores Herramientas Ayuda

Cloud Shell

https://shell.cloud.google.com/? 140%

Introdução

Cloud Shell Editor

File Edit Selection View Go Run Terminal Help

Welcome to Cloud Shell

## Cloud Shell Editor

Create, build and deploy your cloud-native applications in an online editor. To get started, open your [home folder](#) or one of the options below to load your workspace.

Cloud Code minikube

cloudshell

```
nuvemaula@cloudshell:~$
```



Arquivo Editar Ver Historial Marcadores Herramientas Ayuda

Cloud Shell

https://shell.cloud.google.com/?show=ide%2Ct... 140%

Introdução

Cloud Shell Editor

File Edit Selection View Go Run Terminal Help

Welcome to Cloud Shell

## Cloud Shell Editor

Create, build and deploy your cloud-native applications in an online editor. To get started, open your [home folder](#) or one of the options below to load your workspace.

0 0 Cloud Code minikube

cloudshell

```
nuvemaula@cloudshell:~$
```

Arquivo Editar Ver Historial Marcadores Herramientas Ayuda

Cloud Shell

https://shell.cloud.google.com/?show=ide%2Cte 140%

Introdução

# Cloud Shell Editor

File Edit Selection View Go Run Terminal Help

Welcome to Cloud Shell

## Cloud Shell Editor

Create, build and deploy your cloud-native applications in an online editor. To get started, open your [home folder](#) or one of the options below to load your workspace.

0 0 Cloud Code minikube

cloudshell

```
nuvemaula@cloudshell:~$ ls
README-cloudshell.txt
nuvemaula@cloudshell:~$
```

Arquivo Editar Ver Historial Marcadores Herramientas Ayuda

Cloud Shell

https://shell.cloud.google.com/?show=ide%2Cte 140%

Introdução

# Cloud Shell Editor

File Edit Selection View Go Run Terminal Help

Welcome to Cloud Shell

## Cloud Shell Editor

Create, build and deploy your cloud-native applications in an online editor. To get started, open your [home folder](#) or one of the options below to load your workspace.

0 0 Cloud Code minikube

cloudshell

```
nuvemaula@cloudshell:~$ pwd
/home/nuvemaula
nuvemaula@cloudshell:~$
```

Arquivo Editar Ver Historial Marcadores Herramientas Ayuda

Cloud Shell

https://shell.cloud.google.com/?show=ide%2Cte 140%

Introdução

## Cloud Shell Editor

File Edit Selection View Go Run Terminal Help

Welcome to Cloud Shell

# Cloud Shell Editor

Create, build and deploy your cloud-native applications in an online editor. To get started, open your [home folder](#) or one of the options below to load your workspace.

0 0 Cloud Code minikube

cloudshell

```
nuvemaula@cloudshell:~$ pwd
/home/nuvemaula
nuvemaula@cloudshell:~$ clear
```

Arquivo Editar Ver Historial Marcadores Herramientas Ayuda

Cloud Shell

https://shell.cloud.google.com/?show=ide%2Cte 140%

Introdução

Cloud Shell Editor

File Edit Selection View Go Run Terminal Help

Welcome to Cloud Shell

## Cloud Shell Editor

Create, build and deploy your cloud-native applications in an online editor. To get started, open your [home folder](#) or one of the options below to load your workspace.

0 0 Cloud Code minikube

cloudshell

```
nuvemaula@cloudshell:~$
```

```
nuvemaula@cloudshell:~$ python --version
*****
Python 2 is deprecated. Upgrade to Python 3 as soon as possible.
See https://cloud.google.com/python/docs/python2-sunset

Cloud Shell will soon default to Python 3.

To suppress this warning, create an empty ~/.cloudshell/no-python2-warning file.
The command will automatically proceed in seconds or on any key.
*****
Python 2.7.18
nuvemaula@cloudshell:~$
```



Archivo Editar Ver Historial Marcadores Herramientas Ayuda

Cloud Shell x +

← → ↻ 🏠 🔒 https://shell.cloud.google.com/?show=ide%2Cte 140% ... 📄 ☆

Introdução

Cloud Shell Editor

File Edit Selection View Go Run Terminal Help

Welcome to Cloud Shell x

Cloud Shell Editor

cloudshell x +

```
nuvemaula@cloudshell:~$ python3 --version
Python 3.9.2
nuvemaula@cloudshell:~$
```

Arquivo Editar Ver Historial Marcadores Herramientas Ayuda

Cloud Shell

https://shell.cloud.google.com/?show=ide%2Cte 140%

Introdução

Cloud Shell Editor

File Edit Selection View Go Run Terminal Help

Welcome to Cloud Shell

Cloud Shell Editor

cloudshell

```
nuvemaula@cloudshell:~$ python3
Python 3.9.2 (default, Feb 28 2021, 17:03:44)
[GCC 10.2.1 20210110] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Arquivo Editar Ver Historial Marcadores Herramientas Ayuda

Cloud Shell

https://shell.cloud.google.com/?show=ide%2Cte 140%

Introdução

Cloud Shell Editor

File Edit Selection View Go Run Terminal Help

Welcome to Cloud Shell

Cloud Shell Editor

cloudshell

```
nuvemaula@cloudshell:~$ python3
Python 3.9.2 (default, Feb 28 2021, 17:03:44)
[GCC 10.2.1 20210110] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> A=4
>>> B=A+1
>>> print(B)
5
>>>
```

# Primeiros Comandos em Python

- 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.

- 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!
```

- 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
```

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

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

```
1 Hello world!
```

- Podemos modificar isso utilizando o parâmetro `sep`.

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

```
1 Hello+world!
```



- Os comandos a seguir produzem o mesmo resultado:

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

- Resposta obtida:

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

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

```
1 print("Unicamp")  
2 print("MC102!")
```

```
1 Unicamp  
2 MC102!
```

- Também podemos modificar isso utilizando o parâmetro `end`.

```
1 print("Unicamp", end = "")  
2 print("MC102!")
```

```
1 UnicampMC102!
```

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

```
1 print("MC102", "Unicamp", "2020", sep = " - ", end = "!")  
2 print("Novo Texto!")
```

```
1 MC102 - Unicamp - 2020!Novo Texto!
```

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

```
1 print("MC102", "Unicamp", "2020", sep = " - ", end = "\n")  
2 print("Novo Texto!")
```

```
1 MC102 - Unicamp - 2020!  
2 Novo Texto!
```

- Em Python é possível adicionar um comentário utilizando o caractere `#`, seguido pelo texto desejado.
- Os comentários 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!
```

- 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 recebidos.
3 # end: Texto impresso no final da execução da função.
4 print("MC102", "Unicamp", sep = " - ", end = "!\\n")
5 # MC102 - Unicamp!
```

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

```
1 '''
2 Parâmetros importantes da função print
3 sep: Texto usado na separação dos argumentos recebidos.
4 end: Texto impresso no final da execução da função.
5 '''
6 print("MC102", "Unicamp", sep = " - ", end = "!\n")
7 # MC102 - Unicamp!
```

## 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}.

Observação: Não utilize o parâmetro `sep`.

## Exemplo

- Nome: José
- RA: 999999
- Código da disciplina: MC102
- Ano: 2020

```
1 # ?  
2 # José - 999999 - MC102 - 2020
```

## 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}.

Observação: Não utilize o parâmetro `sep`.

## Exemplo

- Nome: José
- RA: 999999
- Código da disciplina: MC102
- Ano: 2020

```
1 print("José - 999999 - MC102 - 2020")  
2 # José - 999999 - MC102 - 2020
```



## 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}.

Observação: Utilize o parâmetro `sep`.

## Exemplo

- Nome: José
- RA: 999999
- Código da disciplina: MC102
- Ano: 2020

```
1 # ?  
2 # José - 999999 - MC102 - 2020
```

## 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}.

Observação: Utilize o parâmetro `sep`.

## Exemplo

- Nome: José
- RA: 999999
- Código da disciplina: MC102
- Ano: 2020

```
1 print("José", "999999", "MC102", "2020", sep=" - ")  
2 # José - 999999 - MC102 - 2020
```

# Tipos de Variáveis

- 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: `"Unicamp"` e `"MC102"`).
  - `bool` Valores booleanos: `True` (Verdadeiro) e `False` (Falso).

- 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:

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

```
1 print(type(10))  
2 # <class 'int'>
```

```
1 print(type(10.0))  
2 # <class 'float'>
```

```
1 print(type("10"), type("10.0"))  
2 # <class 'str'> <class 'str'>
```

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

- 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 # 3.1416
```

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

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

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

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



- Nomes de variáveis devem começar com uma letra (maiúscula ou minúscula) ou um sublinhado (\_).
- Nomes de variáveis podem conter letras maiúsculas, minúsculas, números ou sublinhado.
- 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))
4 # 0 <class 'int'> 1 <class 'str'>
```

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

```
1 nota_1 = 10
2 nota_2 = 7.8
3 nota_final = 8.75
```

```
1 print(nota_1, type(nota_1))
2 # 10 <class 'int'>
```

```
1 print(nota_2, type(nota_2))
2 # 7.8 <class 'float'>
```

```
1 print(nota_final, type(nota_final))
2 # 8.75 <class 'float'>
```

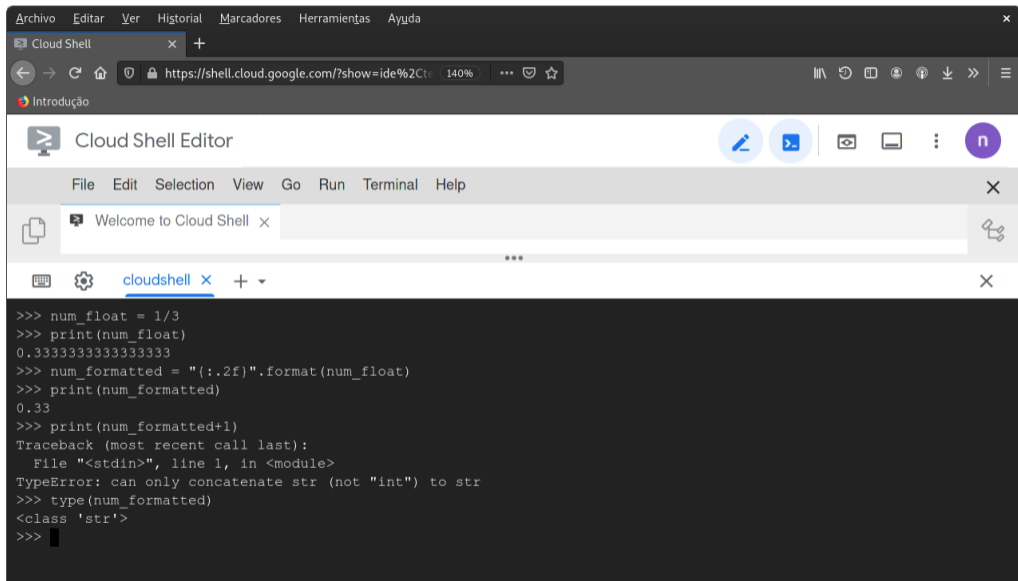
- Exemplo de variáveis do tipo `str`:

```
1 Unicamp = "Universidade Estadual de Campinas"  
2 print(Unicamp, type(Unicamp))  
3 # Universidade Estadual de Campinas <class 'str'>
```

```
1 mc102_2020_1s = "MC102"  
2 print(mc102_2020_1s, type(mc102_2020_1s))  
3 # MC102 <class 'str'>
```

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

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



The screenshot shows a web browser window with the Google Cloud Shell Editor. The browser's address bar shows the URL `https://shell.cloud.google.com/?show=ide%2Cte`. The editor's menu bar includes File, Edit, Selection, View, Go, Run, Terminal, and Help. The terminal window contains the following Python code and output:

```
>>> num_float = 1/3
>>> print(num_float)
0.3333333333333333
>>> num_formatted = "{:.2f}".format(num_float)
>>> print(num_formatted)
0.33
>>> print(num_formatted+1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can only concatenate str (not "int") to str
>>> type(num_formatted)
<class 'str'>
>>>
```

# Operadores

```
1 1 + 1  
2 # 2
```

```
1 1.5 + 2  
2 # 3.5
```

```
1 a = 10  
2 a + 10  
3 # 20
```

```
1 a = 10  
2 b = 20  
3 a + b  
4 # 30
```

```
1 5 - 1.5  
2 # 3.5
```

```
1 a = 100  
2 a - 50  
3 # 50
```

```
1 a = 1000  
2 b = 0.1  
3 b - a  
4 # -999.9  
5 a - b  
6 # 999.9
```



```
1 11 * 13  
2 # 143
```

```
1 2.5 * 2.5  
2 # 6.25
```

```
1 3 * 0.5  
2 # 1.5
```

```
1 a = 11  
2 b = 17  
3 a * b  
4 # 187
```

- Divisão:

```
1 7 / 2
2 # 3.5
```

```
1 a = 10
2 a / 7
3 # 1.4285714285714286
```

- Divisão Inteira:

```
1 7 // 2
2 # 3
```

```
1 a = 10
2 a // 3.4
3 # 2.0
```

```
1 2 ** 2
2 # 4
```

```
1 a = 10
2 2 ** a
3 # 1024
4 a ** 2
5 # 100
```

```
1 2.5 ** 3.5
2 # 24.705294220065465
```

```
1 3.5 ** 2.5
2 # 22.91765149399039
```

- Módulo: resto da divisão inteira.

```
1 57 % 13  
2 # 5
```

```
1 3 % 2  
2 # 1
```

```
1 5.5 % 2  
2 # 1.5
```

```
1 5 % 1.5  
2 # 0.5
```

- 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  $x = x \% y$ .

```
1 a = 100
2 a += 50
3 print(a)
4 # 150
```

```
1 a -= 75
2 print(a)
3 # 75
```

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

```
1 a /= 15
2 print(a)
3 # 15.0
```

```
1 a %= 4
2 print(a)
3 # 3.0
```

- 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.
  - Operadores unários (+ ou -).
  - 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 as mesmas devem ser avaliadas.

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

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

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

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



## Análise de Expressões

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

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

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

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

## Análise de Expressões

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

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

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

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

## Análise de Expressões

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

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

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

```
1 print(100 - 99 / 3 % 2)
2 # 99.0
```

- Divisão por zero:

```
1 10 / 0
2 # ZeroDivisionError: division by zero
```

```
1 10 / 0.0
2 # ZeroDivisionError: float division by zero
```

```
1 2 // 0
2 # ZeroDivisionError: integer division or modulo by zero
```

```
1 2 // 0.0
2 # ZeroDivisionError: float divmod()
```

- Resto da divisão por zero:

```
1 10 % 0
2 # ZeroDivisionError: integer division or modulo by zero
```

```
1 10 % 0.0
2 # ZeroDivisionError: float modulo
```

```
1 3 + * 3
2 # SyntaxError: invalid syntax
```

```
1 2 + % 3
2 # SyntaxError: invalid syntax
```

```
1 5 - / 2
2 # SyntaxError: invalid syntax
```

```
1 -2 * * 2
2 # SyntaxError: invalid syntax
```

```
1 3 * + 3
2 # ?
```

```
1 2 % + 3
2 # ?
```

```
1 5 / - 2
2 # ?
```

```
1 -2 ** 2
2 # ?
```

```
1 3 * + 3  
2 # 9
```

```
1 2 % + 3  
2 # 2
```

```
1 5 / - 2  
2 # -2.5
```

```
1 -2 ** 2  
2 # -4
```



```
>>> 1+++++-1
0
>>> 2+++++-----1
3
>>> 2---4++1
-1
>>> █
```

```
1 "Hello" + " World"
2 # 'Hello World'
```

```
1 Unicamp = "Universidade" + " Estadual" + " de Campinas"
2 print(Unicamp)
3 # Universidade Estadual de Campinas
```

```
1 nome = "Fulano"
2 mensagem = ", você está na turma de MC102!"
3 print(nome + mensagem)
4 # Fulano, você está na turma de MC102!
```

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

```
1 print(4 * "Unicamp ")
2 # Unicamp Unicamp Unicamp Unicamp
```

```
1 letra = "Z"
2 n = 10
3 print(letra * n)
4 # ZZZZZZZZZZ
```

- 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:

```
1 "a" + "b" * 3  
2 # 'abbb'
```

```
1 ("a" + "b") * 3  
2 # 'ababab'
```

```
1 4 + 5  
2 # 9
```

```
1 "4" + "5"  
2 # '45'
```

```
1 "4" + 5  
2 # TypeError: can only concatenate str (not "int") to str
```

```
1 4 + "5"  
2 # TypeError: unsupported operand type(s) for +: 'int' and  
   'str'
```

```
1 5 < 4  
2 # False
```

```
1 5 > 4  
2 # True
```

```
1 5 <= 4  
2 # False
```

```
1 5 <= 5  
2 # True
```

```
1 5 >= 4  
2 # True
```

```
1 5 != 4  
2 # True
```

```
1 5 == 4  
2 # False
```

```
1 5 == 5.0  
2 # True
```

```
1 5 == 5.000001  
2 # False
```

```
1 5 == "5"  
2 # False
```

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

```
1 "a" > "b"  
2 # False
```

```
1 "a" < "b"  
2 # True
```

```
1 "a" == "a"  
2 # True
```

```
1 "a" == "A"  
2 # False
```



```
1 "A" < "a"  
2 # True
```

```
1 "A" > "a"  
2 # False
```

```
1 "Z" < "a"  
2 # True
```

```
1 "z" < "a"  
2 # False
```

```
1 "Araraquara" < "Araras"  
2 # True
```

```
1 "Maria" < "Maria Clara"  
2 # True
```

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

```
1 "Marvel" > "DC"  
2 # True
```

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

```
1 "senha" > "s3nh4"  
2 # True
```

```
1 "aa aa" >= "aaaa"  
2 # False
```

```
1 "@mor" < "amor"  
2 # True
```

```
1 "21+7" < "2+31"  
2 # False
```

```
1 True and True  
2 # True
```

```
1 True and False  
2 # False
```

```
1 False and True  
2 # False
```

```
1 False and False  
2 # False
```

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

```
1 (4 == 4.0) and (4 == "4")  
2 # False
```

```
1 (4 < 4) and ("aaaa" >= "aaa")  
2 # False
```

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

```
1 True or True  
2 # True
```

```
1 True or False  
2 # True
```

```
1 False or True  
2 # True
```

```
1 False or False  
2 # False
```

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

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

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

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

```
1 not True  
2 # False
```

```
1 not False  
2 # True
```

```
1 not True and False  
2 # False
```

```
1 not (True and False)  
2 # True
```



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

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

```
1 not ("MA111" < "MC102") and ("Noobmaster69" > "Thor")
2 # False
```

```
1 not(("MA111" < "MC102") and ("Noobmaster69" > "Thor"))
2 # True
```

- 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:

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

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

- 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 esquerda para direita.

- 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.

- Exemplos:

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

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

- 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.

- Exemplos:

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

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

# Conversões de Tipos



- Alguns tipos de dados permitem que o seu valor seja convertido para outro tipo (cast).
- Para isso, podemos usar as seguintes 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).
  - `bool()` converte o valor para o tipo `bool` (booleano).

- Convertendo uma string para um número inteiro:

```
1 a = "45"  
2 b = int(a)  
3 a  
4 # '45'  
5 type(a)  
6 # <class 'str'>  
7 b  
8 # 45  
9 type(b)  
10 # <class 'int'>
```

- Convertendo uma string para um número real:

```
1 a = "4.5"  
2 b = float(a)  
3 a  
4 # '4.5'  
5 type(a)  
6 # <class 'str'>  
7 b  
8 # 4.5  
9 type(b)  
10 # <class 'float'>
```

- Nem toda string pode ser convertida para um valor numérico:

```
1 a = "MC102"  
2 int(a)  
3 # ValueError: invalid literal for int() with base 10  
4 float(a)  
5 # ValueError: could not convert string to float: 'MC102'
```

- Convertendo valores numéricos:

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

- Valores numéricos ou booleanos podem ser convertidos para strings:

```
1 str(102)
2 # '102'
3 str(3.1416)
4 # '3.1416'
5 str(True)
6 # 'True'
7 str(False)
8 # 'False'
```

- Convertendo um número inteiro numa string:

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

- Qualquer número, com exceção do 0 (zero), quando convertido para booleano resulta em **True**:

```
1 bool(-1.2)
2 # True
3 bool(0)
4 # False
5 bool(0.5)
6 # True
7 bool(2)
8 # True
```



- Qualquer string, com exceção da string vazia (" "), quando convertida para booleano resulta em **True**:

```
1 bool("MC102")
2 # True
3 bool("")
4 # False
5 bool("Programação de Computadores")
6 # True
7 bool("X")
8 # True
```

- Valores booleanos podem ser convertidos para números:

```
1 int(True)
2 # 1
3 int(False)
4 # 0
5 float(True)
6 # 1.0
7 float(False)
8 # 0.0
```

- Valores booleanos podem ser usados em operações aritméticas, sem necessidade de conversão explícita:

```
1 False + False  
2 # 0  
3 True + False  
4 # 1  
5 True + True  
6 # 2  
7 5.2 + (3 < 7)  
8 # 6.2
```

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

## 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  $\sqrt{x} = x^{(1/2)}$ .

- Possível resposta:

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

# Entrada de Dados

- 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.

- Sintaxe da função `input`:

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

- Armazenando os valores lidos nas variáveis `a` e `b`:

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



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

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

## 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:

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

## 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:

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

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

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

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



EXPLORER

... exemplo01.py x

&gt; OPEN EDITORS

v NUVEMAULA

+ exemplo01.py

README-cloudshell.txt



```
exemplo01.py > ...  
1 a = int(input())  
2 b = int(input())  
3 print((a+b)%2==1)
```

Problems Python x

```
3  
4  
True  
nuvemaula@cloudshell:~$
```



cloudshell x



&gt;&gt;&gt;





EXPLORER



+ exemplo01.py x



&gt; OPEN EDITORS



v NUVEMAULA



+ exemplo01.py



README-cloudshell.txt



```
exemplo01.py > ...  
1 a = int(input())  
2 b = int(input())  
3 print((a+b)%2==1)
```

Problems Python x

```
3  
4  
True  
nuvemaula@cloudshell:~$
```



cloudshell x



```
nuvemaula@cloudshell:~$ python3 exemplo01.py
```

```
3  
4  
True  
nuvemaula@cloudshell:~$
```

EXPLORER

- OPEN EDITORS
- NUVEMAULA
  - dados.in
  - exemplo01.py
  - README-cloudshell.txt

exemplo01.py dados.in x

```
dados.in
1 3
2 4
3
```

0 0 Cloud Code minikube Ln 3, Col 1 LF UTF-8 Spaces: 4 Plain Text

cloudshell x +

```
nuvemaula@cloudshell:~$ python3 exemplo01.py <dados.in
True
nuvemaula@cloudshell:~$
```

# Perguntas ....

# Referências

- Zanoni Dias, MC102, Algoritmos e Programação de Computadores, IC/UNICAMP, 2021. <https://ic.unicamp.br/~mc102/>
  - Aula Introdutória [ [slides](#) ] [ [vídeo](#) ]
  - Primeira Aula de Laboratório [ [slides](#) ] [ [vídeo](#) ]
  - Python Básico: Tipos, Variáveis, Operadores, Entrada e Saída [ [slides](#) ] [ [vídeo](#) ]
  - Comandos Condicionais [ [slides](#) ] [ [vídeo](#) ]
  - Comandos de Repetição [ [slides](#) ] [ [vídeo](#) ]
  - Listas e Tuplas [ [slides](#) ] [ [vídeo](#) ]
  - Strings [ [slides](#) ] [ [vídeo](#) ]
  - Dicionários [ [slides](#) ] [ [vídeo](#) ]
  - Funções [ [slides](#) ] [ [vídeo](#) ]
  - Objetos Multidimensionais [ [slides](#) ] [ [vídeo](#) ]
  - Algoritmos de Ordenação [ [slides](#) ] [ [vídeo](#) ]
  - Algoritmos de Busca [ [slides](#) ] [ [vídeo](#) ]
  - Recursão [ [slides](#) ] [ [vídeo](#) ]
  - Algoritmos de Ordenação Recursivos [ [slides](#) ] [ [vídeo](#) ]
  - Arquivos [ [slides](#) ] [ [vídeo](#) ]
  - Expressões Regulares [ [slides](#) ] [ [vídeo](#) ]
  - Execução de Testes no Google Cloud Shell [ [slides](#) ] [ [vídeo](#) ]
  - Numpy [ [slides](#) ] [ [vídeo](#) ]
  - Pandas [ [slides](#) ] [ [vídeo](#) ]
- Panda - Cursos de Computação em Python (IME -USP) <https://panda.ime.usp.br/>
  - Como Pensar Como um Cientista da Computação <https://panda.ime.usp.br/pensepy/static/pensepy/>
  - Aulas de Introdução à Computação em Python <https://panda.ime.usp.br/aulasPython/static/aulasPython/>
- Fabio Kon, Introdução à Ciência da Computação com Python <http://bit.ly/FabioKon/>
- Socratica, Python Programming Tutorials <http://bit.ly/SocraticaPython/>
- Google - online editor for cloud-native applications (Python programming) <https://shell.cloud.google.com/>
- w3schools - Python Tutorial <https://www.w3schools.com/python/>
- Outros, citados nos Slides.