

# MC102 — Variáveis, Tipos, Operações Aritmética e Entrada/Saída

Rafael C. S. Schouery  
rafael@ic.unicamp.br

Universidade Estadual de Campinas

Atualizado em: 2023-03-14 11:15

## Calculando a média de três números

Queremos calcular a média  $M$  de três números  $x_1$ ,  $x_2$  e  $x_3$ , i.e.,

$$M = \frac{x_1 + x_2 + x_3}{3}$$

## Calculando a média de três números

Queremos calcular a média  $M$  de três números  $x_1$ ,  $x_2$  e  $x_3$ , i.e.,

$$M = \frac{x_1 + x_2 + x_3}{3}$$

E podemos usar o Python para tanto...

## Calculando a média de três números

Queremos calcular a média  $M$  de três números  $x_1$ ,  $x_2$  e  $x_3$ , i.e.,

$$M = \frac{x_1 + x_2 + x_3}{3}$$

E podemos usar o Python para tanto...

```
1 bash$ python
```

## Calculando a média de três números

Queremos calcular a média  $M$  de três números  $x_1$ ,  $x_2$  e  $x_3$ , i.e.,

$$M = \frac{x_1 + x_2 + x_3}{3}$$

E podemos usar o Python para tanto...

```
1 bash$ python
```

## Calculando a média de três números

Queremos calcular a média  $M$  de três números  $x_1$ ,  $x_2$  e  $x_3$ , i.e.,

$$M = \frac{x_1 + x_2 + x_3}{3}$$

E podemos usar o Python para tanto...

```
1 bash$ python
2 Python 3.7.5 (default, Dec  9 2019, 11: 40: 43)
3 [Clang 11.0.0 (clang-1100.0.33.12)] on darwin
4 Type "help", "copyright", "credits" or "license" for more
   information.
```

## Calculando a média de três números

Queremos calcular a média  $M$  de três números  $x_1$ ,  $x_2$  e  $x_3$ , i.e.,

$$M = \frac{x_1 + x_2 + x_3}{3}$$

E podemos usar o Python para tanto...

```
1 bash$ python
2 Python 3.7.5 (default, Dec  9 2019, 11: 40: 43)
3 [Clang 11.0.0 (clang-1100.0.33.12)] on darwin
4 Type "help", "copyright", "credits" or "license" for more
   information.
5 >>> (5.8 + 9.3 + 5.9)/3
```

# Calculando a média de três números

Queremos calcular a média  $M$  de três números  $x_1$ ,  $x_2$  e  $x_3$ , i.e.,

$$M = \frac{x_1 + x_2 + x_3}{3}$$

E podemos usar o Python para tanto...

```
1 bash$ python
2 Python 3.7.5 (default, Dec  9 2019, 11: 40: 43)
3 [Clang 11.0.0 (clang-1100.0.33.12)] on darwin
4 Type "help", "copyright", "credits" or "license" for more
   information.
5 >>> (5.8 + 9.3 + 5.9)/3
6 7.0
```



# Operações aritméticas básicas

Algumas operações que podemos fazer:

# Operações aritméticas básicas

Algumas operações que podemos fazer:

- soma: +

# Operações aritméticas básicas

Algumas operações que podemos fazer:

- soma: +
- subtração: -

# Operações aritméticas básicas

Algumas operações que podemos fazer:

- soma: +
- subtração: -
- multiplicação: \*

# Operações aritméticas básicas

Algumas operações que podemos fazer:

- soma: +
- subtração: -
- multiplicação: \*
- divisão: /

# Operações aritméticas básicas

Algumas operações que podemos fazer:

- soma:  $+$
- subtração:  $-$
- multiplicação:  $*$
- divisão:  $/$
- exponenciação:  $**$

# Operações aritméticas básicas

Algumas operações que podemos fazer:

- soma: **+**
- subtração: **-**
- multiplicação: **\***
- divisão: **/**
- exponenciação: **\*\***

```
1 >>> 3.2 + 7
```

# Operações aritméticas básicas

Algumas operações que podemos fazer:

- soma: **+**
- subtração: **-**
- multiplicação: **\***
- divisão: **/**
- exponenciação: **\*\***

```
1 >>> 3.2 + 7
```



# Operações aritméticas básicas

Algumas operações que podemos fazer:

- soma: **+**
- subtração: **-**
- multiplicação: **\***
- divisão: **/**
- exponenciação: **\*\***

```
1 >>> 3.2 + 7  
2 10.2
```

# Operações aritméticas básicas

Algumas operações que podemos fazer:

- soma: **+**
- subtração: **-**
- multiplicação: **\***
- divisão: **/**
- exponenciação: **\*\***

```
1 >>> 3.2 + 7
2 10.2
3 >>> 5.3 - -2
```

# Operações aritméticas básicas

Algumas operações que podemos fazer:

- soma: **+**
- subtração: **-**
- multiplicação: **\***
- divisão: **/**
- exponenciação: **\*\***

```
1 >>> 3.2 + 7
2 10.2
3 >>> 5.3 - -2
4 7.3
```

# Operações aritméticas básicas

Algumas operações que podemos fazer:

- soma: **+**
- subtração: **-**
- multiplicação: **\***
- divisão: **/**
- exponenciação: **\*\***

```
1 >>> 3.2 + 7
2 10.2
3 >>> 5.3 - -2
4 7.3
5 >>> 2 * 7.5
```

# Operações aritméticas básicas

Algumas operações que podemos fazer:

- soma: **+**
- subtração: **-**
- multiplicação: **\***
- divisão: **/**
- exponenciação: **\*\***

```
1 >>> 3.2 + 7
2 10.2
3 >>> 5.3 - -2
4 7.3
5 >>> 2 * 7.5
6 15.0
```

# Operações aritméticas básicas

Algumas operações que podemos fazer:

- soma: **+**
- subtração: **-**
- multiplicação: **\***
- divisão: **/**
- exponenciação: **\*\***

```
1 >>> 3.2 + 7
2 10.2
3 >>> 5.3 - -2
4 7.3
5 >>> 2 * 7.5
6 15.0
7 >>> 15 / 3
```

# Operações aritméticas básicas

Algumas operações que podemos fazer:

- soma: **+**
- subtração: **-**
- multiplicação: **\***
- divisão: **/**
- exponenciação: **\*\***

```
1 >>> 3.2 + 7
2 10.2
3 >>> 5.3 - -2
4 7.3
5 >>> 2 * 7.5
6 15.0
7 >>> 15 / 3
8 5.0
```

# Operações aritméticas básicas

Algumas operações que podemos fazer:

- soma: **+**
- subtração: **-**
- multiplicação: **\***
- divisão: **/**
- exponenciação: **\*\***

```
1 >>> 3.2 + 7
2 10.2
3 >>> 5.3 - -2
4 7.3
5 >>> 2 * 7.5
6 15.0
7 >>> 15 / 3
8 5.0
9 >>> 2.5 ** 3
```



# Operações aritméticas básicas

Algumas operações que podemos fazer:

- soma: **+**
- subtração: **-**
- multiplicação: **\***
- divisão: **/**
- exponenciação: **\*\***

```
1 >>> 3.2 + 7
2 10.2
3 >>> 5.3 - -2
4 7.3
5 >>> 2 * 7.5
6 15.0
7 >>> 15 / 3
8 5.0
9 >>> 2.5 ** 3
10 15.625
```

# Operações aritméticas básicas

Algumas operações que podemos fazer:

- soma: **+**
- subtração: **-**
- multiplicação: **\***
- divisão: **/**
- exponenciação: **\*\***

```
1 >>> 3.2 + 7
2 10.2
3 >>> 5.3 - -2
4 7.3
5 >>> 2 * 7.5
6 15.0
7 >>> 15 / 3
8 5.0
9 >>> 2.5 ** 3
10 15.625
11 >>> 3 / 0
```

# Operações aritméticas básicas

Algumas operações que podemos fazer:

- soma: **+**
- subtração: **-**
- multiplicação: **\***
- divisão: **/**
- exponenciação: **\*\***

```
1 >>> 3.2 + 7
2 10.2
3 >>> 5.3 - -2
4 7.3
5 >>> 2 * 7.5
6 15.0
7 >>> 15 / 3
8 5.0
9 >>> 2.5 ** 3
10 15.625
11 >>> 3 / 0
12 Traceback (most recent call last):
13   File "<stdin>", line 1, in <module>
14 ZeroDivisionError: division by zero
```

# Precedência de operadores

Temos a seguinte ordem:

# Precedência de operadores

Temos a seguinte ordem:

- exponenciação (**\*\***) tem a maior precedência

# Precedência de operadores

Temos a seguinte ordem:

- exponenciação (**\*\***) tem a maior precedência
- **\*** e **/** precede **+** e **-**

# Precedência de operadores

Temos a seguinte ordem:

- exponenciação (**\*\***) tem a maior precedência
- **\*** e **/** precede **+** e **-**
- em caso de empate, o operador da esquerda tem a maior precedência

# Precedência de operadores

Temos a seguinte ordem:

- exponenciação (**\*\***) tem a maior precedência
- **\*** e **/** precede **+** e **-**
- em caso de empate, o operador da esquerda tem a maior precedência
- parênteses podem ser usados para forçar a precedência



# Precedência de operadores

Temos a seguinte ordem:

- exponenciação (**\*\***) tem a maior precedência
- **\*** e **/** precede **+** e **-**
- em caso de empate, o operador da esquerda tem a maior precedência
- parênteses podem ser usados para forçar a precedência
  - mas não chaves e colchetes

# Precedência de operadores

Temos a seguinte ordem:

- exponenciação (**\*\***) tem a maior precedência
- **\*** e **/** precede **+** e **-**
- em caso de empate, o operador da esquerda tem a maior precedência
- parênteses podem ser usados para forçar a precedência
  - mas não chaves e colchetes

1 >>> 2 + 3 \* 7

# Precedência de operadores

Temos a seguinte ordem:

- exponenciação (**\*\***) tem a maior precedência
- **\*** e **/** precede **+** e **-**
- em caso de empate, o operador da esquerda tem a maior precedência
- parênteses podem ser usados para forçar a precedência
  - mas não chaves e colchetes

1 >>> 2 + 3 \* 7

# Precedência de operadores

Temos a seguinte ordem:

- exponenciação (**\*\***) tem a maior precedência
- **\*** e **/** precede **+** e **-**
- em caso de empate, o operador da esquerda tem a maior precedência
- parênteses podem ser usados para forçar a precedência
  - mas não chaves e colchetes

```
1 >>> 2 + 3 * 7
```

```
2 23
```

# Precedência de operadores

Temos a seguinte ordem:

- exponenciação (**\*\***) tem a maior precedência
- **\*** e **/** precede **+** e **-**
- em caso de empate, o operador da esquerda tem a maior precedência
- parênteses podem ser usados para forçar a precedência
  - mas não chaves e colchetes

```
1 >>> 2 + 3 * 7
2 23
3 >>> (2 + 3) * 7
```

# Precedência de operadores

Temos a seguinte ordem:

- exponenciação (**\*\***) tem a maior precedência
- **\*** e **/** precede **+** e **-**
- em caso de empate, o operador da esquerda tem a maior precedência
- parênteses podem ser usados para forçar a precedência
  - mas não chaves e colchetes

```
1 >>> 2 + 3 * 7
2 23
3 >>> (2 + 3) * 7
4 35
```

# Precedência de operadores

Temos a seguinte ordem:

- exponenciação (**\*\***) tem a maior precedência
- **\*** e **/** precede **+** e **-**
- em caso de empate, o operador da esquerda tem a maior precedência
- parênteses podem ser usados para forçar a precedência
  - mas não chaves e colchetes

```
1 >>> 2 + 3 * 7
2 23
3 >>> (2 + 3) * 7
4 35
5 >>> ((2 + 3) * 7 + 15) * 2
```

# Precedência de operadores

Temos a seguinte ordem:

- exponenciação (**\*\***) tem a maior precedência
- **\*** e **/** precede **+** e **-**
- em caso de empate, o operador da esquerda tem a maior precedência
- parênteses podem ser usados para forçar a precedência
  - mas não chaves e colchetes

```
1 >>> 2 + 3 * 7
2 23
3 >>> (2 + 3) * 7
4 35
5 >>> ((2 + 3) * 7 + 15) * 2
6 100
```



# Precedência de operadores

Temos a seguinte ordem:

- exponenciação (**\*\***) tem a maior precedência
- **\*** e **/** precede **+** e **-**
- em caso de empate, o operador da esquerda tem a maior precedência
- parênteses podem ser usados para forçar a precedência
  - mas não chaves e colchetes

```
1 >>> 2 + 3 * 7
2 23
3 >>> (2 + 3) * 7
4 35
5 >>> ((2 + 3) * 7 + 15) * 2
6 100
7 >>> [(2 + 3) * 7 + 15] * 2
```

# Precedência de operadores

Temos a seguinte ordem:

- exponenciação (**\*\***) tem a maior precedência
- **\*** e **/** precede **+** e **-**
- em caso de empate, o operador da esquerda tem a maior precedência
- parênteses podem ser usados para forçar a precedência
  - mas não chaves e colchetes

```
1 >>> 2 + 3 * 7
2 23
3 >>> (2 + 3) * 7
4 35
5 >>> ((2 + 3) * 7 + 15) * 2
6 100
7 >>> [(2 + 3) * 7 + 15] * 2
8 [50, 50]
```

# Movimento Uniformemente Variado

Calcular a posição final  $s$  de um objeto

# Movimento Uniformemente Variado

Calcular a posição final  $s$  de um objeto

- inicialmente na posição  $s_0$

# Movimento Uniformemente Variado

Calcular a posição final  $s$  de um objeto

- inicialmente na posição  $s_0$
- que se move com aceleração constante  $a$

# Movimento Uniformemente Variado

Calcular a posição final  $s$  de um objeto

- inicialmente na posição  $s_0$
- que se move com aceleração constante  $a$
- durante  $t$  segundos

# Movimento Uniformemente Variado

Calcular a posição final  $s$  de um objeto

- inicialmente na posição  $s_0$
- que se move com aceleração constante  $a$
- durante  $t$  segundos
- e com velocidade inicial  $v_0$

# Movimento Uniformemente Variado

Calcular a posição final  $s$  de um objeto

- inicialmente na posição  $s_0$
- que se move com aceleração constante  $a$
- durante  $t$  segundos
- e com velocidade inicial  $v_0$

$$s = s_0 + v_0 t + \frac{at^2}{2}$$



# Movimento Uniformemente Variado

Calcular a posição final  $s$  de um objeto

- inicialmente na posição  $s_0$
- que se move com aceleração constante  $a$
- durante  $t$  segundos
- e com velocidade inicial  $v_0$

$$s = s_0 + v_0t + \frac{at^2}{2}$$

Em Python, algo do tipo:

# Movimento Uniformemente Variado

Calcular a posição final  $s$  de um objeto

- inicialmente na posição  $s_0$
- que se move com aceleração constante  $a$
- durante  $t$  segundos
- e com velocidade inicial  $v_0$

$$s = s_0 + v_0t + \frac{at^2}{2}$$

Em Python, algo do tipo:

```
10 + 2 * 8 + 3 * 8**2 / 2
```

# Movimento Uniformemente Variado

Calcular a posição final  $s$  de um objeto

- inicialmente na posição  $s_0$
- que se move com aceleração constante  $a$
- durante  $t$  segundos
- e com velocidade inicial  $v_0$

$$s = s_0 + v_0 t + \frac{at^2}{2}$$

Em Python, algo do tipo:

```
10 + 2 * 8 + 3 * 8**2 / 2
```

E se pudéssemos guardar os valores de  $s_0$ ,  $a$ ,  $t$  e  $v_0$ ?

# Armazenando informação

Algo mais longo, mas mais legível:

```
1 >>> s0 = 10
```

# Armazenando informação

Algo mais longo, mas mais legível:

```
1 >>> s0 = 10
```

# Armazenando informação

Algo mais longo, mas mais legível:

```
1 >>> s0 = 10
2 >>> v0 = 2
3 >>> t = 8
4 >>> a = 3
```

# Armazenando informação

Algo mais longo, mas mais legível:

```
1 >>> s0 = 10
2 >>> v0 = 2
3 >>> t = 8
4 >>> a = 3
5 >>> s = s0 + v0 * t + a * t ** 2 / 2
```

# Armazenando informação

Algo mais longo, mas mais legível:

```
1 >>> s0 = 10
2 >>> v0 = 2
3 >>> t = 8
4 >>> a = 3
5 >>> s = s0 + v0 * t + a * t ** 2 / 2
6 >>> s
```



# Armazenando informação

Algo mais longo, mas mais legível:

```
1 >>> s0 = 10
2 >>> v0 = 2
3 >>> t = 8
4 >>> a = 3
5 >>> s = s0 + v0 * t + a * t ** 2 / 2
6 >>> s
7 122.0
```

# Variáveis

Variáveis são nomes para regiões da memória do computador

# Variáveis

Variáveis são nomes para regiões da memória do computador

- Servem para armazenar informações

# Variáveis

Variáveis são nomes para regiões da memória do computador

- Servem para armazenar informações
- Com um nome bem definido

# Variáveis

Variáveis são nomes para regiões da memória do computador

- Servem para armazenar informações
- Com um nome bem definido
- Mais fácil de lembrar do que um endereço da memória

# Variáveis

Variáveis são nomes para regiões da memória do computador

- Servem para armazenar informações
- Com um nome bem definido
- Mais fácil de lembrar do que um endereço da memória
- Bons nomes de variáveis deixam os códigos mais legíveis

# Variáveis

Variáveis são nomes para regiões da memória do computador

- Servem para armazenar informações
- Com um nome bem definido
- Mais fácil de lembrar do que um endereço da memória
- Bons nomes de variáveis deixam os códigos mais legíveis

O nome “**variável**” vem do fato que o valor pode mudar!

# Variáveis

Variáveis são nomes para regiões da memória do computador

- Servem para armazenar informações
- Com um nome bem definido
- Mais fácil de lembrar do que um endereço da memória
- Bons nomes de variáveis deixam os códigos mais legíveis

O nome “**variável**” vem do fato que o valor pode mudar!

```
1 >>> x = 10
```



# Variáveis

Variáveis são nomes para regiões da memória do computador

- Servem para armazenar informações
- Com um nome bem definido
- Mais fácil de lembrar do que um endereço da memória
- Bons nomes de variáveis deixam os códigos mais legíveis

O nome “**variável**” vem do fato que o valor pode mudar!

```
1 >>> x = 10
```

# Variáveis

Variáveis são nomes para regiões da memória do computador

- Servem para armazenar informações
- Com um nome bem definido
- Mais fácil de lembrar do que um endereço da memória
- Bons nomes de variáveis deixam os códigos mais legíveis

O nome “variável” vem do fato que o valor pode mudar!

```
1 >>> x = 10
2 >>> x
```

# Variáveis

Variáveis são nomes para regiões da memória do computador

- Servem para armazenar informações
- Com um nome bem definido
- Mais fácil de lembrar do que um endereço da memória
- Bons nomes de variáveis deixam os códigos mais legíveis

O nome “variável” vem do fato que o valor pode mudar!

```
1 >>> x = 10
2 >>> x
3 10
```

# Variáveis

Variáveis são nomes para regiões da memória do computador

- Servem para armazenar informações
- Com um nome bem definido
- Mais fácil de lembrar do que um endereço da memória
- Bons nomes de variáveis deixam os códigos mais legíveis

O nome “variável” vem do fato que o valor pode mudar!

```
1 >>> x = 10
2 >>> x
3 10
4 >>> x = 15
```

# Variáveis

Variáveis são nomes para regiões da memória do computador

- Servem para armazenar informações
- Com um nome bem definido
- Mais fácil de lembrar do que um endereço da memória
- Bons nomes de variáveis deixam os códigos mais legíveis

O nome “**variável**” vem do fato que o valor pode mudar!

```
1 >>> x = 10
2 >>> x
3 10
4 >>> x = 15
5 >>> x
```

# Variáveis

Variáveis são nomes para regiões da memória do computador

- Servem para armazenar informações
- Com um nome bem definido
- Mais fácil de lembrar do que um endereço da memória
- Bons nomes de variáveis deixam os códigos mais legíveis

O nome “**variável**” vem do fato que o valor pode mudar!

```
1 >>> x = 10
2 >>> x
3 10
4 >>> x = 15
5 >>> x
6 15
```

## Regras para nomes de variáveis

- Precisa começar com uma letra ou `_` (underscore)

## Regras para nomes de variáveis

- Precisa começar com uma letra ou `_` (underscore)
  - Válidos: `x`, `y`, `_x`, `variavel`, `minha_variavel`



## Regras para nomes de variáveis

- Precisa começar com uma letra ou `_` (underscore)
  - Válidos: `x`, `y`, `_x`, `variavel`, `minha_variavel`
  - Inválidos: `9`, `9x`, `?x`

## Regras para nomes de variáveis

- Precisa começar com uma letra ou `_` (underscore)
  - Válidos: `x`, `y`, `_x`, `variavel`, `minha_variavel`
  - Inválidos: `9`, `9x`, `?x`
- Outros caracteres podem ser letras, números ou `_`

## Regras para nomes de variáveis

- Precisa começar com uma letra ou `_` (underscore)
  - Válidos: `x`, `y`, `_x`, `variavel`, `minha_variavel`
  - Inválidos: `9`, `9x`, `?x`
- Outros caracteres podem ser letras, números ou `_`
  - Válidos: `x1`, `x_1`

## Regras para nomes de variáveis

- Precisa começar com uma letra ou `_` (underscore)
  - Válidos: `x`, `y`, `_x`, `variavel`, `minha_variavel`
  - Inválidos: `9`, `9x`, `?x`
- Outros caracteres podem ser letras, números ou `_`
  - Válidos: `x1`, `x_1`
  - Inválidos: `x?`, `x 1`

## Regras para nomes de variáveis

- Precisa começar com uma letra ou `_` (underscore)
  - Válidos: `x`, `y`, `_x`, `variavel`, `minha_variavel`
  - Inválidos: `9`, `9x`, `?x`
- Outros caracteres podem ser letras, números ou `_`
  - Válidos: `x1`, `x_1`
  - Inválidos: `x?`, `x 1`
- É sensível a maiúsculas e minúsculas

## Regras para nomes de variáveis

- Precisa começar com uma letra ou `_` (underscore)
  - Válidos: `x`, `y`, `_x`, `variavel`, `minha_variavel`
  - Inválidos: `9`, `9x`, `?x`
- Outros caracteres podem ser letras, números ou `_`
  - Válidos: `x1`, `x_1`
  - Inválidos: `x?`, `x 1`
- É sensível a maiúsculas e minúsculas
  - Variável `x` é diferente de `X`

## Regras para nomes de variáveis

- Precisa começar com uma letra ou `_` (underscore)
  - Válidos: `x`, `y`, `_x`, `variavel`, `minha_variavel`
  - Inválidos: `9`, `9x`, `?x`
- Outros caracteres podem ser letras, números ou `_`
  - Válidos: `x1`, `x_1`
  - Inválidos: `x?`, `x 1`
- É sensível a maiúsculas e minúsculas
  - Variável `x` é diferente de `X`
- Alguns nomes não podem ser usados

## Regras para nomes de variáveis

- Precisa começar com uma letra ou `_` (underscore)
  - Válidos: `x`, `y`, `_x`, `variavel`, `minha_variavel`
  - Inválidos: `9`, `9x`, `?x`
- Outros caracteres podem ser letras, números ou `_`
  - Válidos: `x1`, `x_1`
  - Inválidos: `x?`, `x 1`
- É sensível a maiúsculas e minúsculas
  - Variável `x` é diferente de `X`
- Alguns nomes não podem ser usados
  - Significam outra coisa em Python (`if`, `import`, `while`, etc)



## Regras para nomes de variáveis

- Precisa começar com uma letra ou `_` (underscore)
  - Válidos: `x`, `y`, `_x`, `variavel`, `minha_variavel`
  - Inválidos: `9`, `9x`, `?x`
- Outros caracteres podem ser letras, números ou `_`
  - Válidos: `x1`, `x_1`
  - Inválidos: `x?`, `x 1`
- É sensível a maiúsculas e minúsculas
  - Variável `x` é diferente de `X`
- Alguns nomes não podem ser usados
  - Significam outra coisa em Python (`if`, `import`, `while`, etc)

Bons nomes de variável:

## Regras para nomes de variáveis

- Precisa começar com uma letra ou `_` (underscore)
  - Válidos: `x`, `y`, `_x`, `variavel`, `minha_variavel`
  - Inválidos: `9`, `9x`, `?x`
- Outros caracteres podem ser letras, números ou `_`
  - Válidos: `x1`, `x_1`
  - Inválidos: `x?`, `x 1`
- É sensível a maiúsculas e minúsculas
  - Variável `x` é diferente de `X`
- Alguns nomes não podem ser usados
  - Significam outra coisa em Python (`if`, `import`, `while`, etc)

Bons nomes de variável:

- Use nomes significativos

## Regras para nomes de variáveis

- Precisa começar com uma letra ou `_` (underscore)
  - Válidos: `x`, `y`, `_x`, `variavel`, `minha_variavel`
  - Inválidos: `9`, `9x`, `?x`
- Outros caracteres podem ser letras, números ou `_`
  - Válidos: `x1`, `x_1`
  - Inválidos: `x?`, `x 1`
- É sensível a maiúsculas e minúsculas
  - Variável `x` é diferente de `X`
- Alguns nomes não podem ser usados
  - Significam outra coisa em Python (`if`, `import`, `while`, etc)

Bons nomes de variável:

- Use nomes significativos
  - `nome` é melhor do que `n`

## Regras para nomes de variáveis

- Precisa começar com uma letra ou `_` (underscore)
  - Válidos: `x`, `y`, `_x`, `variavel`, `minha_variavel`
  - Inválidos: `9`, `9x`, `?x`
- Outros caracteres podem ser letras, números ou `_`
  - Válidos: `x1`, `x_1`
  - Inválidos: `x?`, `x 1`
- É sensível a maiúsculas e minúsculas
  - Variável `x` é diferente de `X`
- Alguns nomes não podem ser usados
  - Significam outra coisa em Python (`if`, `import`, `while`, etc)

Bons nomes de variável:

- Use nomes significativos
  - `nome` é melhor do que `n`
- Mas evite nomes muito grandes

## Regras para nomes de variáveis

- Precisa começar com uma letra ou `_` (underscore)
  - Válidos: `x`, `y`, `_x`, `variavel`, `minha_variavel`
  - Inválidos: `9`, `9x`, `?x`
- Outros caracteres podem ser letras, números ou `_`
  - Válidos: `x1`, `x_1`
  - Inválidos: `x?`, `x 1`
- É sensível a maiúsculas e minúsculas
  - Variável `x` é diferente de `X`
- Alguns nomes não podem ser usados
  - Significam outra coisa em Python (`if`, `import`, `while`, etc)

Bons nomes de variável:

- Use nomes significativos
  - `nome` é melhor do que `n`
- Mas evite nomes muito grandes
  - `nome` é melhor do que `nome_do_aluno_de_MC102`

## Regras para nomes de variáveis

- Precisa começar com uma letra ou `_` (underscore)
  - Válidos: `x`, `y`, `_x`, `variavel`, `minha_variavel`
  - Inválidos: `9`, `9x`, `?x`
- Outros caracteres podem ser letras, números ou `_`
  - Válidos: `x1`, `x_1`
  - Inválidos: `x?`, `x 1`
- É sensível a maiúsculas e minúsculas
  - Variável `x` é diferente de `X`
- Alguns nomes não podem ser usados
  - Significam outra coisa em Python (`if`, `import`, `while`, etc)

### Bons nomes de variável:

- Use nomes significativos
  - `nome` é melhor do que `n`
- Mas evite nomes muito grandes
  - `nome` é melhor do que `nome_do_aluno_de_MC102`
- Seja consistente

## Regras para nomes de variáveis

- Precisa começar com uma letra ou `_` (underscore)
  - Válidos: `x`, `y`, `_x`, `variavel`, `minha_variavel`
  - Inválidos: `9`, `9x`, `?x`
- Outros caracteres podem ser letras, números ou `_`
  - Válidos: `x1`, `x_1`
  - Inválidos: `x?`, `x 1`
- É sensível a maiúsculas e minúsculas
  - Variável `x` é diferente de `X`
- Alguns nomes não podem ser usados
  - Significam outra coisa em Python (`if`, `import`, `while`, etc)

### Bons nomes de variável:

- Use nomes significativos
  - `nome` é melhor do que `n`
- Mas evite nomes muito grandes
  - `nome` é melhor do que `nome_do_aluno_de_MC102`
- Seja consistente
  - `nome_do_aluno` ou `NomeDoAluno?`

## Regras para nomes de variáveis

- Precisa começar com uma letra ou `_` (underscore)
  - Válidos: `x`, `y`, `_x`, `variavel`, `minha_variavel`
  - Inválidos: `9`, `9x`, `?x`
- Outros caracteres podem ser letras, números ou `_`
  - Válidos: `x1`, `x_1`
  - Inválidos: `x?`, `x 1`
- É sensível a maiúsculas e minúsculas
  - Variável `x` é diferente de `X`
- Alguns nomes não podem ser usados
  - Significam outra coisa em Python (`if`, `import`, `while`, etc)

### Bons nomes de variável:

- Use nomes significativos
  - `nome` é melhor do que `n`
- Mas evite nomes muito grandes
  - `nome` é melhor do que `nome_do_aluno_de_MC102`
- Seja consistente
  - `nome_do_aluno` ou `NomeDoAluno`?
  - A convenção do Python pede para usar a primeira forma



## Variáveis, constantes e atribuição

Chamamos de constantes os valores que não mudam:

## Variáveis, constantes e atribuição

Chamamos de constantes os valores que não mudam:

- Ex: 2.34, -7.107, 10

## Variáveis, constantes e atribuição

Chamamos de constantes os valores que não mudam:

- Ex: 2.34, -7.107, 10

Atribuição: a operação de armazenar um valor em uma variável

## Variáveis, constantes e atribuição

Chamamos de constantes os valores que não mudam:

- Ex: 2.34, -7.107, 10

Atribuição: a operação de armazenar um valor em uma variável

- O operador é o =

## Variáveis, constantes e atribuição

Chamamos de constantes os valores que não mudam:

- Ex: 2.34, -7.107, 10

Atribuição: a operação de armazenar um valor em uma variável

- O operador é o =
- Podemos armazenar uma constante

## Variáveis, constantes e atribuição

Chamamos de constantes os valores que não mudam:

- Ex: 2.34, -7.107, 10

Atribuição: a operação de armazenar um valor em uma variável

- O operador é o =
- Podemos armazenar uma constante
  - Ex:  $x = 10$

## Variáveis, constantes e atribuição

Chamamos de constantes os valores que não mudam:

- Ex: 2.34, -7.107, 10

Atribuição: a operação de armazenar um valor em uma variável

- O operador é o =
- Podemos armazenar uma constante
  - Ex:  $x = 10$
- Ou o valor armazenado em outra variável

## Variáveis, constantes e atribuição

Chamamos de constantes os valores que não mudam:

- Ex: 2.34, -7.107, 10

Atribuição: a operação de armazenar um valor em uma variável

- O operador é o =
- Podemos armazenar uma constante
  - Ex:  $x = 10$
- Ou o valor armazenado em outra variável
  - Ex:  $x = y$



## Variáveis, constantes e atribuição

Chamamos de constantes os valores que não mudam:

- Ex: 2.34, -7.107, 10

Atribuição: a operação de armazenar um valor em uma variável

- O operador é o =
- Podemos armazenar uma constante
  - Ex:  $x = 10$
- Ou o valor armazenado em outra variável
  - Ex:  $x = y$

```
1 >>> y = 10
2 >>> y
3 10
```

## Variáveis, constantes e atribuição

Chamamos de constantes os valores que não mudam:

- Ex: 2.34, -7.107, 10

Atribuição: a operação de armazenar um valor em uma variável

- O operador é o =
- Podemos armazenar uma constante
  - Ex:  $x = 10$
- Ou o valor armazenado em outra variável
  - Ex:  $x = y$

```
1 >>> y = 10
2 >>> y
3 10
```

## Variáveis, constantes e atribuição

Chamamos de constantes os valores que não mudam:

- Ex: 2.34, -7.107, 10

Atribuição: a operação de armazenar um valor em uma variável

- O operador é o =
- Podemos armazenar uma constante
  - Ex:  $x = 10$
- Ou o valor armazenado em outra variável
  - Ex:  $x = y$

```
1 >>> y = 10
2 >>> y
3 10
4 >>> x = y
5 >>> x
6 10
```

## Variáveis, constantes e atribuição

Chamamos de constantes os valores que não mudam:

- Ex: 2.34, -7.107, 10

Atribuição: a operação de armazenar um valor em uma variável

- O operador é o =
- Podemos armazenar uma constante
  - Ex:  $x = 10$
- Ou o valor armazenado em outra variável
  - Ex:  $x = y$

```
1 >>> y = 10
2 >>> y
3 10
4 >>> x = y
5 >>> x
6 10
7 >>> y = 15
8 >>> y
9 15
```

## Variáveis, constantes e atribuição

Chamamos de constantes os valores que não mudam:

- Ex: 2.34, -7.107, 10

Atribuição: a operação de armazenar um valor em uma variável

- O operador é o =
- Podemos armazenar uma constante
  - Ex:  $x = 10$
- Ou o valor armazenado em outra variável
  - Ex:  $x = y$

```
1 >>> y = 10
2 >>> y
3 10
4 >>> x = y
5 >>> x
6 10
7 >>> y = 15
8 >>> y
9 15
10 >>> x
11 10
```

## Variáveis, constantes e atribuição

Chamamos de constantes os valores que não mudam:

- Ex: 2.34, -7.107, 10

Atribuição: a operação de armazenar um valor em uma variável

- O operador é o =
- Podemos armazenar uma constante
  - Ex:  $x = 10$
- Ou o valor armazenado em outra variável
  - Ex:  $x = y$

```
1 >>> y = 10
2 >>> y
3 10
4 >>> x = y
5 >>> x
6 10
7 >>> y = 15
8 >>> y
9 15
10 >>> x
11 10
```

## Detalhes

O que acontece nesse código?

```
1 >>> x
```

## Detalhes

O que acontece nesse código?

```
1 >>> x
```



# Detalhes

O que acontece nesse código?

```
1 >>> x
2 Traceback (most recent call last):
3   File "<stdin>", line 1, in <module>
4 NameError: name 'x' is not defined
```

# Detalhes

O que acontece nesse código?

```
1 >>> x
2 Traceback (most recent call last):
3   File "<stdin>", line 1, in <module>
4 NameError: name 'x' is not defined
5 >>> x = 10
```

# Detalhes

O que acontece nesse código?

```
1 >>> x
2 Traceback (most recent call last):
3   File "<stdin>", line 1, in <module>
4 NameError: name 'x' is not defined
5 >>> x = 10
6 >>> x
```

# Detalhes

O que acontece nesse código?

```
1 >>> x
2 Traceback (most recent call last):
3   File "<stdin>", line 1, in <module>
4 NameError: name 'x' is not defined
5 >>> x = 10
6 >>> x
7 10
```

# Detalhes

O que acontece nesse código?

```
1 >>> x
2 Traceback (most recent call last):
3   File "<stdin>", line 1, in <module>
4 NameError: name 'x' is not defined
5 >>> x = 10
6 >>> x
7 10
```

A variável passa a existir quando recebe a primeira atribuição!

# Detalhes

O que acontece nesse código?

```
1 >>> x
2 Traceback (most recent call last):
3   File "<stdin>", line 1, in <module>
4 NameError: name 'x' is not defined
5 >>> x = 10
6 >>> x
7 10
```

A variável passa a existir quando recebe a primeira atribuição!

E nesse?

# Detalhes

O que acontece nesse código?

```
1 >>> x
2 Traceback (most recent call last):
3   File "<stdin>", line 1, in <module>
4 NameError: name 'x' is not defined
5 >>> x = 10
6 >>> x
7 10
```

A variável passa a existir quando recebe a primeira atribuição!

E nesse?

```
1 >>> 10 = x
```

# Detalhes

O que acontece nesse código?

```
1 >>> x
2 Traceback (most recent call last):
3   File "<stdin>", line 1, in <module>
4 NameError: name 'x' is not defined
5 >>> x = 10
6 >>> x
7 10
```

A variável passa a existir quando recebe a primeira atribuição!

E nesse?

```
1 >>> 10 = x
```



# Detalhes

O que acontece nesse código?

```
1 >>> x
2 Traceback (most recent call last):
3   File "<stdin>", line 1, in <module>
4 NameError: name 'x' is not defined
5 >>> x = 10
6 >>> x
7 10
```

A variável passa a existir quando recebe a primeira atribuição!

E nesse?

```
1 >>> 10 = x
2   File "<stdin>", line 1
3 SyntaxError: can't assign to literal
```

O `=` é uma atribuição de valor, não uma igualdade matemática!

## Tipo de dados

Até o momento trabalhamos apenas com números...

## Tipo de dados

Até o momento trabalhamos apenas com números...

- Mas variáveis podem guardar muitos outros dados!

## Tipo de dados

Até o momento trabalhamos apenas com números...

- Mas variáveis podem guardar muitos outros dados!
- Ex: `texto = 'MC102'`

# Tipo de dados

Até o momento trabalhamos apenas com números...

- Mas variáveis podem guardar muitos outros dados!
- Ex: `texto = 'MC102'`

O **tipo** de uma informação defini

# Tipo de dados

Até o momento trabalhamos apenas com números...

- Mas variáveis podem guardar muitos outros dados!
- Ex: `texto = 'MC102'`

O **tipo** de uma informação defini

- as operações que podemos fazer com ela

# Tipo de dados

Até o momento trabalhamos apenas com números...

- Mas variáveis podem guardar muitos outros dados!
- Ex: `texto = 'MC102'`

O **tipo** de uma informação defini

- as operações que podemos fazer com ela
- e qual é o resultado

# Tipo de dados

Até o momento trabalhamos apenas com números...

- Mas variáveis podem guardar muitos outros dados!
- Ex: `texto = 'MC102'`

O **tipo** de uma informação defini

- as operações que podemos fazer com ela
- e qual é o resultado

```
1 >>> 10.3 + 5.2
```



# Tipo de dados

Até o momento trabalhamos apenas com números...

- Mas variáveis podem guardar muitos outros dados!
- Ex: `texto = 'MC102'`

O **tipo** de uma informação defini

- as operações que podemos fazer com ela
- e qual é o resultado

```
1 >>> 10.3 + 5.2
```

# Tipo de dados

Até o momento trabalhamos apenas com números...

- Mas variáveis podem guardar muitos outros dados!
- Ex: `texto = 'MC102'`

O **tipo** de uma informação defini

- as operações que podemos fazer com ela
- e qual é o resultado

```
1 >>> 10.3 + 5.2
2 15.5
```

# Tipo de dados

Até o momento trabalhamos apenas com números...

- Mas variáveis podem guardar muitos outros dados!
- Ex: `texto = 'MC102'`

O **tipo** de uma informação defini

- as operações que podemos fazer com ela
- e qual é o resultado

```
1 >>> 10.3 + 5.2
2 15.5
3 >>> 'mc' + '102'
```

# Tipo de dados

Até o momento trabalhamos apenas com números...

- Mas variáveis podem guardar muitos outros dados!
- Ex: `texto = 'MC102'`

O **tipo** de uma informação defini

- as operações que podemos fazer com ela
- e qual é o resultado

```
1 >>> 10.3 + 5.2
2 15.5
3 >>> 'mc' + '102'
4 'mc102'
```

# Tipo de dados

Até o momento trabalhamos apenas com números...

- Mas variáveis podem guardar muitos outros dados!
- Ex: `texto = 'MC102'`

O **tipo** de uma informação defini

- as operações que podemos fazer com ela
- e qual é o resultado

```
1 >>> 10.3 + 5.2
2 15.5
3 >>> 'mc' + '102'
4 'mc102'
5 >>> 'mc' + 102
```

# Tipo de dados

Até o momento trabalhamos apenas com números...

- Mas variáveis podem guardar muitos outros dados!
- Ex: `texto = 'MC102'`

O `tipo` de uma informação defini

- as operações que podemos fazer com ela
- e qual é o resultado

```
1 >>> 10.3 + 5.2
2 15.5
3 >>> 'mc' + '102'
4 'mc102'
5 >>> 'mc' + 102
6 Traceback (most recent call last):
7   File "<stdin>", line 1, in <module>
8 TypeError: can only concatenate str (not "int") to str
```

# Tipo de dados

Até o momento trabalhamos apenas com números...

- Mas variáveis podem guardar muitos outros dados!
- Ex: `texto = 'MC102'`

O **tipo** de uma informação defini

- as operações que podemos fazer com ela
- e qual é o resultado

```
1 >>> 10.3 + 5.2
2 15.5
3 >>> 'mc' + '102'
4 'mc102'
5 >>> 'mc' + 102
6 Traceback (most recent call last):
7   File "<stdin>", line 1, in <module>
8 TypeError: can only concatenate str (not "int") to str
9 >>> 'mc' * 2
```

# Tipo de dados

Até o momento trabalhamos apenas com números...

- Mas variáveis podem guardar muitos outros dados!
- Ex: `texto = 'MC102'`

O **tipo** de uma informação defini

- as operações que podemos fazer com ela
- e qual é o resultado

```
1 >>> 10.3 + 5.2
2 15.5
3 >>> 'mc' + '102'
4 'mc102'
5 >>> 'mc' + 102
6 Traceback (most recent call last):
7   File "<stdin>", line 1, in <module>
8 TypeError: can only concatenate str (not "int") to str
9 >>> 'mc' * 2
10 'mcmc'
```



## Números “reais” (ponto flutuante) - float

É o tipo usado para representar números reais

## Números “reais” (ponto flutuante) - float

É o tipo usado para representar números reais

- Mas temos uma quantidade limitada de dígitos

## Números “reais” (ponto flutuante) - float

É o tipo usado para representar números reais

- Mas temos uma quantidade limitada de dígitos
- Não podemos representar qualquer número real...

## Números “reais” (ponto flutuante) - float

É o tipo usado para representar números reais

- Mas temos uma quantidade limitada de dígitos
- Não podemos representar qualquer número real...
  - $\pi$  tem infinitos dígitos após a vírgula

## Números “reais” (ponto flutuante) - float

É o tipo usado para representar números reais

- Mas temos uma quantidade limitada de dígitos
- Não podemos representar qualquer número real...
  - $\pi$  tem infinitos dígitos após a vírgula
- E as operações podem ter erros de precisão

## Números “reais” (ponto flutuante) - float

É o tipo usado para representar números reais

- Mas temos uma quantidade limitada de dígitos
- Não podemos representar qualquer número real...
  - $\pi$  tem infinitos dígitos após a vírgula
- E as operações podem ter erros de precisão

```
1 >>> 15.3 / 3
```

## Números “reais” (ponto flutuante) - float

É o tipo usado para representar números reais

- Mas temos uma quantidade limitada de dígitos
- Não podemos representar qualquer número real...
  - $\pi$  tem infinitos dígitos após a vírgula
- E as operações podem ter erros de precisão

```
1 >>> 15.3 / 3
```

## Números “reais” (ponto flutuante) - float

É o tipo usado para representar números reais

- Mas temos uma quantidade limitada de dígitos
- Não podemos representar qualquer número real...
  - $\pi$  tem infinitos dígitos após a vírgula
- E as operações podem ter erros de precisão

```
1 >>> 15.3 / 3
2 5.1000000000000005
```



## Números “reais” (ponto flutuante) - float

É o tipo usado para representar números reais

- Mas temos uma quantidade limitada de dígitos
- Não podemos representar qualquer número real...
  - $\pi$  tem infinitos dígitos após a vírgula
- E as operações podem ter erros de precisão

```
1 >>> 15.3 / 3
2 5.10000000000000005
3 >>> 5.3 - 2.1
```

## Números “reais” (ponto flutuante) - float

É o tipo usado para representar números reais

- Mas temos uma quantidade limitada de dígitos
- Não podemos representar qualquer número real...
  - $\pi$  tem infinitos dígitos após a vírgula
- E as operações podem ter erros de precisão

```
1 >>> 15.3 / 3
2 5.10000000000000005
3 >>> 5.3 - 2.1
4 3.1999999999999997
```

## Números “reais” (ponto flutuante) - float

É o tipo usado para representar números reais

- Mas temos uma quantidade limitada de dígitos
- Não podemos representar qualquer número real...
  - $\pi$  tem infinitos dígitos após a vírgula
- E as operações podem ter erros de precisão

```
1 >>> 15.3 / 3
2 5.10000000000000005
3 >>> 5.3 - 2.1
4 3.1999999999999997
5 >>> 5.3 + 1.9
```

## Números “reais” (ponto flutuante) - float

É o tipo usado para representar números reais

- Mas temos uma quantidade limitada de dígitos
- Não podemos representar qualquer número real...
  - $\pi$  tem infinitos dígitos após a vírgula
- E as operações podem ter erros de precisão

```
1 >>> 15.3 / 3
2 5.1000000000000005
3 >>> 5.3 - 2.1
4 3.1999999999999997
5 >>> 5.3 + 1.9
6 7.199999999999999
```

## Números “reais” (ponto flutuante) - float

É o tipo usado para representar números reais

- Mas temos uma quantidade limitada de dígitos
- Não podemos representar qualquer número real...
  - $\pi$  tem infinitos dígitos após a vírgula
- E as operações podem ter erros de precisão

```
1 >>> 15.3 / 3
2 5.10000000000000005
3 >>> 5.3 - 2.1
4 3.1999999999999997
5 >>> 5.3 + 1.9
6 7.1999999999999999
7 >>> 0.1 + 0.2
```

## Números “reais” (ponto flutuante) - float

É o tipo usado para representar números reais

- Mas temos uma quantidade limitada de dígitos
- Não podemos representar qualquer número real...
  - $\pi$  tem infinitos dígitos após a vírgula
- E as operações podem ter erros de precisão

```
1 >>> 15.3 / 3
2 5.1000000000000005
3 >>> 5.3 - 2.1
4 3.1999999999999997
5 >>> 5.3 + 1.9
6 7.199999999999999
7 >>> 0.1 + 0.2
8 0.3000000000000004
```

## Números “reais” (ponto flutuante) - float

É o tipo usado para representar números reais

- Mas temos uma quantidade limitada de dígitos
- Não podemos representar qualquer número real...
  - $\pi$  tem infinitos dígitos após a vírgula
- E as operações podem ter erros de precisão

```
1 >>> 15.3 / 3
2 5.1000000000000005
3 >>> 5.3 - 2.1
4 3.1999999999999997
5 >>> 5.3 + 1.9
6 7.199999999999999
7 >>> 0.1 + 0.2
8 0.3000000000000004
9 >>> (15.6 / 3.0) * 3.0
```

## Números “reais” (ponto flutuante) - float

É o tipo usado para representar números reais

- Mas temos uma quantidade limitada de dígitos
- Não podemos representar qualquer número real...
  - $\pi$  tem infinitos dígitos após a vírgula
- E as operações podem ter erros de precisão

```
1 >>> 15.3 / 3
2 5.1000000000000005
3 >>> 5.3 - 2.1
4 3.1999999999999997
5 >>> 5.3 + 1.9
6 7.199999999999999
7 >>> 0.1 + 0.2
8 0.3000000000000004
9 >>> (15.6 / 3.0) * 3.0
10 15.600000000000001
```

As constantes são representadas por



## Números “reais” (ponto flutuante) - float

É o tipo usado para representar números reais

- Mas temos uma quantidade limitada de dígitos
- Não podemos representar qualquer número real...
  - $\pi$  tem infinitos dígitos após a vírgula
- E as operações podem ter erros de precisão

```
1 >>> 15.3 / 3
2 5.10000000000000005
3 >>> 5.3 - 2.1
4 3.1999999999999997
5 >>> 5.3 + 1.9
6 7.199999999999999
7 >>> 0.1 + 0.2
8 0.30000000000000004
9 >>> (15.6 / 3.0) * 3.0
10 15.600000000000001
```

As constantes são representadas por

- notação decimal: 4.2, 3.1, -7.8

## Números “reais” (ponto flutuante) - float

É o tipo usado para representar números reais

- Mas temos uma quantidade limitada de dígitos
- Não podemos representar qualquer número real...
  - $\pi$  tem infinitos dígitos após a vírgula
- E as operações podem ter erros de precisão

```
1 >>> 15.3 / 3
2 5.1000000000000005
3 >>> 5.3 - 2.1
4 3.1999999999999997
5 >>> 5.3 + 1.9
6 7.199999999999999
7 >>> 0.1 + 0.2
8 0.3000000000000004
9 >>> (15.6 / 3.0) * 3.0
10 15.600000000000001
```

As constantes são representadas por

- notação decimal: 4.2, 3.1, -7.8
  - mesmo se for zero após o ponto, i.e., 2.0

## Números “reais” (ponto flutuante) - float

É o tipo usado para representar números reais

- Mas temos uma quantidade limitada de dígitos
- Não podemos representar qualquer número real...
  - $\pi$  tem infinitos dígitos após a vírgula
- E as operações podem ter erros de precisão

```
1 >>> 15.3 / 3
2 5.1000000000000005
3 >>> 5.3 - 2.1
4 3.1999999999999997
5 >>> 5.3 + 1.9
6 7.199999999999999
7 >>> 0.1 + 0.2
8 0.3000000000000004
9 >>> (15.6 / 3.0) * 3.0
10 15.600000000000001
```

As constantes são representadas por

- notação decimal: 4.2, 3.1, -7.8
  - mesmo se for zero após o ponto, i.e., 2.0
- notação científica: 2e3, 2e-3, -1e-4

## Números inteiros - int

É o tipo usado para representar números inteiros

## Números inteiros - int

É o tipo usado para representar números inteiros

- Com qualquer quantidade de dígitos

## Números inteiros - int

É o tipo usado para representar números inteiros

- Com qualquer quantidade de dígitos

As constantes são representadas por

## Números inteiros - int

É o tipo usado para representar números inteiros

- Com qualquer quantidade de dígitos

As constantes são representadas por

- Números inteiros na base decimal (sem ponto)

## Números inteiros - int

É o tipo usado para representar números inteiros

- Com qualquer quantidade de dígitos

As constantes são representadas por

- Números inteiros na base decimal (sem ponto)
  - Ex: 0, 2, -3, etc...



## Números inteiros - int

É o tipo usado para representar números inteiros

- Com qualquer quantidade de dígitos

As constantes são representadas por

- Números inteiros na base decimal (sem ponto)
  - Ex: 0, 2, -3, etc...
- Números inteiros na base binária precedidos por 0b

## Números inteiros - int

É o tipo usado para representar números inteiros

- Com qualquer quantidade de dígitos

As constantes são representadas por

- Números inteiros na base decimal (sem ponto)
  - Ex: 0, 2, -3, etc...
- Números inteiros na base binária precedidos por 0b
  - Ex: 0b11 é o número 3 em decimal

## Números inteiros - int

É o tipo usado para representar números inteiros

- Com qualquer quantidade de dígitos

As constantes são representadas por

- Números inteiros na base decimal (sem ponto)
  - Ex: 0, 2, -3, etc...
- Números inteiros na base binária precedidos por 0b
  - Ex: 0b11 é o número 3 em decimal
- Números inteiros na base octal precedidos por 0o

## Números inteiros - int

É o tipo usado para representar números inteiros

- Com qualquer quantidade de dígitos

As constantes são representadas por

- Números inteiros na base decimal (sem ponto)
  - Ex: 0, 2, -3, etc...
- Números inteiros na base binária precedidos por 0b
  - Ex: 0b11 é o número 3 em decimal
- Números inteiros na base octal precedidos por 0o
  - Ex: 0o11 é o número 9 em decimal

## Números inteiros - int

É o tipo usado para representar números inteiros

- Com qualquer quantidade de dígitos

As constantes são representadas por

- Números inteiros na base decimal (sem ponto)
  - Ex: 0, 2, -3, etc...
- Números inteiros na base binária precedidos por 0b
  - Ex: 0b11 é o número 3 em decimal
- Números inteiros na base octal precedidos por 0o
  - Ex: 0o11 é o número 9 em decimal
- Números inteiros na base hexadecimal precedidos por 0x

## Números inteiros - int

É o tipo usado para representar números inteiros

- Com qualquer quantidade de dígitos

As constantes são representadas por

- Números inteiros na base decimal (sem ponto)
  - Ex: 0, 2, -3, etc...
- Números inteiros na base binária precedidos por 0b
  - Ex: 0b11 é o número 3 em decimal
- Números inteiros na base octal precedidos por 0o
  - Ex: 0o11 é o número 9 em decimal
- Números inteiros na base hexadecimal precedidos por 0x
  - Ex: 0x11 é o número 17 em decimal

## Números inteiros - int

É o tipo usado para representar números inteiros

- Com qualquer quantidade de dígitos

As constantes são representadas por

- Números inteiros na base decimal (sem ponto)
  - Ex: 0, 2, -3, etc...
- Números inteiros na base binária precedidos por 0b
  - Ex: 0b11 é o número 3 em decimal
- Números inteiros na base octal precedidos por 0o
  - Ex: 0o11 é o número 9 em decimal
- Números inteiros na base hexadecimal precedidos por 0x
  - Ex: 0x11 é o número 17 em decimal

Essas três últimas são usadas em contextos mais específicos

## Números inteiros - int

É o tipo usado para representar números inteiros

- Com qualquer quantidade de dígitos

As constantes são representadas por

- Números inteiros na base decimal (sem ponto)
  - Ex: 0, 2, -3, etc...
- Números inteiros na base binária precedidos por 0b
  - Ex: 0b11 é o número 3 em decimal
- Números inteiros na base octal precedidos por 0o
  - Ex: 0o11 é o número 9 em decimal
- Números inteiros na base hexadecimal precedidos por 0x
  - Ex: 0x11 é o número 17 em decimal

Essas três últimas são usadas em contextos mais específicos

Note que 2.0 e 2 tem tipos diferentes (float e int)!



## Dividindo a pizza

Queremos dividir 2 pizzas igualmente entre 5 alunos

## Dividindo a pizza

Queremos dividir 2 pizzas igualmente entre 5 alunos

- E estamos com preguiça de fazer a conta...

## Dividindo a pizza

Queremos dividir 2 pizzas igualmente entre 5 alunos

- E estamos com preguiça de fazer a conta...

Vamos pedir ajuda para o Python!

## Dividindo a pizza

Queremos dividir 2 pizzas igualmente entre 5 alunos

- E estamos com preguiça de fazer a conta...

Vamos pedir ajuda para o Python!

```
1 >>> 16 / 5
2 3.2
```

## Dividindo a pizza

Queremos dividir 2 pizzas igualmente entre 5 alunos

- E estamos com preguiça de fazer a conta...

Vamos pedir ajuda para o Python!

```
1 >>> 16 / 5
2 3.2
```

Não dá para cada um comer 3.2 pedaços... Vamos tentar de novo!

## Dividindo a pizza

Queremos dividir 2 pizzas igualmente entre 5 alunos

- E estamos com preguiça de fazer a conta...

Vamos pedir ajuda para o Python!

```
1 >>> 16 / 5
2 3.2
```

Não dá para cada um comer 3.2 pedaços... Vamos tentar de novo!

```
1 >>> 16 // 5
2 3
```

## Dividindo a pizza

Queremos dividir 2 pizzas igualmente entre 5 alunos

- E estamos com preguiça de fazer a conta...

Vamos pedir ajuda para o Python!

```
1 >>> 16 / 5
2 3.2
```

Não dá para cada um comer 3.2 pedaços... Vamos tentar de novo!

```
1 >>> 16 // 5
2 3
```

// é o operador de divisão inteira

## Dividindo a pizza

Queremos dividir 2 pizzas igualmente entre 5 alunos

- E estamos com preguiça de fazer a conta...

Vamos pedir ajuda para o Python!

```
1 >>> 16 / 5
2 3.2
```

Não dá para cada um comer 3.2 pedaços... Vamos tentar de novo!

```
1 >>> 16 // 5
2 3
```

// é o operador de divisão inteira

Quantas fatias sobram?



## Dividindo a pizza

Queremos dividir 2 pizzas igualmente entre 5 alunos

- E estamos com preguiça de fazer a conta...

Vamos pedir ajuda para o Python!

```
1 >>> 16 / 5
2 3.2
```

Não dá para cada um comer 3.2 pedaços... Vamos tentar de novo!

```
1 >>> 16 // 5
2 3
```

// é o operador de divisão inteira

Quantas fatias sobram?

```
1 >>> 16 % 5
2 1
```

## Dividindo a pizza

Queremos dividir 2 pizzas igualmente entre 5 alunos

- E estamos com preguiça de fazer a conta...

Vamos pedir ajuda para o Python!

```
1 >>> 16 / 5
2 3.2
```

Não dá para cada um comer 3.2 pedaços... Vamos tentar de novo!

```
1 >>> 16 // 5
2 3
```

// é o operador de divisão inteira

Quantas fatias sobram?

```
1 >>> 16 % 5
2 1
```

% é o operador de resto da divisão inteira (ou módulo)

## Tipo dos resultados

E se nossas expressões misturarem `float` e `int`?

## Tipo dos resultados

E se nossas expressões misturarem `float` e `int`?

```
1 >>> 2.0 + 3
2 5.0
```

## Tipo dos resultados

E se nossas expressões misturarem `float` e `int`?

```
1 >>> 2.0 + 3
```

```
2 5.0
```

A expressão `x / y` sempre resulta em `float`

## Tipo dos resultados

E se nossas expressões misturarem `float` e `int`?

```
1 >>> 2.0 + 3
2 5.0
```

A expressão `x / y` sempre resulta em `float`

Expressões `x + y`, `x - y`, `x * y`, `x ** y`, `x // y` e `x % y`:

## Tipo dos resultados

E se nossas expressões misturarem `float` e `int`?

```
1 >>> 2.0 + 3
```

```
2 5.0
```

A expressão `x / y` sempre resulta em `float`

Expressões `x + y`, `x - y`, `x * y`, `x ** y`, `x // y` e `x % y`:

- Resultam em `int` se `x` e `y` são ambos `int`

## Tipo dos resultados

E se nossas expressões misturarem `float` e `int`?

```
1 >>> 2.0 + 3
```

```
2 5.0
```

A expressão `x / y` sempre resulta em `float`

Expressões `x + y`, `x - y`, `x * y`, `x ** y`, `x // y` e `x % y`:

- Resultam em `int` se `x` e `y` são ambos `int`
- Resultam em `float` se `x` ou `y` são `float`



## Tipo dos resultados

E se nossas expressões misturarem `float` e `int`?

```
1 >>> 2.0 + 3
2 5.0
```

A expressão `x / y` sempre resulta em `float`

Expressões `x + y`, `x - y`, `x * y`, `x ** y`, `x // y` e `x % y`:

- Resultam em `int` se `x` e `y` são ambos `int`
- Resultam em `float` se `x` ou `y` são `float`

Ex:

```
1 >>> 16 // 5
```

## Tipo dos resultados

E se nossas expressões misturarem `float` e `int`?

```
1 >>> 2.0 + 3
2 5.0
```

A expressão `x / y` sempre resulta em `float`

Expressões `x + y`, `x - y`, `x * y`, `x ** y`, `x // y` e `x % y`:

- Resultam em `int` se `x` e `y` são ambos `int`
- Resultam em `float` se `x` ou `y` são `float`

Ex:

```
1 >>> 16 // 5
```

## Tipo dos resultados

E se nossas expressões misturarem `float` e `int`?

```
1 >>> 2.0 + 3
2 5.0
```

A expressão `x / y` sempre resulta em `float`

Expressões `x + y`, `x - y`, `x * y`, `x ** y`, `x // y` e `x % y`:

- Resultam em `int` se `x` e `y` são ambos `int`
- Resultam em `float` se `x` ou `y` são `float`

Ex:

```
1 >>> 16 // 5
2 3
```

## Tipo dos resultados

E se nossas expressões misturarem `float` e `int`?

```
1 >>> 2.0 + 3
2 5.0
```

A expressão `x / y` sempre resulta em `float`

Expressões `x + y`, `x - y`, `x * y`, `x ** y`, `x // y` e `x % y`:

- Resultam em `int` se `x` e `y` são ambos `int`
- Resultam em `float` se `x` ou `y` são `float`

Ex:

```
1 >>> 16 // 5
2 3
3 >>> 16 // 5.0
```

## Tipo dos resultados

E se nossas expressões misturarem `float` e `int`?

```
1 >>> 2.0 + 3
2 5.0
```

A expressão `x / y` sempre resulta em `float`

Expressões `x + y`, `x - y`, `x * y`, `x ** y`, `x // y` e `x % y`:

- Resultam em `int` se `x` e `y` são ambos `int`
- Resultam em `float` se `x` ou `y` são `float`

Ex:

```
1 >>> 16 // 5
2 3
3 >>> 16 // 5.0
4 3.0
```

## Tipo dos resultados

E se nossas expressões misturarem `float` e `int`?

```
1 >>> 2.0 + 3
2 5.0
```

A expressão `x / y` sempre resulta em `float`

Expressões `x + y`, `x - y`, `x * y`, `x ** y`, `x // y` e `x % y`:

- Resultam em `int` se `x` e `y` são ambos `int`
- Resultam em `float` se `x` ou `y` são `float`

Ex:

```
1 >>> 16 // 5
2 3
3 >>> 16 // 5.0
4 3.0
5 >>> 16.0 % 5
```

## Tipo dos resultados

E se nossas expressões misturarem `float` e `int`?

```
1 >>> 2.0 + 3
2 5.0
```

A expressão `x / y` sempre resulta em `float`

Expressões `x + y`, `x - y`, `x * y`, `x ** y`, `x // y` e `x % y`:

- Resultam em `int` se `x` e `y` são ambos `int`
- Resultam em `float` se `x` ou `y` são `float`

Ex:

```
1 >>> 16 // 5
2 3
3 >>> 16 // 5.0
4 3.0
5 >>> 16.0 % 5
6 1.0
```

## Textos - str

O tipo que representa texto no Python é o `str`



## Textos - str

O tipo que representa texto no Python é o `str`

A constante é uma sequência de caracteres entre `"` ou `'`

## Textos - str

O tipo que representa texto no Python é o `str`

A constante é uma sequência de caracteres entre `"` ou `'`

- Ex: `"uma string em Python"`

## Textos - str

O tipo que representa texto no Python é o `str`

A constante é uma sequência de caracteres entre `"` ou `'`

- Ex: `"uma string em Python"`
- Ex: `'outra string em Python'`

## Textos - str

O tipo que representa texto no Python é o `str`

A constante é uma sequência de caracteres entre `"` ou `'`

- Ex: `"uma string em Python"`
- Ex: `'outra string em Python'`
- Escolha um estilo e use somente ele

## Textos - str

O tipo que representa texto no Python é o `str`

A constante é uma sequência de caracteres entre `"` ou `'`

- Ex: `"uma string em Python"`
- Ex: `'outra string em Python'`
- Escolha um estilo e use somente ele
- Precisa ser de uma única linha

## Textos - str

O tipo que representa texto no Python é o `str`

A constante é uma sequência de caracteres entre `"` ou `'`

- Ex: `"uma string em Python"`
- Ex: `'outra string em Python'`
- Escolha um estilo e use somente ele
- Precisa ser de uma única linha

Para textos com várias linhas, use `"""`

## Textos - str

O tipo que representa texto no Python é o `str`

A constante é uma sequência de caracteres entre `"` ou `'`

- Ex: `"uma string em Python"`
- Ex: `'outra string em Python'`
- Escolha um estilo e use somente ele
- Precisa ser de uma única linha

Para textos com várias linhas, use `"""`

Ex:

# Textos - str

O tipo que representa texto no Python é o `str`

A constante é uma sequência de caracteres entre `"` ou `'`

- Ex: `"uma string em Python"`
- Ex: `'outra string em Python'`
- Escolha um estilo e use somente ele
- Precisa ser de uma única linha

Para textos com várias linhas, use `"""`

Ex:

```
1 texto = """Esse é um texto com
2 várias linhas, na verdade com
3 três linhas."""
```



## Textos - str

O tipo que representa texto no Python é o `str`

A constante é uma sequência de caracteres entre `"` ou `'`

- Ex: `"uma string em Python"`
- Ex: `'outra string em Python'`
- Escolha um estilo e use somente ele
- Precisa ser de uma única linha

Para textos com várias linhas, use `"""`

Ex:

```
1 texto = """Esse é um texto com
2 várias linhas, na verdade com
3 três linhas."""
```

Veremos mais sobre `str` durante o curso!

## Verificando o tipo

Podemos descobrir o tipo usando `type()`

```
1 >>> type(2.0)
```

## Verificando o tipo

Podemos descobrir o tipo usando `type()`

```
1 >>> type(2.0)
2 <class 'float'>
```

## Verificando o tipo

Podemos descobrir o tipo usando `type()`

```
1 >>> type(2.0)
2 <class 'float'>
3 >>> type(2)
```

## Verificando o tipo

Podemos descobrir o tipo usando `type()`

```
1 >>> type(2.0)
2 <class 'float'>
3 >>> type(2)
4 <class 'int'>
```

# Verificando o tipo

Podemos descobrir o tipo usando `type()`

```
1 >>> type(2.0)
2 <class 'float'>
3 >>> type(2)
4 <class 'int'>
5 >>> type("2")
```

# Verificando o tipo

Podemos descobrir o tipo usando `type()`

```
1 >>> type(2.0)
2 <class 'float'>
3 >>> type(2)
4 <class 'int'>
5 >>> type("2")
6 <class 'str'>
```

# Verificando o tipo

Podemos descobrir o tipo usando `type()`

```
1 >>> type(2.0)
2 <class 'float'>
3 >>> type(2)
4 <class 'int'>
5 >>> type("2")
6 <class 'str'>
7 >>> x = 10 / 3
```



# Verificando o tipo

Podemos descobrir o tipo usando `type()`

```
1 >>> type(2.0)
2 <class 'float'>
3 >>> type(2)
4 <class 'int'>
5 >>> type("2")
6 <class 'str'>
7 >>> x = 10 / 3
8 >>> type(x)
```

# Verificando o tipo

Podemos descobrir o tipo usando `type()`

```
1 >>> type(2.0)
2 <class 'float'>
3 >>> type(2)
4 <class 'int'>
5 >>> type("2")
6 <class 'str'>
7 >>> x = 10 / 3
8 >>> type(x)
9 <class 'float'>
```

## Verificando o tipo

Podemos descobrir o tipo usando `type()`

```
1 >>> type(2.0)
2 <class 'float'>
3 >>> type(2)
4 <class 'int'>
5 >>> type("2")
6 <class 'str'>
7 >>> x = 10 / 3
8 >>> type(x)
9 <class 'float'>
10 >>> y = 10 // 3
```

# Verificando o tipo

Podemos descobrir o tipo usando `type()`

```
1 >>> type(2.0)
2 <class 'float'>
3 >>> type(2)
4 <class 'int'>
5 >>> type("2")
6 <class 'str'>
7 >>> x = 10 / 3
8 >>> type(x)
9 <class 'float'>
10 >>> y = 10 // 3
11 >>> type(y)
```

## Verificando o tipo

Podemos descobrir o tipo usando `type()`

```
1 >>> type(2.0)
2 <class 'float'>
3 >>> type(2)
4 <class 'int'>
5 >>> type("2")
6 <class 'str'>
7 >>> x = 10 / 3
8 >>> type(x)
9 <class 'float'>
10 >>> y = 10 // 3
11 >>> type(y)
12 <class 'int'>
```

## Verificando o tipo

Podemos descobrir o tipo usando `type()`

```
1 >>> type(2.0)
2 <class 'float'>
3 >>> type(2)
4 <class 'int'>
5 >>> type("2")
6 <class 'str'>
7 >>> x = 10 / 3
8 >>> type(x)
9 <class 'float'>
10 >>> y = 10 // 3
11 >>> type(y)
12 <class 'int'>
```

## Convertendo entre tipos

Podemos converter entre tipos usando `int()`, `float()` e `str()`

## Convertendo entre tipos

Podemos converter entre tipos usando `int()`, `float()` e `str()`

```
1 >>> str(3)
```



## Convertendo entre tipos

Podemos converter entre tipos usando `int()`, `float()` e `str()`

```
1 >>> str(3)
```

## Convertendo entre tipos

Podemos converter entre tipos usando `int()`, `float()` e `str()`

```
1 >>> str(3)
2 '3'
```

# Convertendo entre tipos

Podemos converter entre tipos usando `int()`, `float()` e `str()`

```
1 >>> str(3)
2 '3'
3 >>> str(3.5)
```

# Convertendo entre tipos

Podemos converter entre tipos usando `int()`, `float()` e `str()`

```
1 >>> str(3)
2 '3'
3 >>> str(3.5)
4 '3.5'
```

# Convertendo entre tipos

Podemos converter entre tipos usando `int()`, `float()` e `str()`

```
1 >>> str(3)
2 '3'
3 >>> str(3.5)
4 '3.5'
5 >>> float("3.5")
```

# Convertendo entre tipos

Podemos converter entre tipos usando `int()`, `float()` e `str()`

```
1 >>> str(3)
2 '3'
3 >>> str(3.5)
4 '3.5'
5 >>> float("3.5")
6 3.5
```

# Convertendo entre tipos

Podemos converter entre tipos usando `int()`, `float()` e `str()`

```
1 >>> str(3)
2 '3'
3 >>> str(3.5)
4 '3.5'
5 >>> float("3.5")
6 3.5
7 >>> float("3")
```

# Convertendo entre tipos

Podemos converter entre tipos usando `int()`, `float()` e `str()`

```
1 >>> str(3)
2 '3'
3 >>> str(3.5)
4 '3.5'
5 >>> float("3.5")
6 3.5
7 >>> float("3")
8 3.0
```



# Convertendo entre tipos

Podemos converter entre tipos usando `int()`, `float()` e `str()`

```
1 >>> str(3)
2 '3'
3 >>> str(3.5)
4 '3.5'
5 >>> float("3.5")
6 3.5
7 >>> float("3")
8 3.0
9 >>> int("3")
```

# Convertendo entre tipos

Podemos converter entre tipos usando `int()`, `float()` e `str()`

```
1 >>> str(3)
2 '3'
3 >>> str(3.5)
4 '3.5'
5 >>> float("3.5")
6 3.5
7 >>> float("3")
8 3.0
9 >>> int("3")
10 3
```

# Convertendo entre tipos

Podemos converter entre tipos usando `int()`, `float()` e `str()`

```
1 >>> str(3)
2 '3'
3 >>> str(3.5)
4 '3.5'
5 >>> float("3.5")
6 3.5
7 >>> float("3")
8 3.0
9 >>> int("3")
10 3
11 >>> int("3.5")
```

# Convertendo entre tipos

Podemos converter entre tipos usando `int()`, `float()` e `str()`

```
1 >>> str(3)
2 '3'
3 >>> str(3.5)
4 '3.5'
5 >>> float("3.5")
6 3.5
7 >>> float("3")
8 3.0
9 >>> int("3")
10 3
11 >>> int("3.5")
12 Traceback (most recent call last):
13   File "<stdin>", line 1, in <module>
14 ValueError: invalid literal for int() with base 10: '3.5'
```

# Convertendo entre tipos

Podemos converter entre tipos usando `int()`, `float()` e `str()`

```
1 >>> str(3)
2 '3'
3 >>> str(3.5)
4 '3.5'
5 >>> float("3.5")
6 3.5
7 >>> float("3")
8 3.0
9 >>> int("3")
10 3
11 >>> int("3.5")
12 Traceback (most recent call last):
13   File "<stdin>", line 1, in <module>
14 ValueError: invalid literal for int() with base 10: '3.5'
```

Veremos outras conversões de tipo durante o curso

## Arquivos .py

O terminal do Python é bem útil para:

## Arquivos .py

O terminal do Python é bem útil para:

- Pequenas tarefas

## Arquivos .py

O terminal do Python é bem útil para:

- Pequenas tarefas
- Testar ideias, códigos, etc



# Arquivos .py

O terminal do Python é bem útil para:

- Pequenas tarefas
- Testar ideias, códigos, etc

Para programas maiores utilizamos um (ou mais arquivos) `.py`

# Arquivos .py

O terminal do Python é bem útil para:

- Pequenas tarefas
- Testar ideias, códigos, etc

Para programas maiores utilizamos um (ou mais arquivos) `.py`

- Permite armazenar o código a ser executado

# Arquivos .py

O terminal do Python é bem útil para:

- Pequenas tarefas
- Testar ideias, códigos, etc

Para programas maiores utilizamos um (ou mais arquivos) `.py`

- Permite armazenar o código a ser executado
- Permite compartilhar código entre a equipe

# Arquivos .py

O terminal do Python é bem útil para:

- Pequenas tarefas
- Testar ideias, códigos, etc

Para programas maiores utilizamos um (ou mais arquivos) `.py`

- Permite armazenar o código a ser executado
- Permite compartilhar código entre a equipe
- Permite colaboração e reutilização

## Exemplo de um arquivo .py

`prog1.py` (a numeração de linha não faz parte do código)

## Exemplo de um arquivo .py

`prog1.py` (a numeração de linha não faz parte do código)

```
1 s0 = 10
2 v0 = 2
3 t = 8
4 a = 3
5 s = s0 + v0 * t + a * t ** 2 / 2
6 s
```

## Exemplo de um arquivo .py

`prog1.py` (a numeração de linha não faz parte do código)

```
1 s0 = 10
2 v0 = 2
3 t = 8
4 a = 3
5 s = s0 + v0 * t + a * t ** 2 / 2
6 s
```

Executando no terminal:

```
python prog1.py
```

## Exemplo de um arquivo .py

`prog1.py` (a numeração de linha não faz parte do código)

```
1 s0 = 10
2 v0 = 2
3 t = 8
4 a = 3
5 s = s0 + v0 * t + a * t ** 2 / 2
6 s
```

Executando no terminal:

```
python prog1.py
```

O que é impresso?



## Exemplo de um arquivo .py

`prog1.py` (a numeração de linha não faz parte do código)

```
1 s0 = 10
2 v0 = 2
3 t = 8
4 a = 3
5 s = s0 + v0 * t + a * t ** 2 / 2
6 s
```

Executando no terminal:

```
python prog1.py
```

O que é impresso?

- Nada...

## Imprimindo dados

No Python, podemos usar `print` para imprimir informações

# Imprimindo dados

No Python, podemos usar `print` para imprimir informações

Exemplo:

```
1 s0 = 10
2 v0 = 2
3 t = 8
4 a = 3
5 s = s0 + v0 * t + a * t ** 2 / 2
6 print(s)
```

# Imprimindo dados

No Python, podemos usar `print` para imprimir informações

Exemplo:

```
1 s0 = 10
2 v0 = 2
3 t = 8
4 a = 3
5 s = s0 + v0 * t + a * t ** 2 / 2
6 print(s)
```

Imprime `122.0`

## Imprimindo dados

Podemos passar vários valores para ser impresso

## Imprimindo dados

Podemos passar vários valores para ser impresso

- Podem ser variáveis

## Imprimindo dados

Podemos passar vários valores para ser impresso

- Podem ser variáveis
- constantes

## Imprimindo dados

Podemos passar vários valores para ser impresso

- Podem ser variáveis
- constantes
- ou até mesmo expressões



## Imprimindo dados

Podemos passar vários valores para ser impresso

- Podem ser variáveis
- constantes
- ou até mesmo expressões

```
print(s0, v0, t, a) imprime:
```

```
10 2 8 3
```

## Imprimindo dados

Podemos passar vários valores para ser impresso

- Podem ser variáveis
- constantes
- ou até mesmo expressões

```
print(s0, v0, t, a) imprime:
```

```
10 2 8 3
```

```
print(2, 1.3 + 2, "abc") imprime:
```

```
2 3.3 abc
```

## Imprimindo dados

Podemos passar vários valores para ser impresso

- Podem ser variáveis
- constantes
- ou até mesmo expressões

```
print(s0, v0, t, a) imprime:
```

```
10 2 8 3
```

```
print(2, 1.3 + 2, "abc") imprime:
```

```
2 3.3 abc
```

Por padrão, o `print` separa por espaço e termina a linha

## Imprimindo dados

Podemos passar vários valores para ser impresso

- Podem ser variáveis
- constantes
- ou até mesmo expressões

```
print(s0, v0, t, a) imprime:
```

```
10 2 8 3
```

```
print(2, 1.3 + 2, "abc") imprime:
```

```
2 3.3 abc
```

Por padrão, o `print` separa por espaço e termina a linha

- `print(2, 3, sep=',')` imprime `2,3` (e termina a linha)

## Imprimindo dados

Podemos passar vários valores para ser impresso

- Podem ser variáveis
- constantes
- ou até mesmo expressões

```
print(s0, v0, t, a) imprime:
```

```
10 2 8 3
```

```
print(2, 1.3 + 2, "abc") imprime:
```

```
2 3.3 abc
```

Por padrão, o `print` separa por espaço e termina a linha

- `print(2, 3, sep=',')` imprime `2,3` (e termina a linha)
- `print(2, 3, sep=',', end='')` imprime `2,3`

## Imprimindo dados

Podemos passar vários valores para ser impresso

- Podem ser variáveis
- constantes
- ou até mesmo expressões

```
print(s0, v0, t, a) imprime:
```

```
10 2 8 3
```

```
print(2, 1.3 + 2, "abc") imprime:
```

```
2 3.3 abc
```

Por padrão, o `print` separa por espaço e termina a linha

- `print(2, 3, sep=',')` imprime `2,3` (e termina a linha)
- `print(2, 3, sep=',', end='')` imprime `2,3`
  - sem terminar a linha

## Imprimindo dados

Podemos passar vários valores para ser impresso

- Podem ser variáveis
- constantes
- ou até mesmo expressões

```
print(s0, v0, t, a) imprime:
```

```
10 2 8 3
```

```
print(2, 1.3 + 2, "abc") imprime:
```

```
2 3.3 abc
```

Por padrão, o `print` separa por espaço e termina a linha

- `print(2, 3, sep=',')` imprime `2,3` (e termina a linha)
- `print(2, 3, sep=',', end='')` imprime `2,3`
  - sem terminar a linha
  - próximo `print` começa na mesma linha

## Voltando ao código

Esse programa tem um problema



## Voltando ao código

Esse programa tem um problema

```
1 s0 = 10
2 v0 = 2
3 t = 8
4 a = 3
5 s = s0 + v0 * t + a * t ** 2 / 2
6 print(s)
```

## Voltando ao código

Esse programa tem um problema

```
1 s0 = 10
2 v0 = 2
3 t = 8
4 a = 3
5 s = s0 + v0 * t + a * t ** 2 / 2
6 print(s)
```

Para calcular com outros valores é necessário alterar o arquivo!

## Lendo dados

Podemos ler dados do teclado com `input`

## Lendo dados

Podemos ler dados do teclado com `input`

Ex: `x = input()`

## Lendo dados

Podemos ler dados do teclado com `input`

Ex: `x = input()`

`input`

## Lendo dados

Podemos ler dados do teclado com `input`

Ex: `x = input()`

`input`

- lê uma linha inteira de texto do terminal (até o enter)

# Lendo dados

Podemos ler dados do teclado com `input`

Ex: `x = input()`

`input`

- lê uma linha inteira de texto do terminal (até o enter)
- devolve uma `string` com todo o texto

## Lendo dados

Podemos ler dados do teclado com `input`

Ex: `x = input()`

`input`

- lê uma linha inteira de texto do terminal (até o enter)
- devolve uma `string` com todo o texto

Podemos também colocar uma mensagem para o usuário:



## Lendo dados

Podemos ler dados do teclado com `input`

Ex: `x = input()`

`input`

- lê uma linha inteira de texto do terminal (até o enter)
- devolve uma `string` com todo o texto

Podemos também colocar uma mensagem para o usuário:

Ex: `x = input("Entre com o texto:")`

## Lendo dados

Podemos ler dados do teclado com `input`

Ex: `x = input()`

`input`

- lê uma linha inteira de texto do terminal (até o enter)
- devolve uma `string` com todo o texto

Podemos também colocar uma mensagem para o usuário:

Ex: `x = input("Entre com o texto:")`

Mas e se quisermos ler um `int` ou `float`?

## Lendo dados

Podemos ler dados do teclado com `input`

Ex: `x = input()`

`input`

- lê uma linha inteira de texto do terminal (até o enter)
- devolve uma `string` com todo o texto

Podemos também colocar uma mensagem para o usuário:

Ex: `x = input("Entre com o texto:")`

Mas e se quisermos ler um `int` ou `float`?

- usamos conversão de tipos!

# Versão final

```
1 s0 = float(input("Entre com s0: "))
2 v0 = float(input("Entre com v0: "))
3 t = float(input("Entre com t: "))
4 a = float(input("Entre com a: "))
5 s = s0 + v0 * t + a * t ** 2 / 2
6 print("Posição final: ", s)
```

# Versão final

```
1 s0 = float(input("Entre com s0: "))
2 v0 = float(input("Entre com v0: "))
3 t = float(input("Entre com t: "))
4 a = float(input("Entre com a: "))
5 s = s0 + v0 * t + a * t ** 2 / 2
6 print("Posição final: ", s)
```

Dica: `python prog1.py < entrada.txt > saida.txt`

# Versão final

```
1 s0 = float(input("Entre com s0: "))
2 v0 = float(input("Entre com v0: "))
3 t = float(input("Entre com t: "))
4 a = float(input("Entre com a: "))
5 s = s0 + v0 * t + a * t ** 2 / 2
6 print("Posição final: ", s)
```

Dica: `python prog1.py < entrada.txt > saida.txt`

- `<` redireciona a entrada padrão para o arquivo `entrada.txt`

# Versão final

```
1 s0 = float(input("Entre com s0: "))
2 v0 = float(input("Entre com v0: "))
3 t = float(input("Entre com t: "))
4 a = float(input("Entre com a: "))
5 s = s0 + v0 * t + a * t ** 2 / 2
6 print("Posição final: ", s)
```

Dica: `python prog1.py < entrada.txt > saida.txt`

- `<` redireciona a entrada padrão para o arquivo `entrada.txt`
- `>` redireciona a saída padrão para o arquivo `saida.txt`

# Versão final

```
1 s0 = float(input("Entre com s0: "))
2 v0 = float(input("Entre com v0: "))
3 t = float(input("Entre com t: "))
4 a = float(input("Entre com a: "))
5 s = s0 + v0 * t + a * t ** 2 / 2
6 print("Posição final: ", s)
```

Dica: `python prog1.py < entrada.txt > saida.txt`

- `<` redireciona a entrada padrão para o arquivo `entrada.txt`
- `>` redireciona a saída padrão para o arquivo `saida.txt`
- Você pode usar apenas um dos dois se quiser



# Versão final

```
1 s0 = float(input("Entre com s0: "))
2 v0 = float(input("Entre com v0: "))
3 t = float(input("Entre com t: "))
4 a = float(input("Entre com a: "))
5 s = s0 + v0 * t + a * t ** 2 / 2
6 print("Posição final: ", s)
```

Dica: `python prog1.py < entrada.txt > saida.txt`

- `<` redireciona a entrada padrão para o arquivo `entrada.txt`
- `>` redireciona a saída padrão para o arquivo `saida.txt`
- Você pode usar apenas um dos dois se quiser
- Muito útil para os laboratórios!

## Exercício

Faça um programa que calcula o valor de um investimento com saldo inicial  $s$  após  $n$  meses com uma taxa de juros anual de  $j$ .

- Não há depósitos ou retiradas do investimento
- A taxa de juros é dada em porcentagem sem o símbolo %.
  - Ex: 10 significa 10% ao ano.
- O juro é aplicado ao final do mês.